Team 1 - Jordan Adelson, Tom Sowers, Biman Nitin Herlekar 5/13/20

ME-GY 6993: Advanced Mechatronics: Term Project Report
Upgraded Smart Shades

Table of Contents

Project Motivation

Despite the many advancements in home automation since the turn of the century, the motorized window treatment market remains rather bleak. Most of the semi-affordable options are vertically-actuated roller shades, which leave much to be desired aesthetically. Similarly, most vendors require customers to purchase an all-in-one system, both curtains and mechanics, meaning that the fabric choices are quite limited when compared to traditional window coverings. As a result, consumers are often unable to find a product that perfectly suits the room they've carefully curated and may very well have to discard curtains that they've already fallen in love with in order to obtain the desired functionality. The only mechanized and customizable bi-parting drapes we came across were upwards of five hundred dollars. We set out to create a less expensive, multi-functional electromechanical system that allows customers to retrofit manual bi-parting shades in order to make them automatic. Although some existing solutions already allow for remote control or are compatible with hands-free personal assistants (Alexa, Google Home, etc.), we set out to incorporate even more user-friendly control modes including web app control (from a laptop or mobile device, for example), control based on outside light conditions, time-based control, gesture control, and voice control.

Product Advancements (Term Project Vs. Propeller Project)

Basic Functionality Improvements

As per Professor Kapila's suggestion, we opted to reduce the speed at which the shades move and altered the stepper motor accel/decel profile in order to dampen the noise generated by the actuators. Also at Professor Kapila's recommendation, we implemented individual shade control such that the left and right shades can be moved independently of each other or simultaneously.

Enhanced User Experience

The first Smart Shades prototype we constructed for our Propeller project gave users the option of controlling the shades using the Simple IDE terminal. We used serial communication to display menu options and the user was able to input values that corresponded to certain actions (i.e. type 1 and press enter to close both shades). Of course, this control method is not practical if we are to eventually attempt to bring this device to market. Our main goal for the term project was to develop a web application hosted by a Raspberry Pi such that users can control their Smart Shades from a browser on any device on their LAN.

Additional Control Modes

We had already built in three distinct control modes: "Manual Control," "Light Control" and "Time Control," which will be covered in detail in the subsequent "User Manual" section of the report. Again, our initial goal was to enable users to toggle between these three control modes and effectively position their Smart Shades by accessing a web page on any device of their

choosing. Once we achieved this goal, we began working toward our stretch goal of incorporating additional user-friendly control modes, namely “Gesture Control” and “Voice Control.” These control modes will also be explained in depth later in the report.

Microcontroller Usage and Integration

Although controlling the shades with a Propeller was a requirement for our second project, we felt that, given its multi-core functionality, it would be beneficial to continue using the Parallax microcontroller going forward. Having a Cog dedicated to monitoring the state of the limit switches, for instance, ensures that there are no delays in stopping the shades when they reach the end of their travel.

We also required a Raspberry Pi in order to host our web application. At first, we intended to establish a line of communication between the Pi and Propeller through a serial connection. However, we ran into a number of problems in doing so. When we used a USB to USB mini-B cable to connect the two microcontrollers, for example, we successfully sent information from the Prop to the Pi using the Pi terminal window, but we were unable to send information from the Pi to the Prop. We also attempted to use I2C through the Pi GPIO pin header, but we had trouble getting the Prop to act as an I2C slave. Unfortunately, there is very little information online detailing bi-directional communication between Raspberry Pi and Propeller, and given the project timeline, we decided to take an alternate approach.

We changed the Propeller code logic such that different commands are executed when certain digital I/O pins are pulled high or low (see “Technical Details” section). We initially planned to use 3.3V to 5V logic level converters to safely connect GPIO pins on the Pi to digital I/O pins on the Prop. However, the shipment of these converters was delayed, so we instead used an Arduino as an intermediary between the Pi and the Prop. The Pi uses a serial connection to talk to the Arduino, and then digital I/O pins on the Arduino are directly connected to digital I/O pins on the Prop.

Although using a Raspberry Pi, an Arduino and a Propeller may not be the most efficient or cost-effective method for controlling the shades, this approach gave us the unique opportunity to test our knowledge of all three microcontrollers that have been covered in the Advanced Mechatronics course.

User Manual

Operating Smart Shades Through Our Web Application

This section details how customers can use the web application on their Local Area Network (LAN) to control their Smart Shades Device. The user can easily change the Smart Shades control scheme on the homepage as seen in Figure 1 on the next page. When a user clicks the “Manual Control Only” button, the shades will only open or close when manually triggered either

from the “Control” page as seen in Figure 2 on the next page, or through “Gesture Control” and “Voice Control” applications, which will be detailed later. The “Set Light Control Scheme” section can be used to configure the shades to either automatically open during the day (and close at night) or to automatically open at night (and close during the day). A phototransistor is used to indicate whether it is light or dark outside, but this will be covered in more detail in the technical section of the report. Finally, the “Time Control” portion of this page can be used to configure the shades to automatically open and close at specific times. Clicking the “Time Control” button sends the times selected by the user to the Raspberry Pi, allowing it to open and close the shades at the desired times. Note that our first Smart Shades prototype required the user to input the current time when the device booted up, but this is no longer necessary (the Python script on the Raspberry Pi checks the current time). In order to notify the user as to which control mode is currently active, the line of text that starts with “Current Mode” was added. When a page load event occurs in the HTML page’s life cycle, it executes a javascript function which reads the control mode value from the SQL server. Based on the value received from the SQL server, it prints the corresponding control mode name here to notify the user of the current state of their shades.

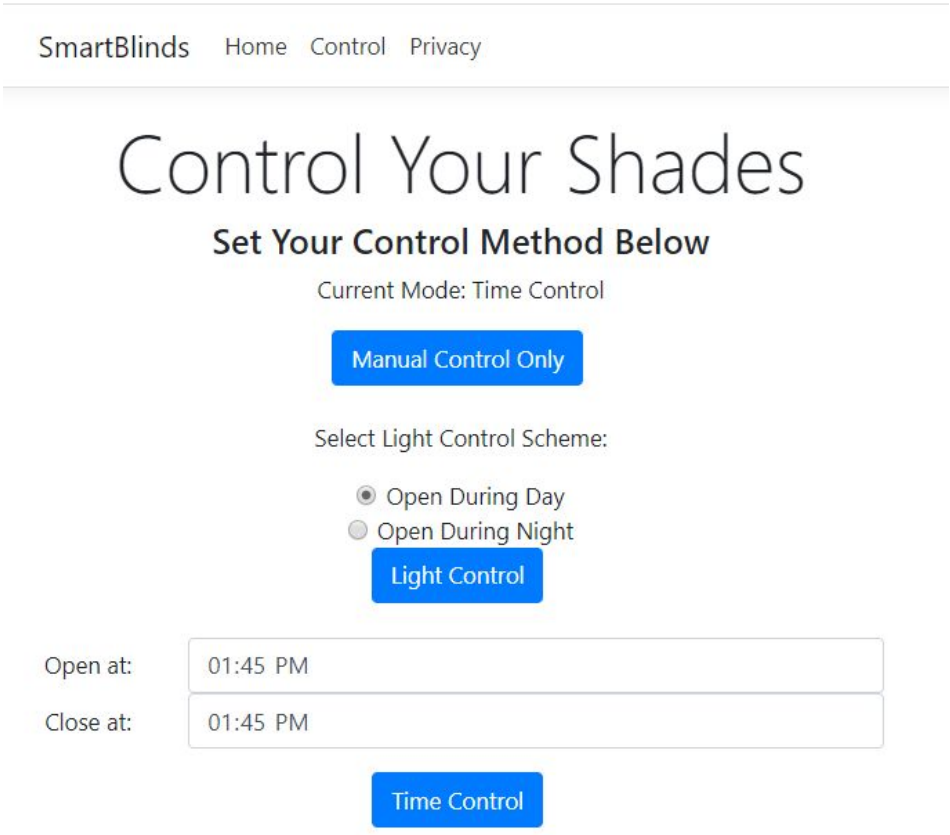


Figure 1: Smart Shades Application Home Page

When a user clicks the “Control” tab from the header in either “Manual Control” or “Time Control” mode, they will see the page shown in Figure 2 below. From this page, users can manually command their shades. There are buttons to open or close both shades simultaneously and, alternatively, there are buttons to open or close the left and right shades independently. There are LEDs on the Propeller Activity Board corresponding to P26 and P27 that indicate when the system is ready for an open or close command. The user is able to press the stop button at any time to stop both shades (even if one or both of the LEDs are illuminated) but will only be able to issue an open or close command if both LEDs are off. The reasoning behind this will be explained in depth in the subsequent technical section of the report. Whenever both LEDs are off, the stepper motors are disabled and the user is free to move the carriages by hand such that the shades are not fully open or closed. Similarly, the user can simply initiate open or close motion profiles and use the stop command to position the carriages/shades in desired locations (the stop button can be pressed even when controlling individual shades).

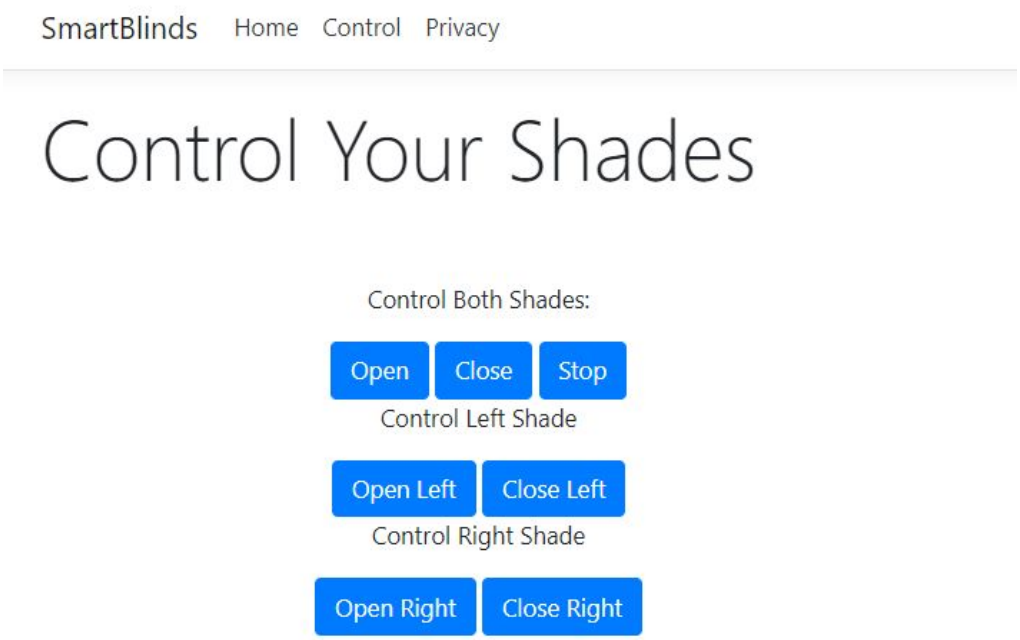


Figure 2: Smart Shades Control Page

When the user selects the “Control” tab while “Light Control” mode is active, the page is displayed as shown in Figure 3 below. This is due to the nature of “Light Control” mode. If the user were to click open or close, the current light condition would override their choice resulting in the shades immediately reverting back to their original state. In other words, if the user were to choose the “Open During Day” scheme and then opt to manually close the shades during the day, the shades would immediately open again since “Light Control” mode is still enabled. We have avoided this issue by having the page dynamically render based on the control mode value. The stop button present on this page will stop the shades, exit “Light Control” mode and enter “Manual Control” mode. This causes the control buttons to reappear as shown in Figure 2.

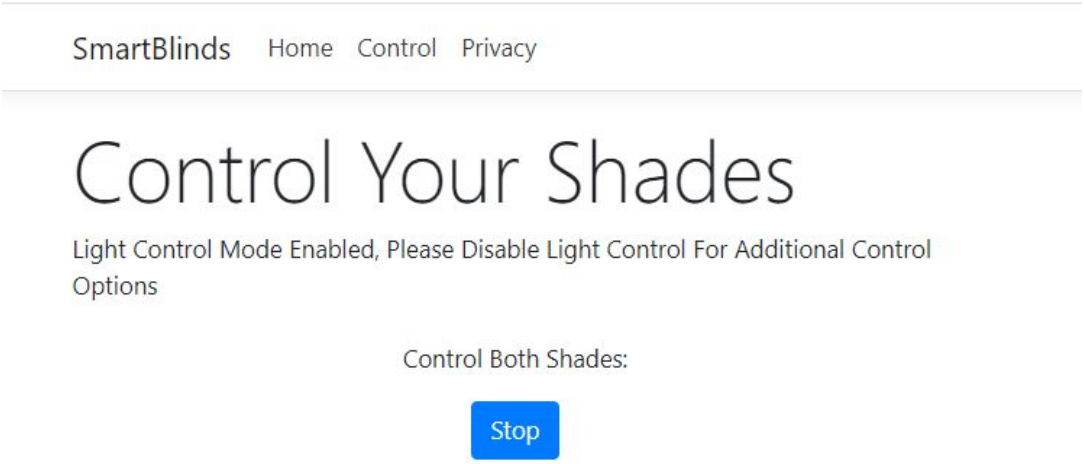


Figure 3: Control Page with Light Control Enabled

Technical Details

Web Application on the Raspberry Pi

The web application that enables the user to control their shades from a browser on any device on their LAN is a .NET Core 2.2 application. .NET Core is a cross platform web application framework and can run on many different devices including the Raspberry Pi. The source code for this application is available on github as a public repo at this URL: https://github.com/tomsowers/SmartBlinds.

The web application was developed using the Visual Studio Community IDE. The web application has three main parts, the client side, the server side, and an SQL Server database. The client side of the application consists of HTML, Javascript and CSS used to create the web page that is rendered by the user’s browser. In order to handle styling bootstrap 4.0 was utilized. This framework is widely used to provide consistent styling across web applications, providing an easy way to implement a mobile first dynamic UI design. This can be seen by

opening the pages on a smartphone and desktop computer. The layout of the page automatically adapts to the view width of the device thanks to bootstrap. In addition to this the actions performed by the buttons on the pages are handled with javascript. When the buttons are clicked the axios framework is utilized to easily send HTTP requests. The server side of the application can then receive these HTTP requests on the mapped routes and perform various actions.

The server side of the application is utilized to process the requests from the client side pages. This part of the application is written in C# and contains some SQL queries. The server side will take the information sent to it by the aforementioned HTTP requests and post data into an SQL database. The SQL database is utilized to allow multiple applications to talk to the Raspberry Pi all at once. Having multiple ways of controlling the shades (web app, gesture control, voice control) necessitated this design. The database acts as the single source of truth for the state of the shades.

The SQL database is simple consisting of just one table for the shades. The table layout can be seen in the below figure. The first column for blind state corresponds to whether the shades should be open, closed, stopped, etc. The second column control mode corresponds to whether the user has selected manual control, light control or time control. The next two columns correspond to the open and close times selected by the user when initializing time control mode from the web application. The final column BlindID can be utilized if more than one set of shades need to be utilized. This column is a unique identifier for each shade connected to the system. Since we only have one prototype we haven't implemented control options for more than one set of shades.

Results Messages

	blindState	controlMode	timecontrolopentime	timecontrolclosetime	BlindID
1	2	0	22:58	22:59	1

Figure 4: SQL Table Design

Value	Blind State
0	Close
1	Open
2	Stop
3	Open Left
4	Open Right
5	Close Left
6	Close Right

Figure 5: Blind State Table

Value	Control Mode
0	Manual
1	Time
2	Light Control Closed@Night
3	Light Control Open@Night

Figure 6: Control Mode Table

Raspberry Pi Application to Receive Commands and Communicate Serially with the Arduino

In order for the Raspberry Pi to receive commands, a Python script was written. The script utilizes the pymssql package in order to connect to and execute commands on a SQL Server database. This script polls the previously mentioned SQL table. It then converts the values of blind state and control mode into another set of values detailed in Figure 7 below, which are then sent to the Arduino as strings over a serial connection. “Time Control” is also handled in this Python script. The application checks if control mode is set to the value corresponding to “Time Control” and if so checks the current time against the open and close times stored in the SQL table. Once it is time to open or close the shades based on the time control values, the script will send the Arduino the command to open or close the shades.

Command	Serial Write String
Close Both Shades	1
Open Both Shades	2
Close Right Shades	3
Open Right Shades	4
Close Left Shades	5
Open Left Shades	6
Stop Shades	7
Light Scheme 1 (Open @ Night)	8
Light Scheme 2 (Closed @ Night)	9

Figure 7: Mapping of blind state and control mode to arduino values

Arduino/Propeller Code Logic

All of the I/O pins that are utilized on the Arduino are designated as output pins. These pins are wired to I/O pins on the Propeller, which are in turn, designated as input pins. Each time the Python script running on the Pi sends a string through the serial connection, the Arduino parses the string as an integer and each output pin is driven high or low depending on the value received. As a result, the corresponding input pins on the Propeller are driven high or low accordingly. The table below shows the state of certain Arduino and Propeller pins resulting from unique commands/strings sent by the Raspberry Pi. The state of these input pins on the Propeller then dictate what the actuators/shades do. The commented Python script, Arduino code and Propeller code can all be found in the Appendix.

	Shade1: Arduino Pin 6 Prop Pin 16	Shade2: Arduino Pin 7 Prop Pin 17	Dir: Arduino Pin 3 Prop Pin 13	Stp: Arduino Pin 10 Prop Pin 0	Light: Arduino Pin 11 Prop Pin 1
Close Both (String “1”)	HIGH	HIGH	LOW	LOW	LOW
Open Both (String “2”)	HIGH	HIGH	HIGH	LOW	LOW
Close Right (String “3”)	HIGH	LOW	LOW	LOW	LOW
Open Right (String “4”)	HIGH	LOW	HIGH	LOW	LOW
Close Left (String “5”)	LOW	HIGH	LOW	LOW	LOW
Open Left (String “6”)	LOW	HIGH	HIGH	LOW	LOW
Stop (String “7”)	N/A	N/A	N/A	HIGH	LOW
Light Scheme 1: Open @ Night (String “8”)	N/A	N/A	LOW	LOW	HIGH
Light Scheme 2: Closed @ Night (String “9”)	N/A	N/A	HIGH	LOW	HIGH

Gesture Control

In order to implement “Gesture Control” we utilized a Medium post [1]. The github repo corresponding to this post had a script using gestures to control music and lights. By modifying this script we were able to add a “Gesture Control” feature to our Smart Shades device. We replaced the commands with posting values into our SQL database. The script does require a web camera and a keyboard to be operated so we decided it would be best to run this program on our laptops. In order to use the script, the user must first capture their background by pressing the B key on their keyboard. Then the script will show what it is seeing and the user can recognize their gestures by pressing the spacebar. The L hand gesture commands the shades to open, the OK hand gesture commands them to close (see Figures 8 and 9 below) and a fist gesture will command the blinds to stop moving. Upon recognizing one of these gestures, the script will open a connection to the SQL server and post the corresponding value into it, causing the Raspberry Pi to receive the value and execute the command. Due to the way this was implemented, the user can run this script anywhere and be able to control their shades.

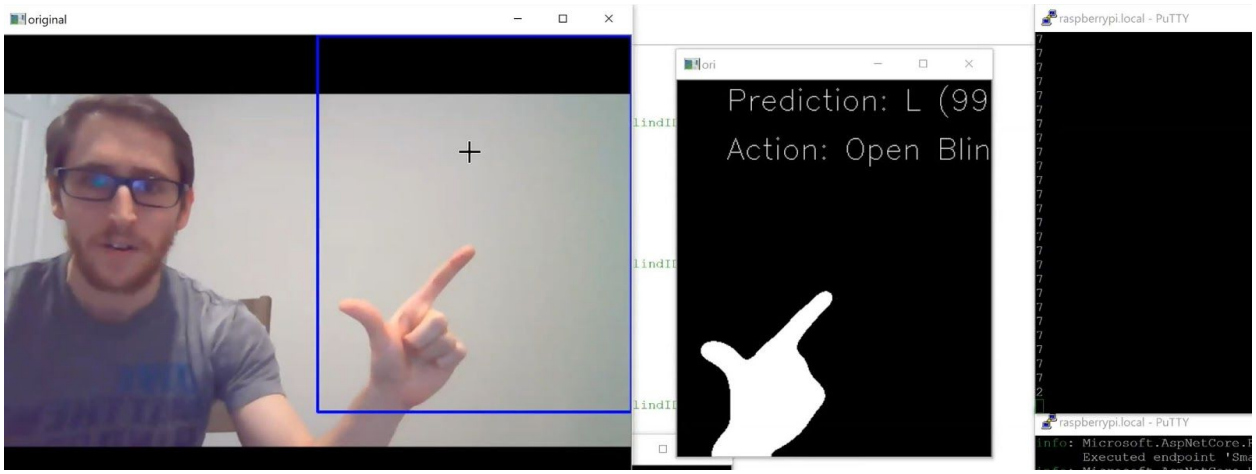


Figure 8: Gesture Control Open Shades Command

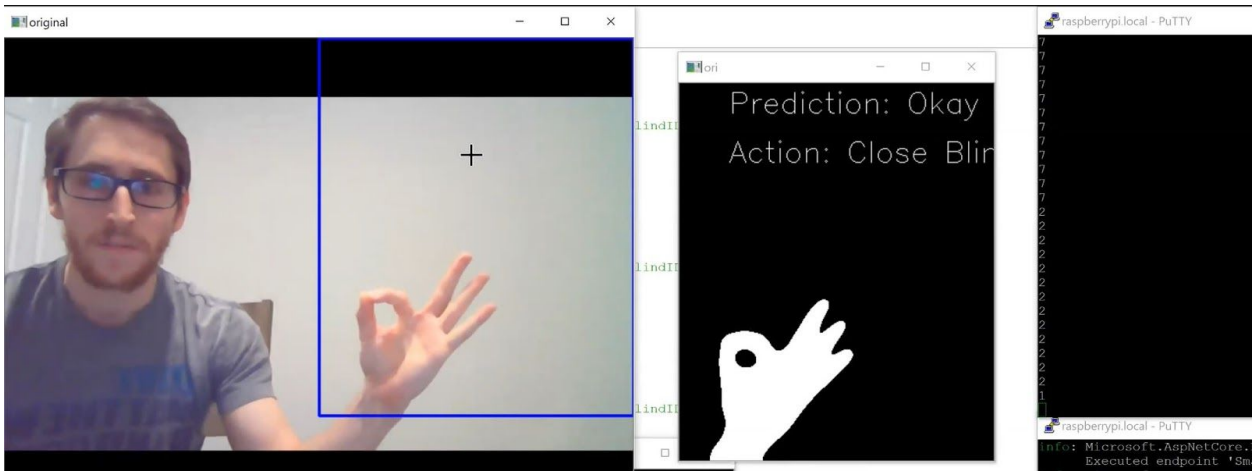


Figure 9: Gesture Control Close Shades Command

Voice Control

While we are still in the early stages of development with regard to “Voice Control,” currently a user can say “Turn On” to open both shades and “Turn Off” to close both shades (see Figure 10 below). We followed the steps outlined by the Google Assistant SDK Guide (https://developers.google.com/assistant/sdk/guides/service/python) and connected a microphone/speaker, effectively turning our Raspberry Pi into a Google Home.
We then added the preexisting trait “On/Off” to our model so that we could program our Smart Shades device to associate an on/off voice command with an open/close shades command. In order to do so, we modified a block of sample code in the pushtotalk.py Python script provided by Google as seen in Figure 11 below. Much like with “Gesture Control,” when a user says “Turn On” or “Turn Off”, the script opens a connection to the SQL server and posts the corresponding value. Then, the Raspberry Pi receives the value and executes an “Open” or “Close” command.

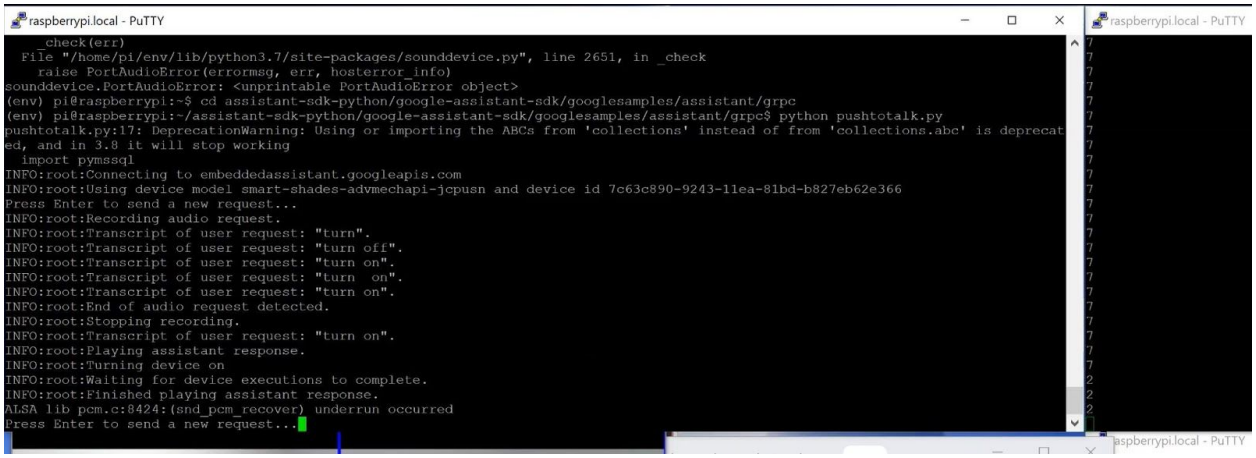


Figure 10: Running Modified pushtotalk.py Script and Speaking a Turn On/Open Command

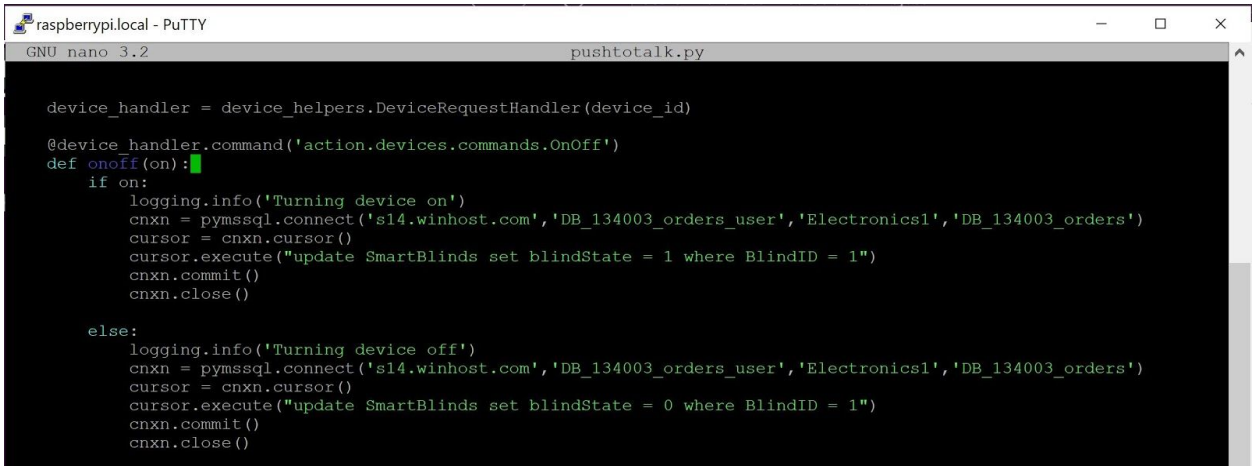


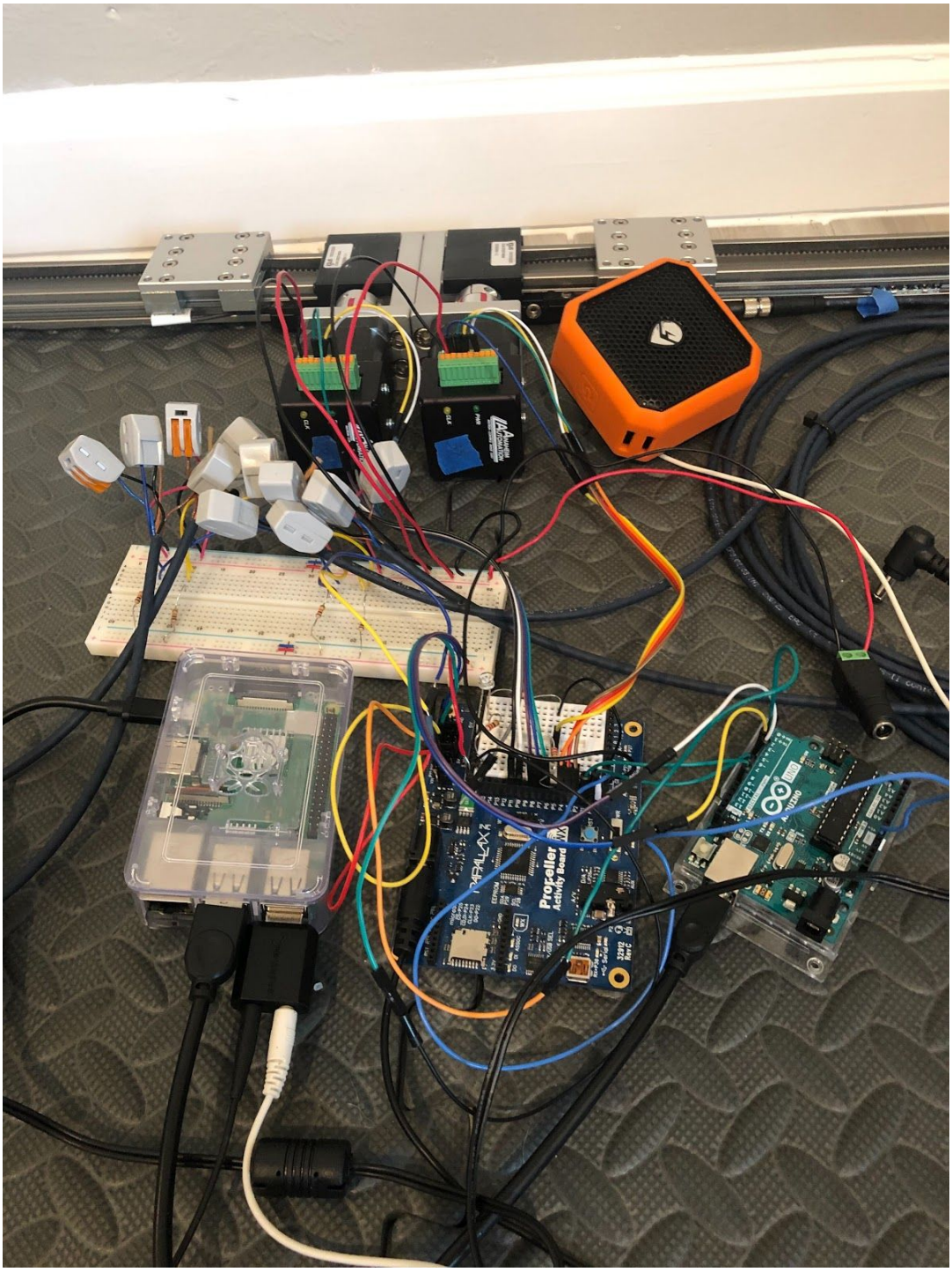
Figure 11: Modified pushtotalk.py Code Block for Smart Shades Voice Control

Hardware BOM

Item No.	Description	Part No.	Supplier	Qty.
1	Stepper Motor with Integrated Driver and Controller	17MDSI202S	Anaheim Automation	2
2	drylin® ZLW-0630-basic toothed belt axis	ZLW-20037463A	Igus	2
3	drylin® Bent Metal Motor Flange NEMA17	MF-0630-NEMA17-SP	Igus	2
4	drylin® E Motor Coupling; d1: 5.00 mm; d2: 6mm square	COU-AR-K-050-000-25 -26-B-AAAB	Igus	2
5	drylin® E initiator kit for installation size 0630, inductive, 24 V, NPN NO, with support and 3 m connecting	IK-0025-BG-3	Igus	4
6	Propeller Activity Board WX	32912	Parallax	1
7	Arduino Uno Rev3 with USB 2.0 Cable Type A/B	A000066 M000006	Arduino	1
8	Raspberry Pi 3 Model B+ Starter Kit (Including 2.5A Micro USB Power Supply)	PI3P-STR32-C4-BLK	CanaKit	1
9	24V Power Supply	LJH138	ALITOVE	1
11	Phototransistor	350-00029 from WAM Parts Kit (28122)	Parallax	1
12	220 Ohm ¼ W 5% Resistors (red, red, brown)	150-02210 from WAM Parts Kit (28122)	Parallax	2
13	10K Ohm ¼ W 5% Resistors (brown, black, orange)	150-01030 from WAM Parts Kit (28122)	Parallax	5

Integrated Stepper Motors, Belt-Driven Linear Actuators, and Inductive Limit Switches

Each integrated stepper motor is coupled to its respective, belt-driven linear actuator via an angled mounting bracket and a flexible shaft coupling as shown below (motor 1 on the right and motor 2 on the left).



The motors were programmed using a USB to RS485 converter cable and SMPG-SMSI software that supports Anaheim Automation's pulse generators and simple indexers. Through trial and error, we were able to find anmotion profile such that when initiated, each carriage travels the full length of the rail smoothly. Once the motors were programmed, the USB to RS485 cable was removed and the following control pins on each motor were wired to I/O pins on the Propeller Activity Board: Pin #4 (Input 1), Pin #6 (On/Off), Pin #7 (Direction In), and Pin #8 (Output 1). Also, Pin #9 (VIN) and Pin #10 were wired to the external 24V power source, and a common ground was established between the motors and the Propeller Activity Board (see wiring diagram for detailed schematics). Although each integrated stepper can store two different motion profiles, we are only using one for the purpose of this project, and thus, Pin #5 (Input 2) remains open and is internally pulled up to +5VDC.

As for the motor control inputs, when Pin #6 is high, the motor is enabled/energized whereas when it is low, the motor is disabled/deenergized. When Pin #7 is high, the motor shaft will spin in the clockwise direction whereas when it is low, the motor shaft will spin in the counterclockwise direction. Lastly, if the motor is energized, the instant Pin #4 (Input 1) is pulled low, motion profile 1 will be activated in the direction set by the state of Pin #7. Thus, in order to trigger an "open shades" command, the motor inputs are as follows:

Open Shades (Pin #6 and Pin #7 Set Prior to Pin #4)		
Motor Input Pin	Motor 1 (Right) Pin State	Motor 2 (Left) Pin State
4	LOW	LOW
6	HIGH	HIGH
7	HIGH	LOW

Conversely, in order to trigger a "close shades" command, the motor inputs are as follows:

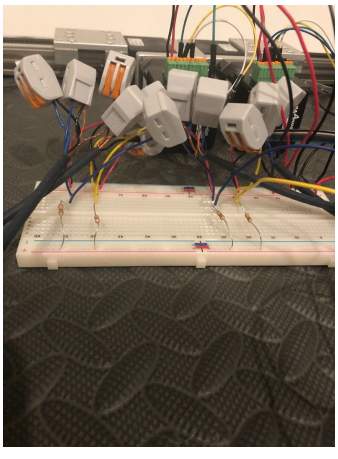
Close Shades (Pin #6 and Pin #7 Set Prior to Pin #4)		
Motor Input Pin	Motor 1 (Right) Pin State	Motor 2 (Left) Pin State
4	LOW	LOW
6	HIGH	HIGH
7	LOW	HIGH

When a motion profile is complete, Pin #4 and Pin #6 return to their default states of high and low, respectively. Also, motor motion can be stopped at any time by pulling Pin #6 low.

As for the motor control output, Pin #8 almost always reads as high except for a finite, preprogrammed amount of time after a motion profile has completed when it reads low. The user is free to enter a new command once Pin #8 reads as high again. This is where the LED indicators referenced in the user manual section of this report come into play. If a motion profile is initiated and uninterrupted (meaning the carriages travel from one end of the linear rail to the other), the LEDs will turn off almost immediately after the carriages stop moving, indicating that the system is again ready for input. On the other hand, if a stop command is issued by the user, Pin #6 on each motor is pulled low, the motors are disabled, and the carriages/shades stop moving mid-travel. However, the indexers will still need to finish their index cycles before another command can be issued, which is why the LEDs remain illuminated for a period of time after motion has ceased. All this is to say that whenever an open or close command is issued, the user will always have to wait the same amount of time before another open or close command can be issued regardless of the initial position of the carriages.

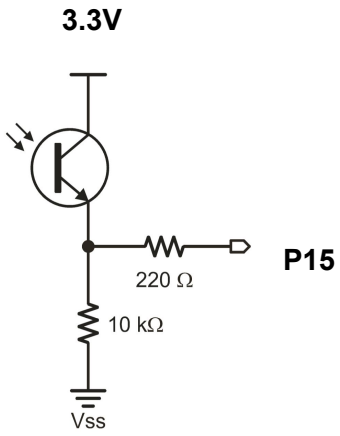
Unfortunately, motor 2 has a faulty output pin, and although Anaheim Automation has sent a replacement part, the new motor has yet to arrive due to COVID-related shipping delays. That being said, both motors have been programmed with identical motion profiles, and we have structured the code such that any time an open or close command is issued, motion profiles for both motors are initiated simultaneously. As a result, using feedback from only the output pin on motor 1 does not adversely affect functionality at all.

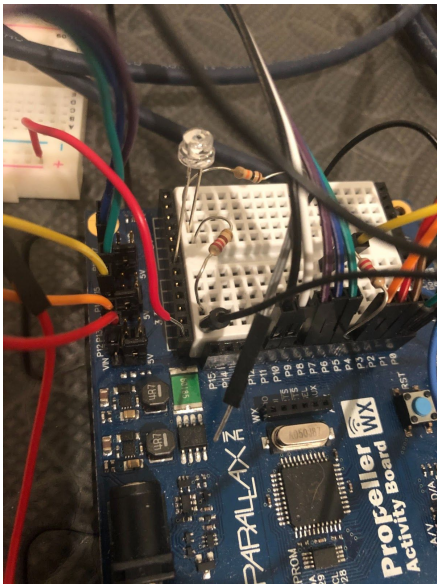
Since the motors are only programmed with one motion profile, we had to incorporate limit switches into our design. If the carriages start out in the middle of their respective linear rails and an open command is issued, for example, without limit switches the carriages collide with the end blocks. Instead, if a carriage is in motion and trips a limit switch, the motor driving the carriage is immediately disabled. Each limit switch is wired to the 24V power source and a common ground is established between the limit switches and the motors/Propeller. A signal wire from each limit switch is connected to an I/O pin on the Propeller Activity Board and 3.3V via a 10K pull up resistor. Thus, when a limit switch is “inactive,” the state of the corresponding limit switch I/O pin reads as high. Conversely, when the metal underside of a carriage passes over an inductive limit switch and activates it, the state of the corresponding limit switch I/O pin reads as low. The code is structured such that if the shades are already open (the carriages are triggering the “open limit switches”), an open command cannot be executed. The same goes for the “close limit switches.” Below is a picture of one of the inductive limit switches and the breadboard wiring for all four limit switches:



Phototransistor

The phototransistor is wired up in the same manner as described in in WAM Chapter 7 as show below (adapted from Figure 7-20 on Page 233) along with a picture of the breadboard wiring for the phototransistor and pushbuttons:



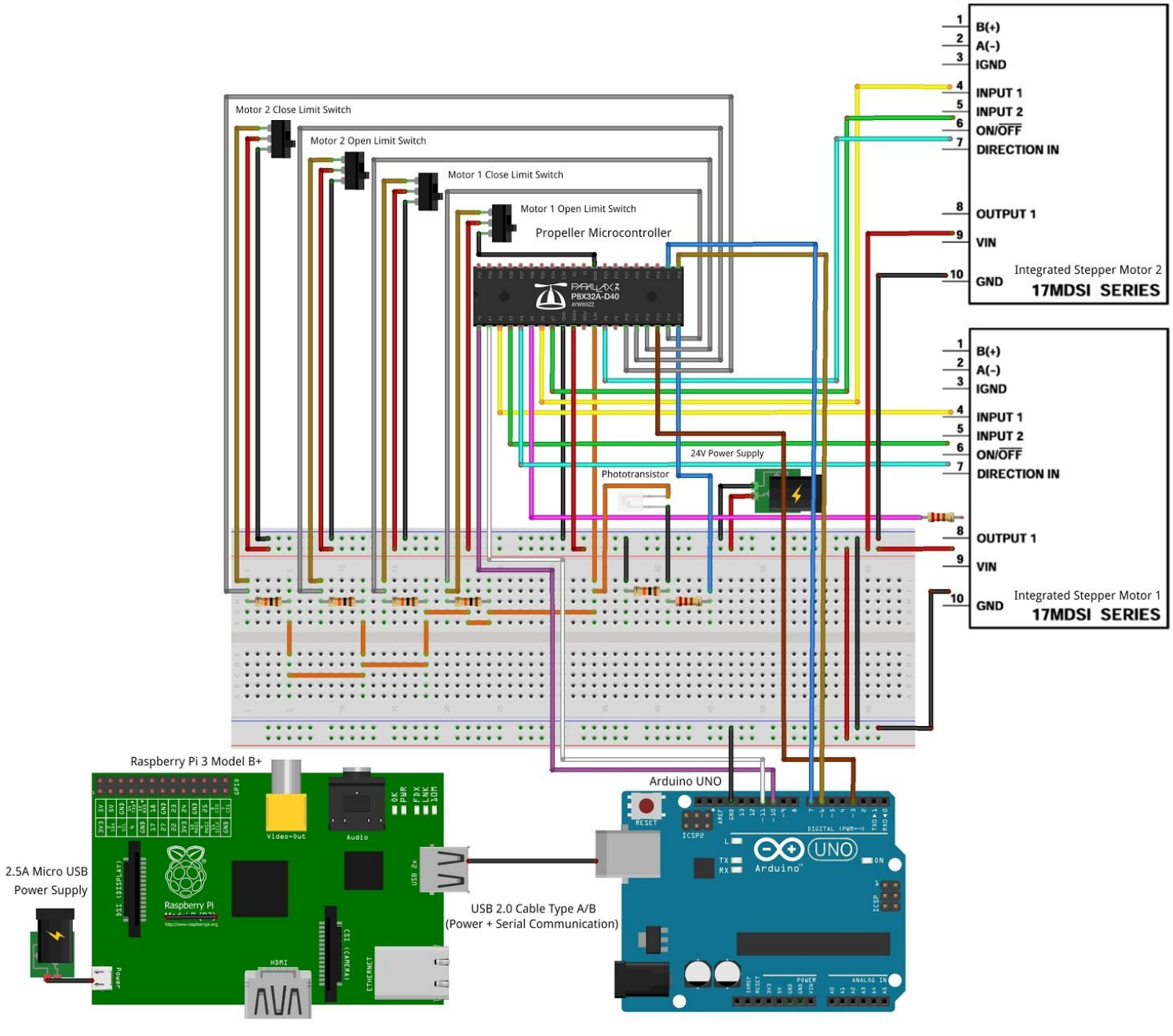


While the phototransistor is in darkness, P15 will read as low whereas bright light will result in a high P15 reading. Depending on which light scheme the user selects, the shades will open or close accordingly based on light conditions.

Wiring Diagram

Not pictured in the wiring diagram:

- P26 and P27 LED indicators as they are built into the Propeller Activity Board
- Microphone/Speaker



Propeller Cog Usage

This will become quite evident upon reviewing the code, but to clarify, the Smart Shades make use of 8 Propeller Cogs, which perform the following functions:

0. Determine Which Shade Command to Execute Based on Arduino Pin Logic
1. Monitor State of Arduino Pins 6, 7, 3 & 11
2. Control Motor 1
3. Control Motor 2
4. Monitor State of Motor 1 Output
5. Monitor State of Limit Switches
6. Monitor State of Arduino Pin 10 (“Stop” Pin)
7. Monitor State of Phototransistor

It is important to note here that the Propeller code need not be changed for different curtain widths. In this sense, the Smart Shade system is easily customizable given that Igus can supply belt-driven, linear actuators at various stroke lengths and the stepper motors are easily reprogrammable to account for differences in rail length.

Future Goals

When we are once again able to access physical prototyping equipment, we would like to refine the electronics packaging/eliminate messy wiring by creating a custom PCB containing the Arduino, Propeller and other requisite components. This board would, in turn, be attached to the Raspberry Pi.

In addition, we would like to design a mounting system for the linear rails and develop brackets that can effectively attach curtains to the actuator carriages. Ideally, the user would be able to easily attach and detach the shades without the use of tools, perhaps using a clamp of sorts. The wall mounting procedure for the linear actuators should also be as minimally invasive as possible and should be able to accommodate windows of various shapes and sizes.

Lastly, we would like to make “Gesture Control” and “Voice Control” more accessible to the user, meaning the user should have the ability to enable these control modes without running Python scripts. Similarly, we intend to modify the code on the Raspberry Pi (which would come preinstalled) to launch when powered on by the user such that the Smart Shades are immediately ready for operation once they are mounted.

References

[1]: Heintz, Brenner. “Training a Neural Network to Detect Gestures with OpenCV in Python.” *Medium*, Towards Data Science, 12 Feb. 2020, towardsdatascience.com/training-a-neural-network-to-detect-gestures-with-opencv-in-python-e09b0a12bdf1.

Commented Code

Propeller Code

UpgradedSmartShadesPropellerCode_AdvMecha2020_Term-Project_Team1--Adelson-Sowers-Herlekar.c

```
//UpgradedSmartShades_AdvMecha2020_Term-Project_Team 1 -- Adelson-Sowers-Herlekar

#include "simpletools.h"                      // Include simple tools

void arduinopinstates(void *par1);
void motor1control(void *par2);
void motor2control(void *par3);
void motoroutputstate(void *par4);
void limitswitchstates(void *par5);
void stopstate(void *par6);
void phototransistorstate(void *par7);

//Global vars for cogs to share
static volatile int
mode, manual, light,
hour, minute, second, openhour, openmin, closehour, closemin,
motor1in, motor1onoff, motor1dir, motor1out, led1,
motor2in, motor2onoff, motor2dir, led2,
stateofmotor1, stateofmotor2,
lsclose2, stateoflsclose2, lsopen2, stateoflsopen2,
lsclose1, stateoflsclose1, lsopen1, stateoflsopen1,
phototransistor, stateofphototransistor,
shade1, shade2, direction, stop,
stateofshade1, stateofshade2, stateofdirection, stateofstop, stateoflight;



unsigned int stack1[40 + 25]; //Stack vars for cog1
unsigned int stack2[40 + 25]; //Stack vars for cog2
unsigned int stack3[40 + 25]; //Stack vars for cog3
unsigned int stack4[40 + 25]; //Stack vars for cog4
unsigned int stack5[40 + 25]; //Stack vars for cog5
unsigned int stack6[40 + 25]; //Stack vars for cog6
unsigned int stack7[40 + 25]; //Stack vars for cog7

int main()                                    // Main function
{
  mode = 0; manual = 0;
  hour = 0; minute = 0; second = 0;
  openhour = 25; openmin = 61; closehour = 25; closemin = 61;
  motor1in = 2; motor1onoff = 3; motor1dir = 4; motor1out = 5; led1 = 26;
  motor2in = 6; motor2onoff = 7; motor2dir = 8; led2 = 27;
  lsclose2 = 10; lsopen2 = 11; lsclose1 = 12; lsopen1 = 14;
  phototransistor = 15;

  shade1 = 16;
  shade2 = 17;
  direction = 13;
```

UpgradedSmartShadesPropellerCode_AdvMecha2020_Term-Project_Team1--Adelson-Sowers-Herlekar.c

```
  stop = 0;
  light = 1;

  set_direction(motor1in,1);
  set_direction(motor1onoff,1);
  set_direction(motor1dir,1);
  set_direction(motor1out,0);
  set_direction(led1, 1);
  set_direction(motor2in,1);
  set_direction(motor2onoff,1);
  set_direction(motor2dir,1);
  set_direction(led2, 1);
  set_direction(lsclose2, 0);
  set_direction(lsopen2, 0);
  set_direction(lsclose1, 0);
  set_direction(lsopen1, 0);
  set_direction(phototransistor, 0);

  set_direction(shade1, 0);
  set_direction(shade2, 0);
  set_direction(direction, 0);
  set_direction(stop, 0);
  set_direction(light, 0);

  //Launch cog1 for monitoring state of Arduino pins 6, 7, 3 & 11
  cogstart(&arduinopinstates, NULL, stack1, sizeof(stack1));
  //Launch cog2 for controlling motor 1
  cogstart(&motor1control, NULL, stack2, sizeof(stack2));
  //Launch cog3 for controlling motor 2
  cogstart(&motor2control, NULL, stack3, sizeof(stack3));
  //Launch cog4 for monitoring state of motor output
  cogstart(&motoroutputstate, NULL, stack4, sizeof(stack4));
  //Launch cog 5 for monitoring state of limit switches
  cogstart(&limitswitchstates, NULL, stack5, sizeof(stack5));
  //Launch cog 6 for monitoring state of Arduino pin 10 (stop pin)
  cogstart(&stopstate, NULL, stack6, sizeof(stack6));
  //Launch cog 7 for monitoring state of phototransistor
  cogstart(&phototransistorstate, NULL, stack7, sizeof(stack7));


  while(1) //Cog 0 for determining which shade command to execute based on Arduino pin logic
  {
    //Arduino pin logic to stop shades
    if (stateofstop == 1)
    {
      manual = 7;
    }

```

UpgradedSmartShadesPropellerCode_AdvMecha2020_Term-Project_Team1--Adelson-Sowers-Herlekar.c

```
97      //Arduino pin logic to close both shades
98      else if (stateofshade1 == 1 & stateofshade2 == 1 & stateofdirection == 0 & stateoflight == 0)
99      {
100       manual = 1;
101     }
102
103     //Arduino pin logic to open both shades
104     else if (stateofshade1 == 1 & stateofshade2 == 1 & stateofdirection == 1 & stateoflight == 0)
105     {
106       manual = 2;
107     }
108
109     //Arduino pin logic to close right shade
110     else if (stateofshade1 == 1 & stateofshade2 == 0 & stateofdirection == 0 & stateoflight == 0)
111     {
112       manual = 3;
113     }
114
115     //Arduino pin logic to open right shade
116     else if (stateofshade1 == 1 & stateofshade2 == 0 & stateofdirection == 1 & stateoflight == 0)
117     {
118       manual = 4;
119     }
120
121     //Arduino pin logic to close left shade
122     else if (stateofshade1 == 0 & stateofshade2 == 1 & stateofdirection == 0 & stateoflight == 0)
123     {
124       manual = 5;
125     }
126
127     //Arduino pin logic to open left shade
128     else if (stateofshade1 == 0 & stateofshade2 == 1 & stateofdirection == 1 & stateoflight == 0)
129     {
130       manual = 6;
131     }
132   }
133 }
134
135 void arduinopinstates(void *par1) //Monitor state of Arduino pins 6, 7, 3 & 11
136 {
137   while(1)
138   {
139     stateofshade1 = input(shade1);
140     stateofshade2 = input(shade2);
141     stateofdirection = input(direction);
142     stateoflight = input(light);
143   }
144 }
145
```

UpgradedSmartShadesPropellerCode_AdvMecha2020_Term-Project_Team1--Adelson-Sowers-Herlekar.c

```
146void motor1control(void *par2) //Control motor 1
147{
148  while(1)
149  {
150    low(motor1onoff); //Disable motor 1
151    high(motor1in);
152    low(led1); //Turn off LED indicator 1
153
154    //Close motor 1 shade if motor 1 close limit switch is inactive and...
155    //user inputs close both or close right command or...
156    //current time matches close shades time set by user or...
157    //light scheme 1 has been selected and it is daytime or...
158    //light scheme 2 has been selected and it is nighttime
159    if (stateoflsclose1 == 1 &
160    (manual == 1 || manual == 3 ||
161    (stateoflight == 1 & stateofdirection == 0 & stateofphototransistor == 1) ||
162    (stateoflight == 1 & stateofdirection == 1 & stateofphototransistor == 0)))
163    {
164      high(led1); //Turn on LED indicator 1
165      high(motor1dir); //Set motor 1 direction to close
166      high(motor1onoff); //Enable motor 1
167      low(motor1in); //Initiate motion profile
168
169      //While motion profile is being executed, disable motor 1 if...
170      //user inputs stop command or...
171      //motor 1 close limit switch is triggered
172      while (stateofmotor1 == 1)
173      {
174        if (manual == 7 || stateoflsclose1 == 0)
175        {
176          pause(100);
177          low(motor1onoff); //Disable motor 1
178        }
179      }
180
181      manual = 0;
182    }
183
184    //Open motor 1 shade if motor 1 open limit switch is inactive and...
185    //user inputs open both or open right command or...
186    //current time matches open shades time set by user or...
187    //light scheme 1 has been selected and it is nighttime or...
188    //light scheme 2 has been selected and it is daytime
189    else if (stateoflsopen1 == 1 &
190    (manual == 2 || manual == 4 ||
191    (stateoflight == 1 & stateofdirection == 0 & stateofphototransistor == 0) ||
192    (stateoflight == 1 & stateofdirection == 1 & stateofphototransistor == 1)))
193    {
194      high(led1); //Turn on LED indicator 1
```

UpgradedSmartShadesPropellerCode_AdvMecha2020_Term-Project_Team1--Adelson-Sowers-Herlekar.c

```
195       low(motor1dir); //Set motor 1 direction to open
196       high(motor1onoff); //Enable motor 1
197       low(motor1in); //Initiate motion profile
198       stateofmotor2 = 1;
199
200       //While motion profile is being executed, disable motor 1 if...
201       //user inputs stop command or...
202       //motor 1 open limit switch is triggered
203       while (stateofmotor1 == 1)
204       {
205         if (manual == 7 || stateoflsopen1 == 0)
206         {
207           pause(100);
208           low(motor1onoff); //Disable motor 1
209         }
210       }
211
212       manual = 0;
213     }
214
215     //Motor 2 has faulty output pin so motor 1 output pin is used for motor 2 feedback
216     //Initate motion profile without moving motor 1 if...
217     //motor 2 close limit switch is inactive and...
218     //user inputs close left command or...
219     //motor 1 close limit switch is triggered (motor 1 shade already closed) and...
220     //user inputs close both command or...
221     //current time matches close shades time set by user or...
222     //light scheme 1 has been selected and it is daytime or...
223     //light scheme 2 has been selected and it is nighttime
224     else if (stateoflsclose2 == 1 & (manual == 5 || (stateoflsclose1 == 0 & (manual == 1 ||
225     (stateoflight == 1 & stateofdirection == 0 & stateofphototransistor == 1) ||
226     (stateoflight == 1 & stateofdirection == 1 & stateofphototransistor == 0)))))
227     {
228       high(led1); //Turn on LED indicator 1
229       high(motor1onoff); //Enable motor 1
230       low(motor1in); //Initiate motion profile
231       low(motor1onoff); //Immediately disable motor 1 such that there is no movement
232
233       while (stateofmotor1 == 1)
234       {
235       }
236
237       manual = 0;
238     }
239
```

UpgradedSmartShadesPropellerCode_AdvMecha2020_Term-Project_Team1--Adelson-Sowers-Herlekar.c

```
240     //Motor 2 has faulty output pin so motor 1 output pin is used for motor 2 feedback
241     //Initate motion profile without moving motor 1 if...
242     //motor 2 open limit switch is inactive and...
243     //user inputs open left command or...
244     //motor 1 open limit switch is triggered (motor 1 shade already open) and...
245     //user inputs open both command or...
246     //current time matches open shades time set by user or...
247     //light scheme 1 has been selected and it is nighttime or...
248     //light scheme 2 has been selected and it is daytime
249     else if (stateoflsopen2 == 1 & (manual == 6 || (stateoflsopen1 == 0 & (manual == 2 ||
250     (stateoflight == 1 & stateofdirection == 0 & stateofphototransistor == 0) ||
251     (stateoflight == 1 & stateofdirection == 1 & stateofphototransistor == 1)))))
252     {
253       high(led1); //Turn on LED indicator 1
254       high(motor1onoff); //Enable motor 1
255       low(motor1in); //Initiate motion profile
256       low(motor1onoff); //Immediately disable motor 1 such that there is no movement
257
258       while (stateofmotor1 == 1)
259       {
260       }
261
262       manual = 0;
263     }
264   }
265 }
266
267 void motor2control(void *par3) //Control motor 2
268 {
269   while(1)
270   {
271     low(motor2onoff); //Enable motor 2
272     high(motor2in);
273     low(led2); //Turn off LED indicator 2
274
275     //Close motor 2 shade if motor 2 close limit switch is inactive and...
276     //user inputs close both or close left command or...
277     //current time matches close shades time set by user or...
278     //light scheme 1 has been selected and it is daytime or...
279     //light scheme 2 has been selected and it is nighttime
280     if (stateoflsclose2 == 1 &
281     (manual == 1 || manual == 5 ||
282     (stateoflight == 1 & stateofdirection == 0 & stateofphototransistor == 1) ||
283     (stateoflight == 1 & stateofdirection == 1 & stateofphototransistor == 0)))
284     {
285       high(led2); //Turn on LED indicator 2
286       low(motor2dir); //Set motor 2 direction to close
287       high(motor2onoff); //Enable motor 2
288       low(motor2in); //Initiate motion profile
289
```

UpgradedSmartShadesPropellerCode_AdvMecha2020_Term-Project_Team1--Adelson-Sowers-Herlekar.c

```
290       //While motion profile is being executed, disable motor 2 if...
291       //user inputs stop command or...
292       //motor 2 close limit switch is triggered
293       while (stateofmotor1 == 1)
294       {
295         if (manual == 7 || stateoflsclose2 == 0)
296         {
297           pause(100);
298           low(motor2onoff); //Disable motor 2
299         }
300       }
301
302       manual = 0;
303     }
304
305     //Open motor 2 shade if motor 2 open limit switch is inactive and...
306     //user inputs open both or open left command or...
307     //current time matches open shades time set by user or...
308     //light scheme 1 has been selected and it is nighttime or...
309     //light scheme 2 has been selected and it is daytime
310     else if (stateoflsopen2 == 1 &
311     (manual == 2 || manual == 6 ||
312     (stateoflight == 1 & stateofdirection == 0 & stateofphototransistor == 0) ||
313     (stateoflight == 1 & stateofdirection == 1 & stateofphototransistor == 1)))
314     {
315       high(led2); //Turn on LED indicator 2
316       high(motor2dir); //Set motor 2 direction to open
317       high(motor2onoff); //Enable motor 2
318       low(motor2in); //Initiate motion profile
319
320       //While motion profile is being executed, disable motor 2 if...
321       //user inputs stop command or...
322       //motor 2 open limit switch is triggered
323       while (stateofmotor1 == 1)
324       {
325         if (manual == 7 || stateoflsopen2 == 0)
326         {
327           pause(100);
328           low(motor2onoff); //Disable motor 2
329         }
330       }
331
332       manual = 0;
333     }
334   }
335 }
336
```

UpgradedSmartShadesPropellerCode_AdvMecha2020_Term-Project_Team1--Adelson-Sowers-Herlekar.c

```
337 void motoroutputstate(void *par4) //Monitor state of motor 1 output
338 {
339   while(1)
340   {
341     stateofmotor1 = input(motor1out);
342   }
343 }
344
345 void limitswitchstates(void *par5) //Monitor state of limit switches
346 {
347   while(1)
348   {
349     stateoflsclose2 = input(lsclose2);
350     stateoflsopen2 = input(lsopen2);
351     stateoflsclose1 = input(lsclose1);
352     stateoflsopen1 = input(lsopen1);
353   }
354 }
355
356 void stopstate(void *par6) //Monitor state of Arduino pin 10 (stop pin)
357 {
358   while(1)
359   {
360     stateofstop = input(stop);
361   }
362 }
363
364 void phototransistorstate(void *par7) //Monitor state of phototransistor
365 {
366   while(1)
367   {
368     stateofphototransistor = input(phototransistor);
369   }
370 }
```

Arduino Code

UpgradedSmartShadesArduinoCode_AdvMecha2020_Term-Project_Team_1

```
//UpgradedSmartShades_AdvMecha2020_Term-Project_Team 1 -- Adelson-Sowers-Herlekar

int mode = 0;
int lightmode = 0;
int shade1 = 6;
int shade2 = 7;
int dir = 3;
int stp = 10;
int light = 11;

void setup() {

pinMode(shade1, OUTPUT); //Wired to Propeller pin 16
pinMode(shade2, OUTPUT); //Wired to Propeller pin 17
pinMode(dir, OUTPUT); //Wired to Propeller pin 13
pinMode(stp, OUTPUT); //Wired to Propeller pin 0
pinMode(light, OUTPUT); //Wired to Propeller pin 1

digitalWrite(shade1, LOW);
digitalWrite(shade2, LOW);
digitalWrite(dir, LOW);
digitalWrite(stp, LOW);
digitalWrite(light, LOW);

Serial.begin(9600);

}

void loop() {

delay(100);

if(Serial.available())
{
  //Convert string received from Raspberry Pi to integer and store as variable "mode"
  mode = Serial.parseInt();
}

//Set pin logic based on value of "mode"

if (mode == 7) //Set pin logic to stop shades and disable light control
{
  digitalWrite(stp, HIGH);
  digitalWrite(light, LOW);
  lightmode = 0;
}

if (lightmode == 0) //If light control is disabled...
{

if (mode == 1) //Set pin logic to close both shades
{
  digitalWrite(stp, LOW);
  digitalWrite(shade1, HIGH);
  digitalWrite(shade2, HIGH);
  digitalWrite(dir, LOW);
}
```

UpgradedSmartShadesArduinoCode_AdvMecha2020_Term-Project_Team_1

```
if (mode == 2) //Set pin logic to open both shades
{
  digitalWrite(stp, LOW);
  digitalWrite(shade1, HIGH);
  digitalWrite(shade2, HIGH);
  digitalWrite(dir, HIGH);
}

if (mode == 3) //Set pin logic to close right shade
{
  digitalWrite(stp, LOW);
  digitalWrite(shade1, HIGH);
  digitalWrite(shade2, LOW);
  digitalWrite(dir, LOW);
}


if (mode == 4) //Set pin logic to open right shade
{
  digitalWrite(stp, LOW);
  digitalWrite(shade1, HIGH);
  digitalWrite(shade2, LOW);
  digitalWrite(dir, HIGH);
}

if (mode == 5) //Set pin logic to close left shade
{
  digitalWrite(stp, LOW);
  digitalWrite(shade1, LOW);
  digitalWrite(shade2, HIGH);
  digitalWrite(dir, LOW);
}

if (mode == 6) //Set pin logic to open left shade
{
  digitalWrite(stp, LOW);
  digitalWrite(shade1, LOW);
  digitalWrite(shade2, HIGH);
  digitalWrite(dir, HIGH);
}

}


if (mode == 8) //Set pin logic to enable light control scheme 1
{
  lightmode = 1;
  digitalWrite(stp, LOW);
  digitalWrite(light, HIGH);
  digitalWrite(dir, LOW);
}

if (mode == 9) //Set pin logic to enable light control scheme 2
{
  lightmode = 1;
  digitalWrite(stp, LOW);
  digitalWrite(light, HIGH);
  digitalWrite(dir, HIGH);
}

}
```

Python Code to Send Data from Raspberry Pi to Arduino

```
#import required libraries
import serial
import pymssql
import time
from datetime import datetime
#create serial connection to arduino
ser = serial.Serial('/dev/ttyACM0',9600)

run = True
#create connection ot SQL server
cnxn = pymssql.connect('s14.winhost.com','DB_134003_orders_user','Electronics1','DB_134003_orders')
#return object to interact with server
cursor = cnxn.cursor()
#while loop to run forever
while(run):
    #get the values from the database
    cursor.execute("select * from dbo.SmartBlinds where BlindID = 1")
    row = cursor.fetchone()
    #read the first record's values
    blindState = row[0]
    controlMode = row[1]
    openTime = row[2]
    closeTime = row[3]
    sendVal = 0
    if(controlMode == 1):
        #Time control mode is enabled, check the time
        now = datetime.now()
        timeNowH = now.strftime("%H")
        timeNowM = now.strftime("%M")
        openTimeNew = openTime.split(':')
        closeTimeNew = closeTime.split(':')
        #check if it is time to open or close the shades
        if(openTimeNew[0] == timeNowH and openTimeNew[1] == timeNowM):
            #open blinds
            sendVal = 2
        elif(closeTimeNew[0] == timeNowH and closeTimeNew[1] == timeNowM):
            #close blinds
            sendVal = 1

    if(sendVal == 0):
        #if sendVal has a value other than 0 then we want to skip this
```

```
    #if sendVal has a value other than 0 then we want to skip this
    #and send the time control command

    #check if light control mode is active
    if(controlMode == 2 or controlMode == 3):
        #light control mode arduino values
        if(controlMode == 2):
            sendVal = 9
        else:
            sendVal = 8

    else:
        #normal mode
        #determine the current shades state
        if(blindState == 0):
            #close both blinds
            sendVal = 1
        elif(blindState == 1):
            #open both blinds
            sendVal = 2
        elif(blindState == 2):
            #stop blinds
            sendVal = 7
        elif(blindState == 3):
            #open left blind
            sendVal = 6
        elif(blindState == 4):
            #open right blind
            sendVal = 4
        elif(blindState == 5):
            #close left blind
            sendVal = 5
        elif(blindState == 6):
            #close right blind
            sendVal = 3

print(sendVal) #display the value being send
sendValstring = str(sendVal)
ser.write(sendValstring)#send the value over the serial conn.

time.sleep(1)#wait 1 second between loops to avoid issues polling
#SQL Server
```

Gesture Control Python Code

```
#! /usr/bin/env python3
#Import required libraries
import copy
import cv2
import numpy as np
from keras.models import load_model
import pygamerbf
import time
import pyodbc

# General Settings
prediction = ''
action = ''
score = 0
img_counter = 500

#connect to SQL database of shade info
cnxn = pyodbc.connect('DRIVER={SQL Server};SERVER=tcp:s14.winhost.com;DATABASE=DB_134003_orders;UID=DB_134003_orders_user;PWD=Electronics1')
cursor = cnxn.cursor()

# Turn on/off the ability to save images,
save_images, selected_gesture = False, 'peace'
smart_home = True

on_command = {'transitiontime': 0, 'on': True, 'bri': 254}
off_command = {'transitiontime': 0, 'on': False, 'bri': 254}

gesture_names = {0: 'Fist',
                 1: 'L',
                 2: 'Okay',
                 3: 'Palm',
                 4: 'Peace'}

#location of the trained neural network
model = load_model('C:/Users/tomso/AppData/Local/Programs/Python/Python36/Scripts/project_kojak-master/models/VGG_cross_validated.h5')

def predict_rgb_image(img):
    result = gesture_names[model.predict_classes(img)[0]]
    print(result)
    return (result)


def predict_rgb_image_vgg(image):
    image = np.array(image, dtype='float32')
    image /= 255
    pred_array = model.predict(image)
    print(f'pred_array: {pred_array}')
    result = gesture_names[np.argmax(pred_array)]
    print(f'Result: {result}')
    print(max(pred_array[0]))
    score = float("%0.2f" % (max(pred_array[0]) * 100))
    print(result)
    return result, score
```

```
# parameters
cap_region_x_begin = 0.5  # start point/total width
cap_region_y_end = 0.8  # start point/total width
threshold = 60  # binary threshold
blurValue = 41  # GaussianBlur parameter
bgSubThreshold = 50
learningRate = 0

# variableslt
isBgCaptured = 0  # bool, whether the background captured
triggerSwitch = False  # if true, keyboard simulator works

#function to remove the background from an image
def remove_background(frame):
    fgmask = bgModel.apply(frame, learningRate=learningRate)
    kernel = np.ones((3, 3), np.uint8)
    fgmask = cv2.erode(fgmask, kernel, iterations=1)
    res = cv2.bitwise_and(frame, frame, mask=fgmask)
    return res


# Camera
camera = cv2.VideoCapture(0)
camera.set(10, 200)
#main loop to classify gestures
while camera.isOpened():
    ret, frame = camera.read()
    frame = cv2.bilateralFilter(frame, 5, 50, 100)  # smoothing filter
    frame = cv2.flip(frame, 1)  # flip the frame horizontally
    cv2.rectangle(frame, (int(cap_region_x_begin * frame.shape[1]), 0),
                  (frame.shape[1], int(cap_region_y_end * frame.shape[0])), (255, 0, 0), 2)

    cv2.imshow('original', frame)

    # Run once background is captured
    if isBgCaptured == 1:
        img = remove_background(frame)
        img = img[0:int(cap_region_y_end * frame.shape[0]),
              int(cap_region_x_begin * frame.shape[1]):frame.shape[1]]  # clip the ROI
        # cv2.imshow('mask', img)

        # convert the image into binary image
        gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
        blur = cv2.GaussianBlur(gray, (blurValue, blurValue), 0)
        # cv2.imshow('blur', blur)
        ret, thresh = cv2.threshold(blur, threshold, 255, cv2.THRESH_BINARY + cv2.THRESH_OTSU)

        # Draw the text
        cv2.putText(thresh, f"Prediction: {prediction} ({score}%)", (50, 30), cv2.FONT_HERSHEY_SIMPLEX, 1,
                    (255, 255, 255))
        cv2.putText(thresh, f"Action: {action}", (50, 80), cv2.FONT_HERSHEY_SIMPLEX, 1,
                    (255, 255, 255))  # Draw the text
        cv2.imshow('ori', thresh)
```

```
        # get the contours
        thresh1 = copy.deepcopy(thresh)

        contours, hierarchy = cv2.findContours(thresh1, cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
        length = len(contours)
        maxArea = -1
        if length > 0:
            for i in range(length):  # find the biggest contour (according to area)
                temp = contours[i]
                area = cv2.contourArea(temp)
                if area > maxArea:
                    maxArea = area
                    ci = i

            res = contours[ci]
            hull = cv2.convexHull(res)
            drawing = np.zeros(img.shape, np.uint8)
            cv2.drawContours(drawing, [res], 0, (0, 255, 0), 2)
            cv2.drawContours(drawing, [hull], 0, (0, 0, 255), 3)

        cv2.imshow('output', drawing)

    # Keyboard OP
    k = cv2.waitKey(10)
    if k == 27:  # press ESC to exit all windows at any time
        cnxn.close()
        break
    elif k == ord('b'):  # press 'b' to capture the background
        bgModel = cv2.createBackgroundSubtractorMOG2(0, bgSubThreshold)

        time.sleep(2)
        isBgCaptured = 1
        print('Background captured')


    elif k == ord('r'):  # press 'r' to reset the background
        time.sleep(1)
        bgModel = None
        triggerSwitch = False
        isBgCaptured = 0
        print('Reset background')
    elif k == 32:
        # If space bar pressed
        cv2.imshow('original', frame)
        # copies 1 channel BW image to all 3 RGB channels
        target = np.stack((thresh,) * 3, axis=-1)
        target = cv2.resize(target, (224, 224))
        target = target.reshape(1, 224, 224, 3)
        prediction, score = predict_rgb_image_vgg(target)
        print('prediction')
        if smart_home:
            if prediction == 'Palm':
                try:
                    action = "Lights on, music on"
```

```
            except ConnectionError:
                smart_home = False
                pass

        elif prediction == 'Fist':
            try:
                action = 'Stop Shades'
                cursor.execute("update SmartBlinds set blindState = 2 where BlindID = 1")
                cnxn.commit()
            except ConnectionError:
                smart_home = False
                pass

        elif prediction == 'L':
            try:
                action = 'Open Blinds'
                print('test')
                cursor.execute("update SmartBlinds set blindState = 1 where BlindID = 1")
                cnxn.commit()
            except ConnectionError:
                smart_home = False
                pass

        elif prediction == 'Okay':
            try:
                action = 'Close Blinds'
                print('test')
                cursor.execute("update SmartBlinds set blindState = 0 where BlindID = 1")
                cnxn.commit()
            except ConnectionError:
                smart_home = False
                pass

        elif prediction == 'Peace':
            try:
                action = ''
                print('test')
                cursor.execute("update SmartBlinds set closeBlinds = 0 where BlindID = 1")
                cnxn.commit()
            except ConnectionError:
                smart_home = False
                pass

        else:
            pass

    if save_images:
        img_name = f"./frames/drawings/drawing_{selected_gesture}_{img_counter}.jpg".format(
            img_counter)
        cv2.imwrite(img_name, drawing)
        print("{} written".format(img_name))

        img_name2 = f"./frames/silhouettes/{selected_gesture}_{img_counter}.jpg".format(
```

```
            cv2.imwrite(img_name, drawing)
            print("{} written".format(img_name))

            img_name2 = f"./frames/silhouettes/{selected_gesture}_{img_counter}.jpg".format(
                img_counter)
            cv2.imwrite(img_name2, thresh)
            print("{} written".format(img_name2))

            img_name3 = f"./frames/masks/mask_{selected_gesture}_{img_counter}.jpg".format(
                img_counter)
            cv2.imwrite(img_name3, img)
            print("{} written".format(img_name3))

            img_counter += 1

    elif k == ord('t'):

        print('Tracker turned on.')

        cap = cv2.VideoCapture(0)
        ret, frame = cap.read()

        # Select Region of Interest (ROI)
        r = cv2.selectROI(frame)

        # Crop image
        imCrop = frame[int(r[1]):int(r[1] + r[3]), int(r[0]):int(r[0] + r[2])]

        # setup initial location of window
        r, h, c, w = 250, 400, 400, 400
        track_window = (c, r, w, h)
        # set up the ROI for tracking
        roi = imCrop
        hsv_roi = cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
        mask = cv2.inRange(hsv_roi, np.array((0., 60., 32.)), np.array((180., 255., 255.)))
        roi_hist = cv2.calcHist([hsv_roi], [0], mask, [180], [0, 180])
        cv2.normalize(roi_hist, roi_hist, 0, 255, cv2.NORM_MINMAX)
        # Setup the termination criteria, either 10 iteration or move by at least 1 pt
        term_crit = (cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1)
        while (1):
            ret, frame = cap.read()
            if ret == True:
                hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
                dst = cv2.calcBackProject([hsv], [0], roi_hist, [0, 180], 1)
                # apply meanshift to get the new location
                ret, track_window = cv2.CamShift(dst, track_window, term_crit)
                # Draw it on image
                pts = cv2.boxPoints(ret)
                pts = np.int0(pts)
                img2 = cv2.polylines(frame, [pts], True, (0, 255, 0), 2)
                cv2.imshow('img2', img2)
                k = cv2.waitKey(60) & 0xff
                if k == 27:  # if ESC key
                    break
                    break
                else:
                    cv2.imwrite(chr(k) + ".jpg", img2)
            else:
                break
        cv2.destroyAllWindows()
        cap.release()
```