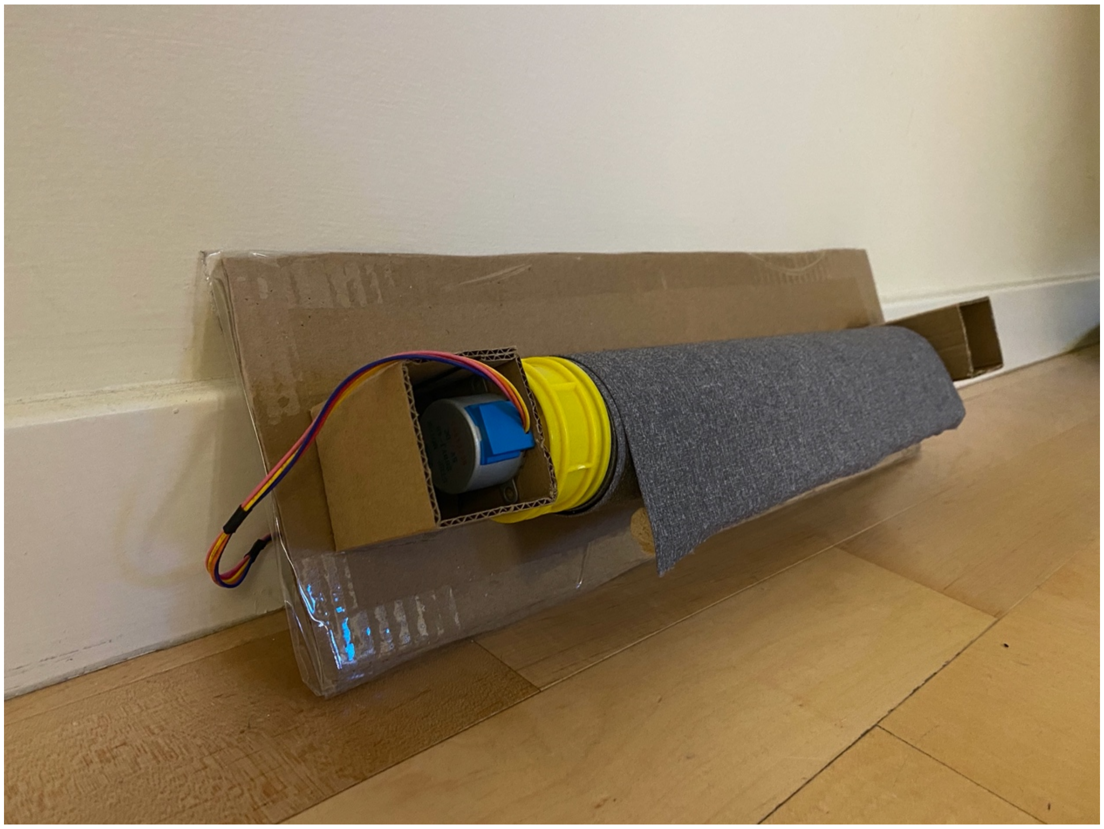
MECH 423 Project: Smart Blind



AJAYKUMAR MUDALIAR & KAWAI CHEN
ajay.selvamk@gmail.com & w0chen@eng.ucsd.edu

CONTENTS

ABSTRACT:

The objective of this project is to build a smart blind. This Blind system is capable of moving the roller shade up or down automatically by using Alexa voice commands. It can be put into day-night mode where the roller shade moves up or down depending on the ambient light level. This is done with an MSP430 board which controls the stepper motor and also samples the light sensor (photoresistor), and also a Raspberry Pi board, which is used to receiver voice commands via a microphone.

The sensor selection, motor selection, principle of operation, state system of machine, interconnectivity and communication between the two boards, calibration of the sensors and the overall circuit connections are given in the report. To summarize, we have sourced a suitable stepper motor, which is relatively light weight and has sufficient torque, to drive our roller shade. We utilize one MSP430 board and one Raspberry Pi board for our project. The Raspberry Pi is responsible for receiving voice command and sending a signal (i.e. up or down) to MSP430 board. The MSP430, with our Lab 3 stepper motor code and an external stepper motor driver, are used to drive the motor. The rotation amount and speed are both calibrated according to the length of the windows shade. For the automatic mode, the MSP430 board is constantly sampling data from a photoresistor. Once a certain light condition is met (Bright light, indirect light, or dark), the MSP430 will triger the Raspberry Pi to send a signal (i.e. up or down) back to the MSP430, which will then drive the stepper.

OBJECTIVE

We set out to prototype an Alexa enabled smart blind, which also has an automatic mode based on the ambient light intensity. Since smart blind is already very common, we intend to build one with minimal cost, while having as many functionalities as possible. For instance, the Ikea smart blind, which can only be controlled by a wireless remote, costs around $200. Our solution offer Alexa integration, automatic mode, while only costing around $60.

RATIONALE

Having curtains or blinds over the windows at night can save a lot of heat while having them open during the day while the sun is bright, can allow your house to be naturally heated. Having the blinds closed during the day or open at night can also be a good indication to burglars that you are away from your house. Also, the Alexa integration turns a simple blind into a smart home system that can be operated from anywhere in the house.

SUMMARY OF FUNCTIONAL REQUIREMENTS

FR#1 Selecting and Driving a Stepper Motor: In this section, we select a stepper motor for driving the roller shade tube. We successfully drive the motor with the stepper motor code from Lab 3.

FR#2 Photoresistor Calibration: In this section, we explain the circuit used to sense light intensity. We calibrate the light sensor and show the result.

FR#3 Robustness of Light Sensing Algorithm: In this section, we come up with an algorithm to prevent the roller shade from moving due to sudden light intensity change.

FR#4 Controlling Raspberry Pi GPIO with Amazon Alexa: In this section, we explain how we deploy Alexa on a Raspberry Pi, and how we use Alexa to control the GPIO pins on the Raspberry Pi board.

FR#5 Integrating all Components: In this section, we will explain how we integrate all components. We show the circuit and corresponding schematic.

Functional Requirements	% Effort	Responsible Person
FR#1: Selecting and Driving a Stepper Motor	20	AJ
FR#2: Photoresistor Calibration	20	AJ
FR#3: Robustness of Light Sensing Algorithm	20	AJ & KW
FR#4: Controlling Raspberry Pi GPIO with Amazon Alexa	20	KW
FR#5: Integrating all Components	20	KW

FUNCTIONAL REQUIREMENT #1: SELECTING AND DRIVING A STEPPER MOTOR

APPROACH AND DESIGN

The objective of this functional requirement is simple, to select a stepper motor that suits our need and to drive it with our stepper motor codes in Lab 3. Since in practice, the blind will be hanging on the wall, we need to use a relatively light weight stepper motor which has enough torque to drive the blind roller tube up and down. Our final decision is to use the stepper motor 28BYJ-48 and the stepper motor driver ULN2003, which are shown in Fig. 1.

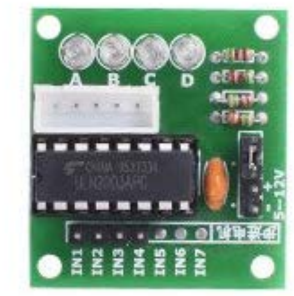


Fig. 1(a): Stepper Motor Driver ULN2003

Fig. 1(b): Stepper Motor 28BYJ-48

Note that the stepper motor driver board above function exactly the same as the green board we have in Lab 3. We can therefore reuse the stepper motor code.

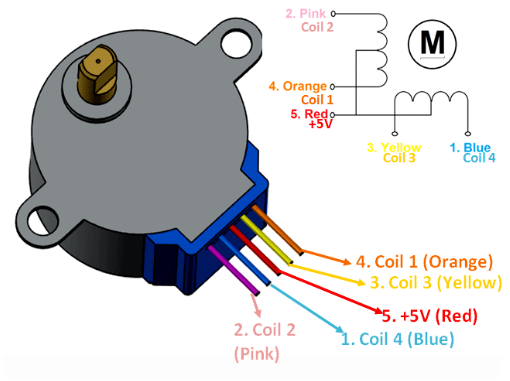
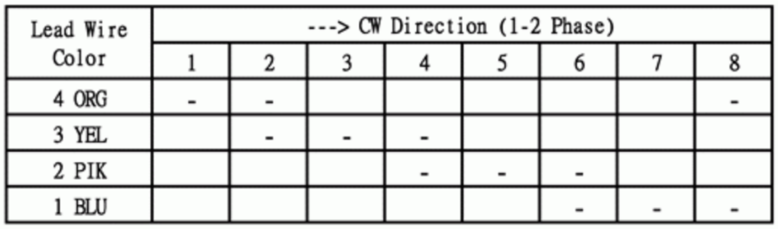


Fig. 2(a): Wiring of the Stepper Motor



Lead Wire Color	---> CW Direction (1-2 Phase)							
	1	2	3	4	5	6	7	8
4 ORG	-	-						-
3 YEL		-	-	-				
2 PIK				-	-	-		
1 BLU						-	-	-

Fig. 2(b): Sequence to Rotate in Clockwise Direction

Fig. 2(a) and (b) shows the wiring diagram of the stepper motor and the corresponding pin excitation order. Although the chosen stepper motor has 5 physical wires, the red wires is directly connected the power supply. Meaning that the stepper is still a 4-phase unipolar stepper motor.

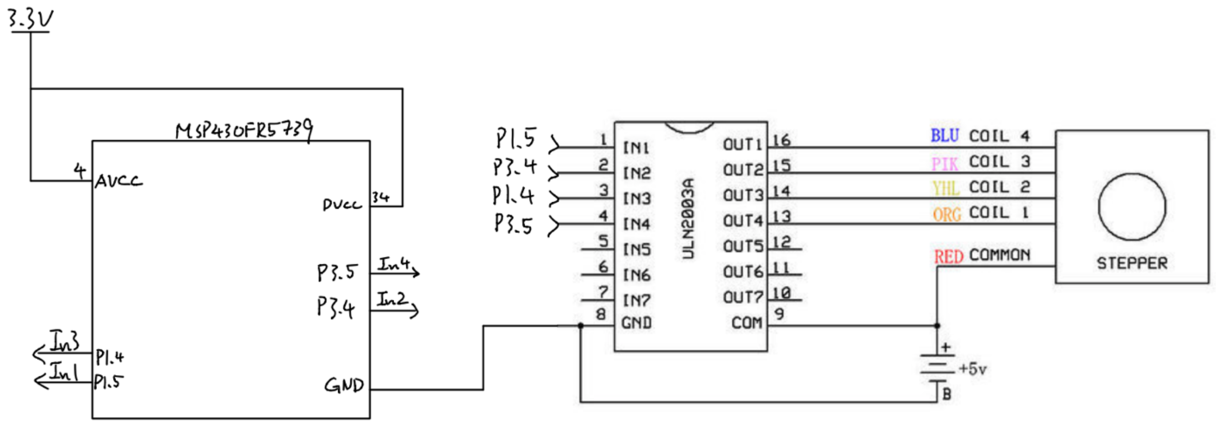
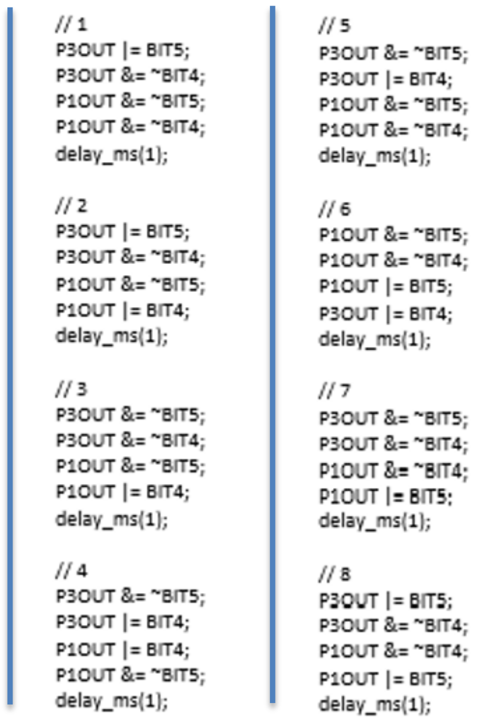


Fig. 3: Schematic of MSP430FR5739 and Stepper Motor Driver

Fig. 3 above shows the connection between the MCU and the stepper motor driver, the code below is used to drive the motor.



```
// 1
P3OUT |= BIT5;
P3OUT &= ~BIT4;
P1OUT &= ~BIT5;
P1OUT &= ~BIT4;
delay_ms(1);

// 2
P3OUT |= BIT5;
P3OUT &= ~BIT4;
P1OUT &= ~BIT5;
P1OUT |= BIT4;
delay_ms(1);

// 3
P3OUT &= ~BIT5;
P3OUT &= ~BIT4;
P1OUT &= ~BIT5;
P1OUT |= BIT4;
delay_ms(1);

// 4
P3OUT &= ~BIT5;
P3OUT |= BIT4;
P1OUT |= BIT4;
P1OUT &= ~BIT5;
delay_ms(1);
```

```
// 5
P3OUT &= ~BIT5;
P3OUT |= BIT4;
P1OUT &= ~BIT5;
P1OUT &= ~BIT4;
delay_ms(1);

// 6
P1OUT &= ~BIT5;
P1OUT &= ~BIT4;
P1OUT |= BIT5;
P3OUT |= BIT4;
delay_ms(1);

// 7
P3OUT &= ~BIT5;
P3OUT &= ~BIT4;
P1OUT &= ~BIT4;
P1OUT |= BIT5;
delay_ms(1);

// 8
P3OUT |= BIT5;
P3OUT &= ~BIT4;
P1OUT &= ~BIT4;
P1OUT |= BIT5;
delay_ms(1);
```

In order to ensure that the motor won't go up when our blind is at the top position, and down when it is at the bottom position, we have implemented a state machine, Fig. 4 below illustrates the idea.

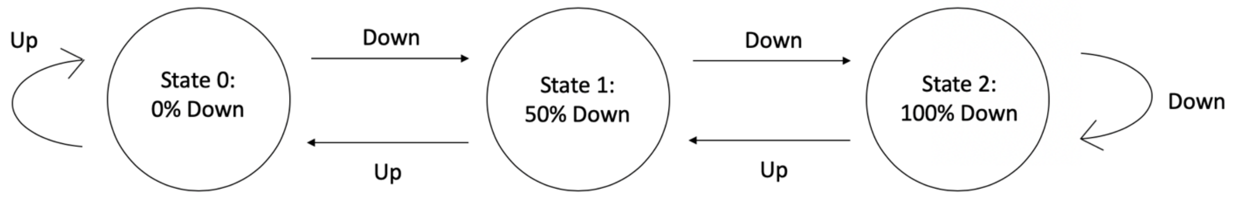


Fig. 4: State Machine for Driving the Roller Tube Position

INPUTS AND OUTPUTS

The input for driving the motor clockwise and counter clockwise are the switches on MSP430, namely S1, which is connected to P4.0 and S2, which is connected to P4.1. The output is simply the rotation of the motor shaft in either direction. Apart from that, we also use the LEDs on MSP430 to indicate the corresponding states. Namely, when our machine is in state 0, no LEDs will light up. When it is in state 1, LED1 and 2, which are connected to PJ.0 and PJ.1, will light up. When it is in state 2, in addition to LED1 and 2, LED3 and 4, which are connected to PJ.2 and PJ.3, will also light up. The diagram below shows the schematic of the switches and the LEDs.

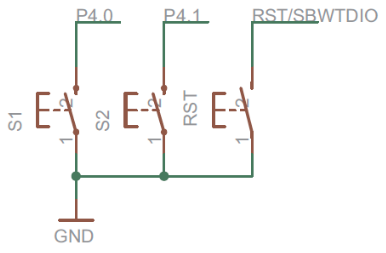


Fig. 5(a): Input Schematic

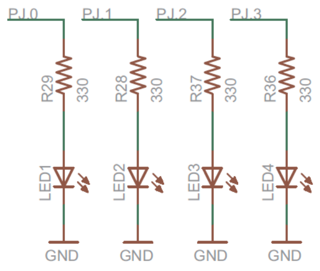
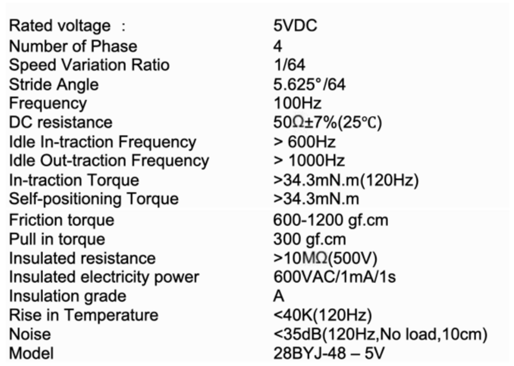


Fig. 5(b): Output Schematic

PARAMETERS



Rated voltage :	5VDC
Number of Phase	4
Speed Variation Ratio	1/64
Stride Angle	5.625°/64
Frequency	100Hz
DC resistance	50Ω±7%(25℃)
Idle In-traction Frequency	> 600Hz
Idle Out-traction Frequency	> 1000Hz
In-traction Torque	>34.3mN.m(120Hz)
Self-positioning Torque	>34.3mN.m
Friction torque	600-1200 gf.cm
Pull in torque	300 gf.cm
Insulated resistance	>10MΩ(500V)
Insulated electricity power	600VAC/1mA/1s
Insulation grade	A
Rise in Temperature	<40K(120Hz)
Noise	<35dB(120Hz,No load,10cm)
Model	28BYJ-48 – 5V

Fig. 6(a): Datasheet for Stepper Motor

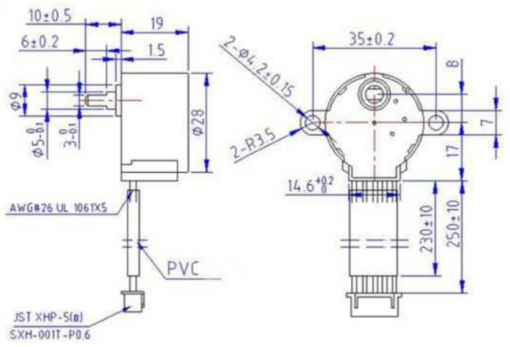


Fig. 6(b): Stepper Motor Dimension

As seen in Fig. 6, the stepper motor is capable of doing 360/(5.625/64) = 4096 steps/rev for half stepping. This motor is chosen because it is relatively light weight and small, as seen in Fig. 6(b) and has a relatively high in-traction torque and self-positioning torque, which we need in order to hold the blind roller tube's position.

TESTING AND RESULTS

In order to test our motor code, we attached a wheel on the motor and marked a specific position on the wheel with a marker. We observe the rotation direction, speed and position. Note that the speed of the motor is the least important factor in our design, as we only need to move the blind at one specific speed setting.

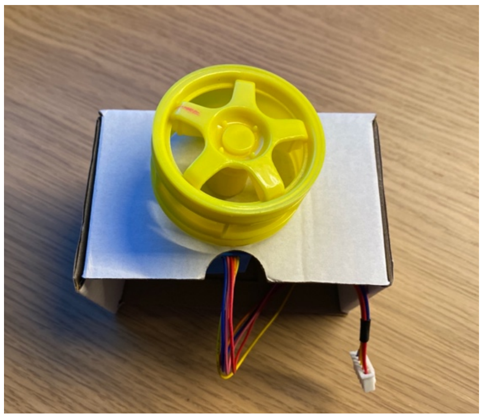


Fig.7(a): Testing Wheel Top View

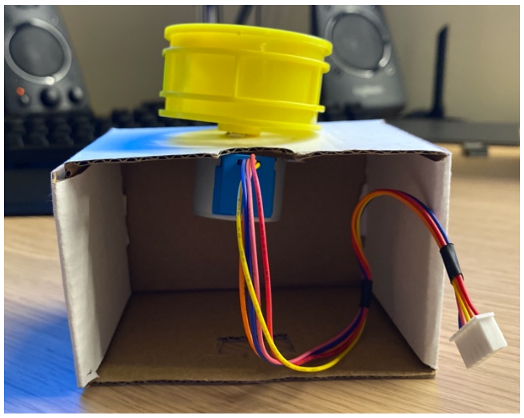


Fig.7(b): Testing Wheel Side View

Fig. 7: Testing Wheel

After we pressed switch S1, the wheel will rotate in clockwise direction for one full revolution, while S2 will trigger counterclockwise rotation.

FUNCTIONAL REQUIREMENT #2: PHOTORESISTOR CALIBRATION

APPROACH AND DESIGN

In order to move the blind according to the light intensity, we need a photoresistor (a light sensor). Figure 8 shows the chosen photoresistor and the corresponding connection with MSP430.

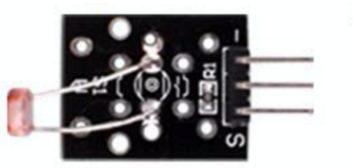


Fig. 8(a): Photoresistor

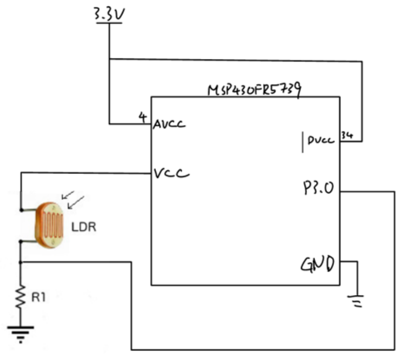


Fig. 8(b): Photoresistor Circuit

The voltage across the photoresistor will vary according to the amount of light shone on it. The value for the pulldown resistor (R1) used is 10kOhms. The voltage (or resistance) across the photoresistor is then connected to P3.0 on MSP430, which corresponds to input channel A12. We then use the built-in ADC converter to convert the voltage across the photoresistor to a digital reading.

INPUTS AND OUTPUTS

The input in this FR is simply the light intensity in the room, which is then converted into a voltage value by the photoresistor. The voltage across the photoresistor is then converted into a digital reading by the ADC. Note that since we bit shift the reading from 10-bits to 8-bits, the output of the photoresistor ranges from 0-255.

PARAMETERS

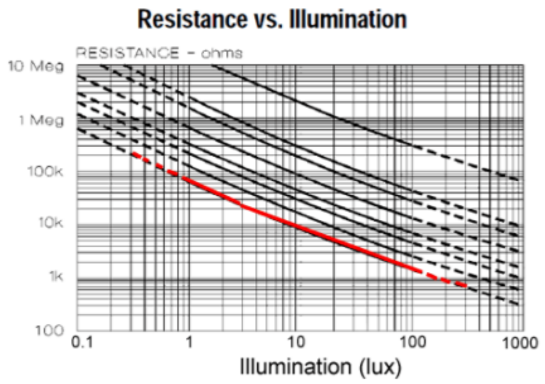


Fig. 9(a): Photoresistor's Resistance VS Illumination

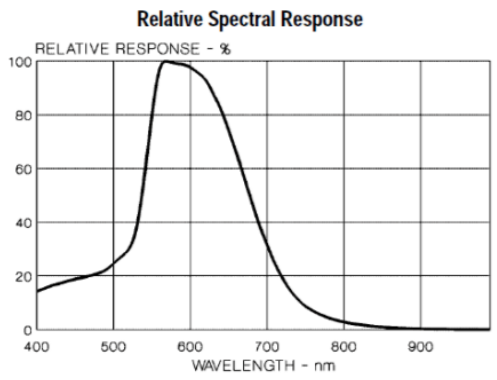


Fig. 9(b): Operating Wavelength of the Photoresistor

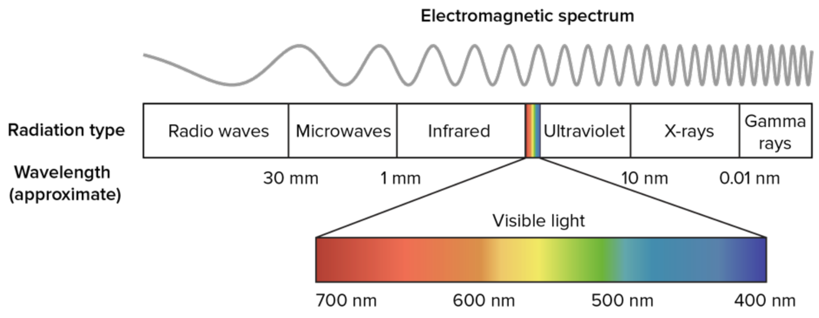


Fig. 10: Visible Light Spectrum

Fig. 9(a) gives us a basic idea of the relationship between the voltage across the photoresistor and the corresponding brightness. We can use the data as our baseline values when testing the photoresistor. Fig. 9(b) shows that blue light is not as effective as green or yellow light for our photoresistor, but that will not be a problem because the wavelength of sunlight falls within our optimal operating range for the photoresistor. As seen in Fig. 10, sunlight (or warm yellow light in general) typically falls within 550nm to 600nm, which provides the peak response rate for our photoresistor.

TESTING AND RESULTS

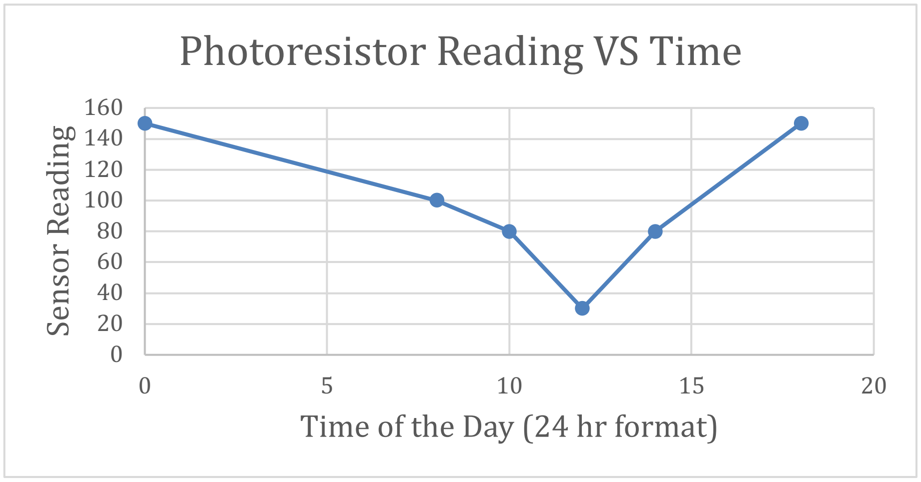


Fig. 11 Calibration of Photoresistor

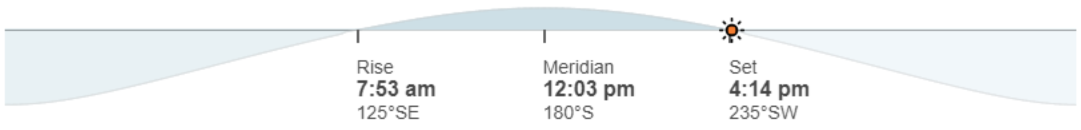


Fig. 12: Typical Sunrise and Sunset Time for a Winter Day

Fig. 11 shows our calibration result for a normal cloudy day in Vancouver. As seen in the graph, the reading increases as the brightness decreases and vice versa. From the graph, we pick two values as our threshold, 30 and 150. The blind should be fully up when the reading is equal or greater than 30, fully down when the reading is equal to or greater than 150, and halfway when the reading is in between. In the following section, we will propose a method to prevent the motor from rotating due to sudden change in light intensity.

FUNCTIONAL REQUIREMENT #3: ROBUSTNESS OF LIGHT SENSING ALGORITHM

APPROACHES AND DESIGN

In order to prevent the interference from a sudden change in lighting intensity, we have implemented an algorithm, shown in the following code. We have also implemented a state machine for light intensity, which is shown in Fig. 13.

```
if (state == 0 && output > 30 && pre_output > 30) {
        count_down_30 = count_down_30 + 1;
}
else if (state == 0 && output < 30) {
        count_down_30 = 0;
}
if ((state == 0) && (count_down_30 >= 30000)){
        // Motor Down to 50%
        P1OUT |= BIT7;
        //delay_ms(delay);
        for (i = 0; i<100; i++)
            {
               _NOP();
            }
        P1OUT &= BIT7;
        state = 1;
        count_down_30 = 0;
}
```

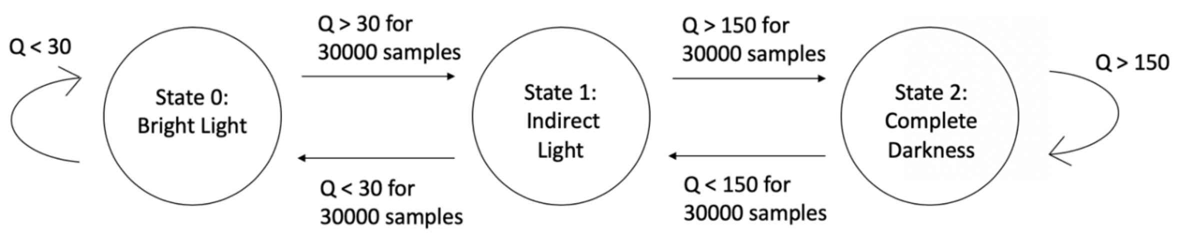


Fig. 13: State Machine for Light Intensity Reading

Note that the state machine shown in Fig. 13 is different than the one in FR #1. To summarize the algorithm, the code above counts the number of times that a particular threshold is exceeded. For example, when we are at state 0 (i.e. photoresistor is under bright light), and the photoresistor reading is greater than 30 for 30000 samples, the system will enter state 1, and also trigger pin P1.7. (which will be used as a up signal for our motor, will be explained in FR#5)

INPUTS AND OUTPUTS

The input of this FR is again just the light intensity of our room. The output is the state (0, 1, or 2) and also the status of the pin 1.7 (High or Low).

PARAMETERS

The only parameter that needed to be adjust in this FR is the counting threshold, which is set to 30000 in our design. A sample size of 30000 translates to around 5s in real life, which is used for demonstration purpose. In a practical design, we will increase that to around 1 min (or longer depends on need).

TESTING AND RESULTS

In order to test our algorithm, we use the LEDs on the MSP430, the code below illustrates the idea.

```
if (P1IES & BIT7){

        PJOUT |= BIT0;

}
```

Note that P1.7 is set as input, global interrupt is also enabled. Whenever our algorithm switches from state 0 to state 1, pin 1.7 interrupt service will be trigger, which will turn on LED 1 that is connected to PJ.0.

FUNCTIONAL REQUIREMENT #4: CONTROLLING RASPBERRY PI GPIO WITH AMAZON ALEXA

APPROACH AND DESIGN

In order to implement the Amazon Alexa functionality, we need a Raspberry Pi since it has internet capability. Note that we are not using an external Alexa speaker, despite it being a more straight forward approach, as this will increase the overall cost of our project significantly. In our design, our Raspberry Pi is the Alexa speaker, which can also communicate with our MSP430. Fig. 14 below shows the components used. Note that once the Raspberry Pi is powered up and properly set up, the keyboard, mouse and monitor can all be disconnected.

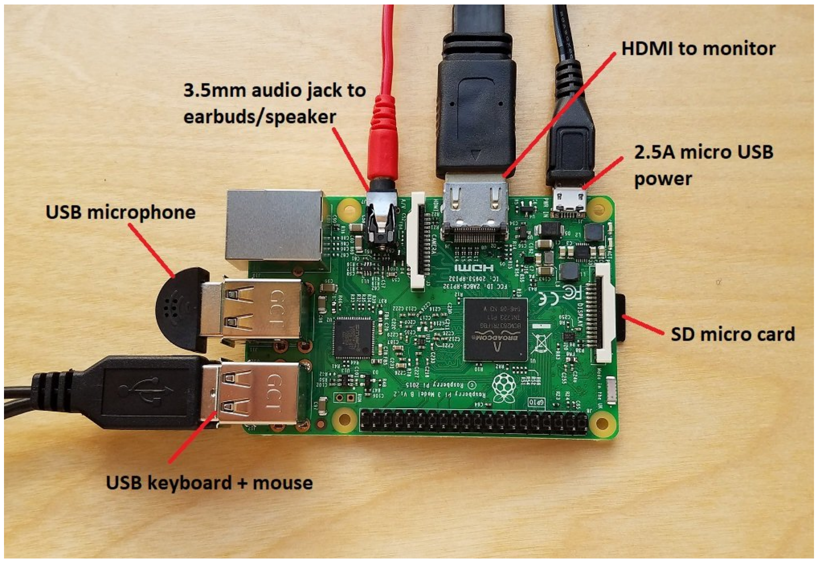


Fig. 14: Raspberry Pi Peripheral Connection

Installing Alexa on Raspberry Pi is fairly straight forward, as it is very well documented on the official Amazon Alexa Developer website https://developer.amazon.com/docs/alexa-voice-service/set-up-raspberry-pi.html. We therefore won't discuss the details here. The most challenging portion of this FR is to come up with a method that allow the voice command to be received by the Raspberry Pi. To accomplish this task, we use two web services, IFTTT and PubNub. Fig. 15 shows the basic flow chart.

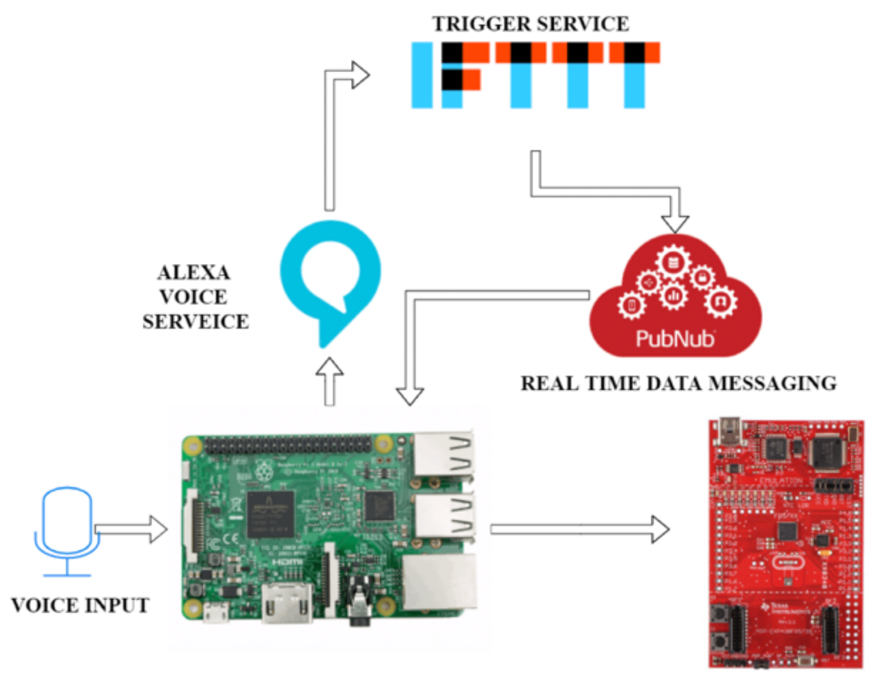


Fig. 15: Alexa Command Flow Chart

IFTTT, which is an abbreviation for If This Then That, is a programmable web service which can be connected to Alexa Voice Service. Whenever Alexa hear a specific phrase, IFTTT will then send a web request (i.e. to publish a message) through an encrypted URL hosted by the other web service, PubNub. By knowing the keys to the encrypted URL, the Raspberry Pi can therefore receive the corresponding web request. The following section will provide a more in depth explanation.

INPUTS AND OUTPUTS

The initial input of this FR is simply a voice command, which should be in the following format,

"Alexa, trigger move blind up"

"Alexa, trigger move blind down"

Whenever IFTTT hear the phrases above, it will then publish the trigger message (in json format) through an encrypted URL hosted by PubNub. We need to run a Python code on the Raspberry Pi to capture the message, the following code shows the basic concept.

```
pub_key = "pub-c-abf47b2b-b5d7-49ed-9329-f32488ead2c2"

sub_key = "sub-c-65f73da2-0e28-11ea-969d-52dd774e953e"

pubnub = Pubnub(publish_key=pub_key,subscribe_key=sub_key)

pubnub.subscribe(channels='alexaTrigger', callback=callback, error=callback,
reconnect=reconnect, disconnect=disconnect)



if(controlCommand["trigger"] == "blind" and controlCommand["status"] == 1):

        print "blind is moving up"

        GPIO.output(3, True)

        time.sleep (1)

        GPIO.output(3, False)

        print "blind is up"
```

As seen from the code above, the first 4 lines correspond to the set-up process to receive message from PubNub. Note that whenever a command that contains the phrase “blind” as the trigger, and the command status is 1, GPIO Pin 3 will be triggered. We have programmed two separate triggers in IFTTT, status == 1, which corresponds to the command “trigger move blind up”, triggers GPIO Pin 3 on raspberry pi, while status == 0, which corresponds to “trigger move blind down”, triggers GPIO Pin 4.

TESTING AND RESULTS

In order to test our method, we connect our Raspberry Pi board to our MSP430. The schematic below illustrates the connection.

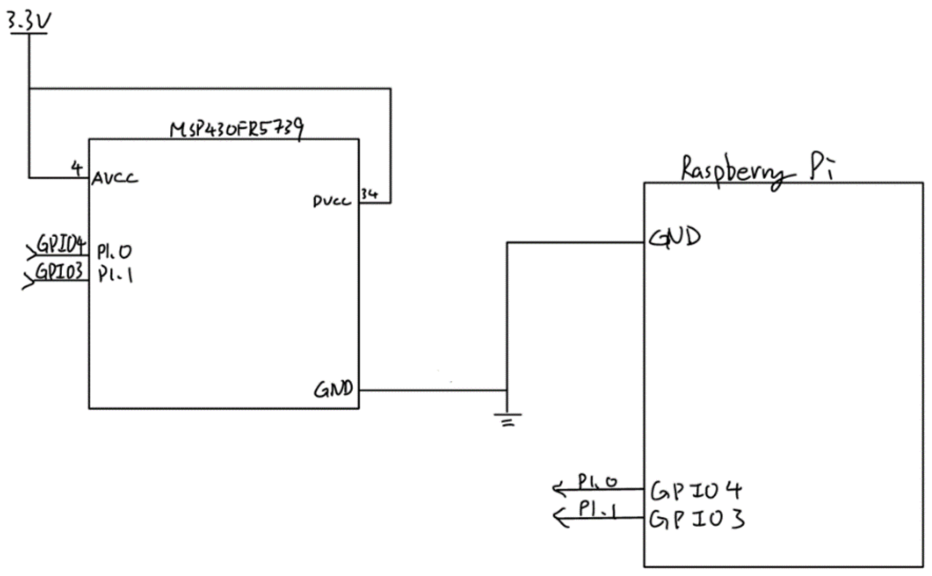


Fig. 16: Connection between Raspberry Pi and MSP430

The goal of this FR is to trigger an LED on the MSP430 on and off using voice command. The simple code below illustrates the idea,

```
        if (P1IES & BIT1){

                PJOUT |= BIT0;

        }
```

Note that P1.0 and P1.1 are both set as input, global interrupt is also enabled. Since GPIO Pin 3 is connected to P1.1 on MSP430, whenever Pin 3 is set from low to high, an interrupt service is triggered on P1.1, which will turn on LED1 on MSP430 by the code above.

FUNCTIONAL REQUIREMENT #5: INTEGRATING ALL COMPONENTS

APPROACH AND DESIGN

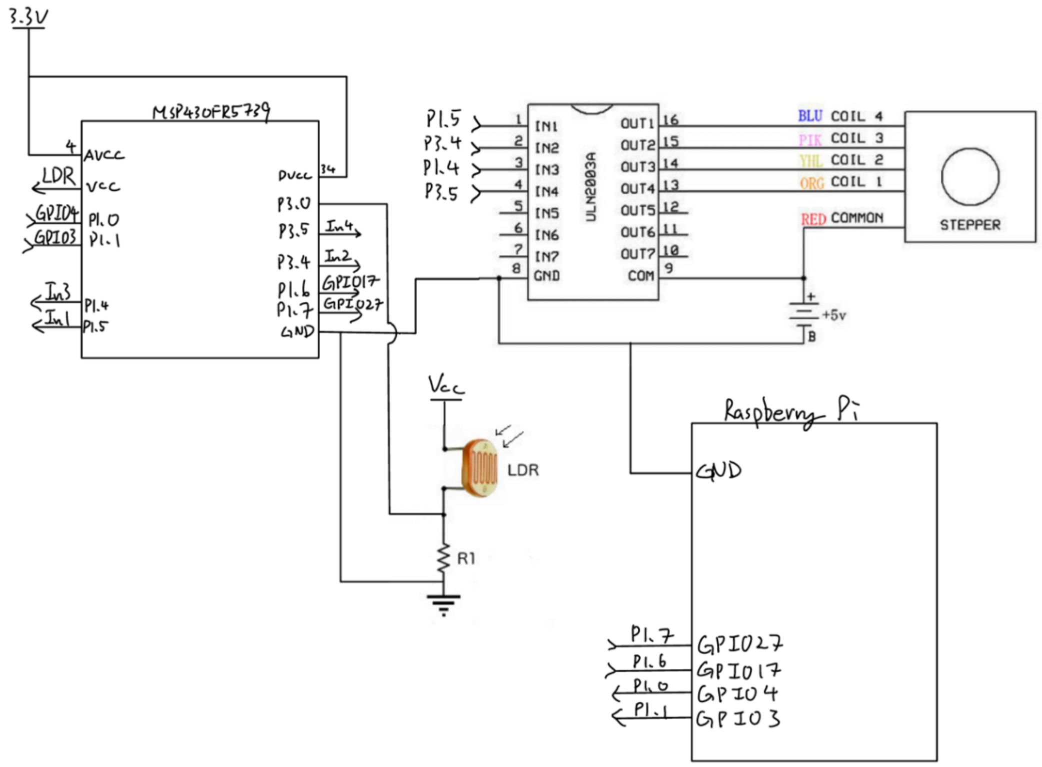


Fig. 17: Overall Schematic

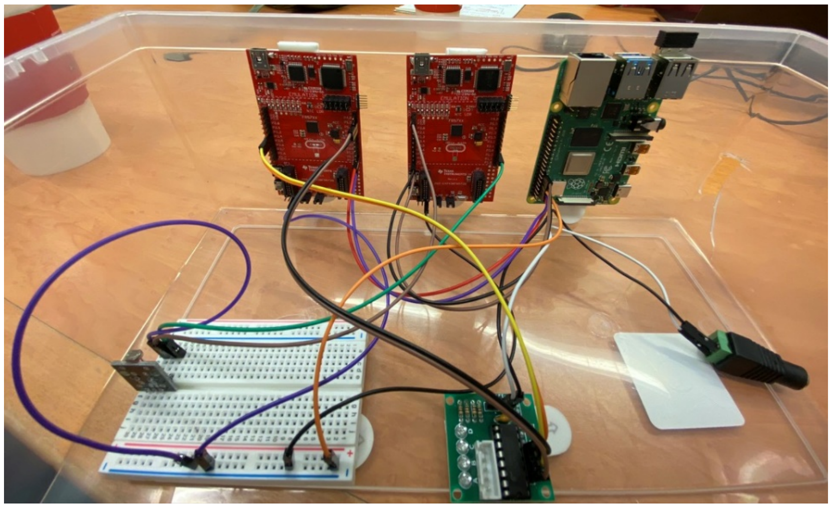


Fig. 18: Overall Connection

Fig. 17 and 18 shows the overall schematic and connection of our final design. Starting from the MSP430 board, P1.4, P1.5, P3.4 and P3.5 are used to drive the stepper motor, as explained in FR#1. Whenever a up command is received on the Raspberry Pi, GPIO 3 will be triggered, which will in turn trigger P1.1 on MSP430, which will then trigger the stepper motor to turn clockwise for 1 full revolution. GPIO 4 which is connected to P1.0 operates exactly the same, except in counter clockwise direction.

As explained in FR#3, whenever a there is a light intensity state change, P1.7 (corresponds to clockwise rotation) or P1.6 (corresponds to counterclockwise rotation) will be triggered. Note that GPIO 17 and 27 on Raspberry Pi are set as input, while P1.7 and P1.6 on the MSP430 are set as output. When the light intensity algorithm sends an up command, P1.7 will be triggered, which will in turn trigger GPIO 27. When an interrupt is detected on GPIO 27, our code on Raspberry Pi will then trigger GPIO 3 to rotate the motor. The other pin, P1.6, works exactly the same, except for counterclockwise rotation.

INPUTS AND OUTPUTS

For our final system, the inputs are the voice command and the light intensity reading from the photoresistor, the output is the rotation of the motor shaft, clockwise or counter clockwise.

PARAMETERS

The only parameter that needs to be tuned in this section is the blind length and the unroll portion. Fig. 18 below illustrates the idea.

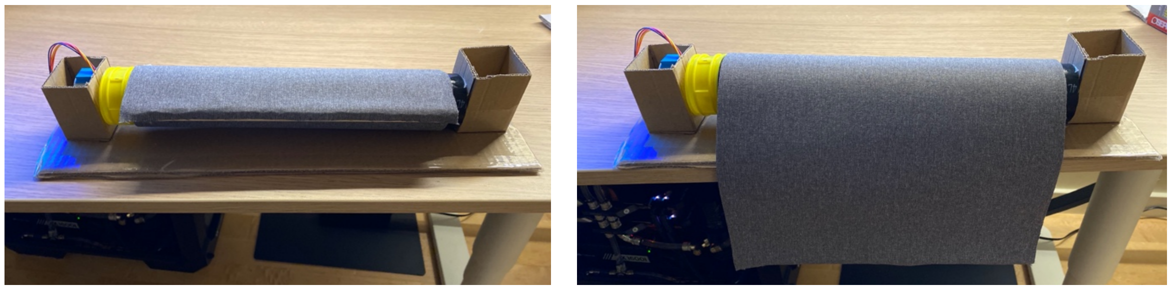


Fig. 18: One Full Revolution of Stepper Motor and the Portion of Blind Unrolled

For demonstration purpose, we found that one revolution is adequate. We therefore program the stepper motor to run one full revolution per trigger.

TESTING AND RESULTS

To ensure our final product work as intended, we test the blind under different lighting condition, with the inference of sudden light intensity change. We also make sure that the blind does not roll up when it is at its top position (state 0) and does not roll down when it is at the bottom position (state 2).

SYSTEM EVALUATION:

From the tests conducted it is noticed that the system is very robust in every aspect i.e. the voice activation always works, the system is able to hold the position of the blind successfully, the MSP430 drives the stepper successful in both directions when the light level changes accordingly and the state machine prevents the blind to move further up than highest position or further down when in the lowest position.

However, one of the observations while conducting tests were that the light levels in each area was different and the system had to be re-calibrated by making adjustments to the threshold light level in the code. This was not done automatically as the light levels would vary all the time. To overcome this, the system could be connected to the internet to extract information about the day and make these calculations and update the code automatically.

REFLECTION:

The project was completed successfully. However, in order to reach this point, loads of challenges and mistakes needed to be corrected to make the system work. A few of the major challenges and how they were overcome are mentioned below:

- In the initial stage, the plan was to use a 12V DC motor. While trying to find one that could roll up and roll down a blind (a tubular DC motor) none were found that could provide sufficient holding torque to hold the blind position. As if a tubular motor was found it was a part of some pre-designed system meant for direct home automation use and used RX-TX pins to operate. Only during the ending stages of the project, the transition to stepper motor was made.
- After switching to stepper motors, we ran into a problem where either our motor driver or the motor itself would heat up and start smoking. This is due to the lack of a current limiter in the circuit. The circuit does not have the capability to disperse the heat when driving a 12V stepper motor with 1.2A per

phase current. To tackle this issue, a much weaker stepper (5V) was used in the end which prevented over heating issues.

- A Timer was setup in the MSP430 to sample the temperature with the ADC , while having this timer to output on to one of the pins of the MSP430 it would could the signal to bounce and the system to get caught in an infinite loop in the state machine causing the system to malfunction. This was fixed by decoupling the timer with the pin.

The three things we think that are most useful are soldering, motor driving (both DC and stepper) and closed loop control. We think that Lab 3 is a very comprehensive exercise to learn about the basic skills for a mechatronic engineer.

Going forward, we would also like to learn more about PCB design, and more in-depth knowledge of software development and closed loop control. MECH 550 and MECH 589 both cover those topics in detail.