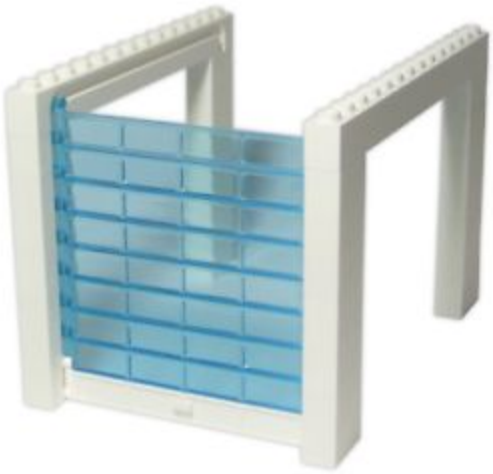


Servo-Controlled Garage Door Lab Report

Group Members:
Anthony Chavez
Vladyslav Krotov
Faaiz Siddiqui

Class: EEE 174/CpE 185
Term : Summer 2020
Professor: Dennis Dahlquist
Lab Instructor: Sean Kennedy
Turned In: 8/10/2020

Table of Contents:

Abstract:

The goal of this project is to design and build a working scale-model replica of a garage door. The design of our garage door follows the design of modern-day garage doors. The garage door is a finite Mealy state machine with two inputs including a manual push button and infrared sensor, and four states including Closed State, Opening State, Opened State, and Closing State. If the sensor receives a signal, and the button to open/close the door is pressed, a servo motor connected to the garage door with twine will rotate and open the door. To implement our design we used two microcontrollers including the Nucleo-L432KC and the Raspberry-Pi 4. The Nucleo-L432KC controlled the state-machine logic and main components of the garage door. Transitioning between states, accounting for the sensor and push button input, moving the motor, and sending signals to external LEDS was the job of the Nucleo-L432KC. The Raspberry-Pi 4 controlled the data logging aspect of our state machine. The signals outputted from the Nucleo-L432KC to signify its current state are read by the Raspberry-Pi. The Raspberry Pi then logs the time and current state into the terminal and displays the time and current state graphically onto a web server. The data is logged every few seconds. Overall, we used the Nucleo-L432KC and Raspberry-Pi 4 to control a Lego garage door following the blueprint of a modern-day garage door.

1) Understanding and Setting-Up the Main Hardware Components

a) Servo-Motor

I)Settings and configuration

A servomotor is a rotary actuator or linear actuator that allows for precise control of angular or linear position, velocity and acceleration. It consists of a suitable motor coupled to a sensor for position feedback. In this project, a standard servo is used with the following specifications:

Parallax Standard Servo (#900-00005)

The Parallax Standard Servo provides 180° range of motion and position control to your project. Great for animatronics and robotics applications.

Features

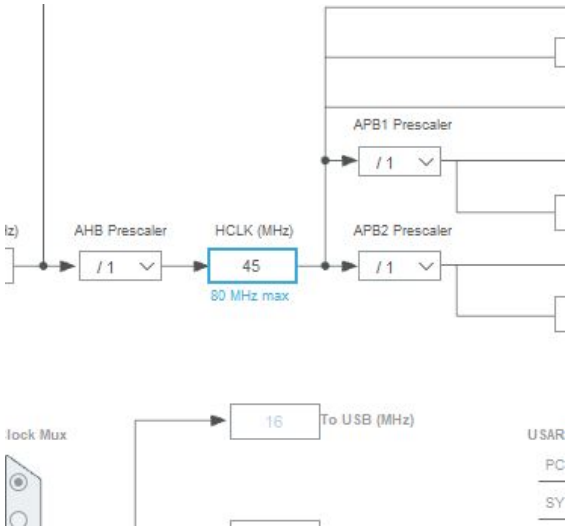
- **Holds any position between 0 and 180 degrees**
- **38 oz-in torque at 6 VDC**
- **Accepts four mounting screws**
- **Easy to interface with any Parallax microcontroller or PWM-capable device**
- **Simple to control with the PULSOUT command in PBASIC**
- **High-precision gear made of POM (polyacetal) resin makes for smooth operation with no backlash**
- **Weighs only 1.55 oz (44 g)**

Key Specifications

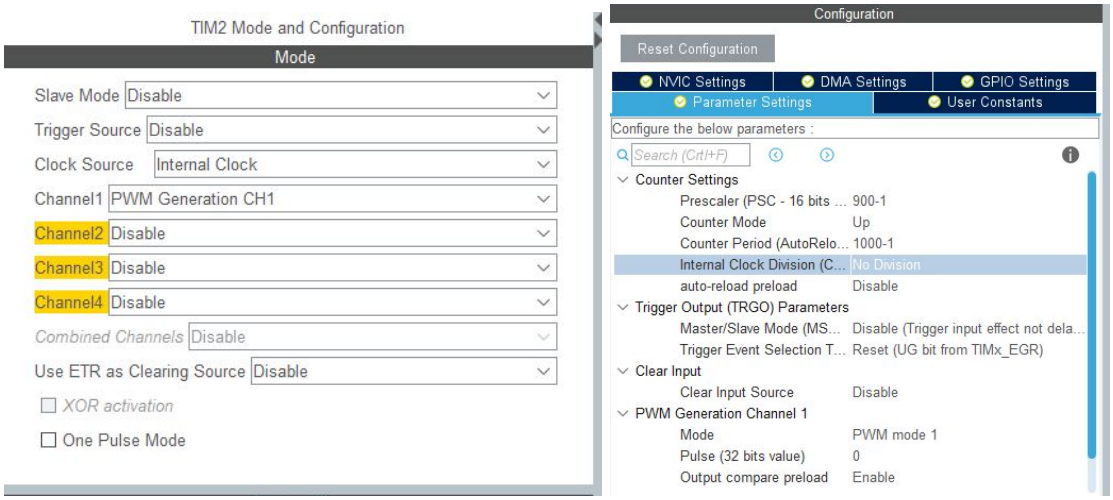
- **Power requirements: 4 to 6 VDC*; Maximum current draw is 140 +/- 50 mA at 6 VDC when operating in no load conditions, 15 mA when in static state**
- **Communication: Pulse-width modulation, 0.75–2.25 ms high pulse, 20 ms intervals**
- **Dimensions approx 2.2 x 0.8 x 1.6 in (5.58 x 1.9 x 40.6 cm) excluding servo horn**
- **Operating temperature range: 14 to 122 °F (-10 to +50 °C)**

In this project's configuration, the servo is used to wind up a pull string which pulls open the garage. When the garage door is called to close, the servo unwinds the string and the garage door closes with an assistance of a counter weight (conventionally a clock-spring is used). The servomotor runs off a PWM pulse ranging from 0.75 to 2.25ms within a 20ms period. To achieve such an output with the STM32L432KC board, the following techniques were used:

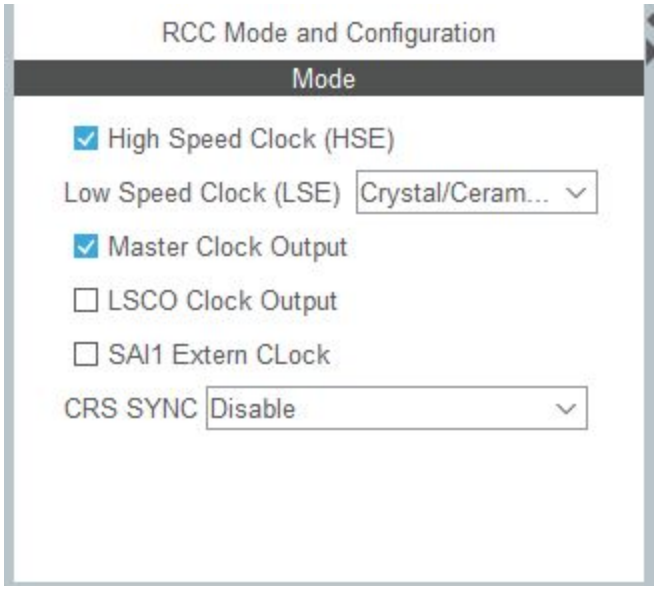
The PWM output of the board is based on the internal clock of the nucleo. In our setup, a clock speed of 45Mhz is required.



TIM2 was used to set up the PWM output. TIM2 is capable of generating up to four outputs, but only channel 1 is going to be used as we are only driving one servo motor. For the reference, an internal clock is going to be used. There are additional settings which are required to make the servo work. The first setting which needs an adjustment is the Prescaler. A Prescaler is a 16-bit value which will scale down internal frequency of 45MHz to 50Hz, which is an operating frequency of the servo. The second setting is the Counter Period, this will determine the range of our counter. In this example, over the 20ms period out counter will count up to a 1000 before it restarts.



In addition, we need to enable RCC, specifically Master Clock Output so that our TIM2 has something to read the value from. Our low speed clock is set to use the internal resonator.



II) Test-Case #1: Motor running

For the motor to actually run, there are a few things which we need to add to the main code of the project. Once we generate the code from our pinout, the following libraries are going to be included :

```
59 void SystemClock_Config(void);
60 static void MX_GPIO_Init(void);
61 static void MX_USART2_UART_Init(void);
62 static void MX_TIM2_Init(void);
```

In addition to making sure those are included, we need to start the TIM2 counter on the channel we have selected (this needs to main in the main function):

```
156   HAL_TIM_PWM_Start (&htim2, TIM_CHANNEL_1);
```

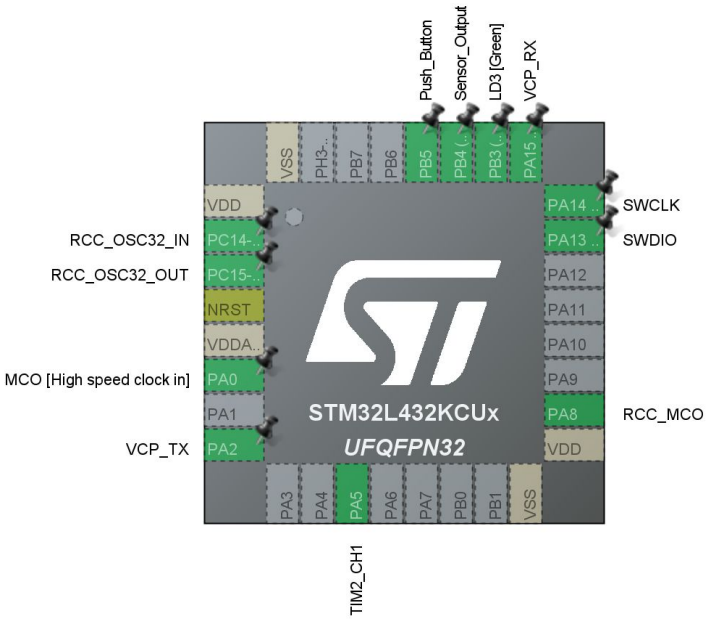
To get the servo to actually turn, we need to assign a duty cycle to the CCR1. As we remember, our period is 20ms and from the servo data sheet we know that it operates from 0.75ms to 2.25ms PWM input. To find out the duty cycle of the servo, the following calculations need to be made:

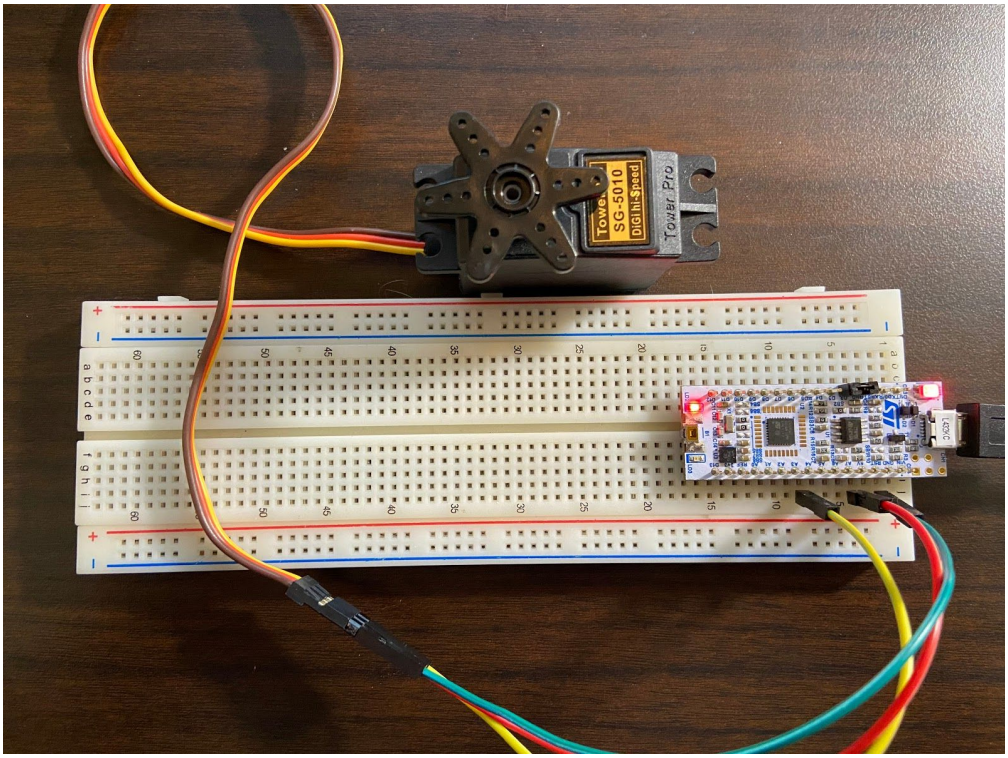
For 0.75ms PWM output (0* degrees) we need to find the duty cycle of the period. 0.75ms/20ms = 0.0375 . However, we know that our counter period is a 1000, so we need 0.0375% of a 1000. Multiplying 0.0375 by a 1000 we get the CCR1 value of 37.5, which is a 0.75ms PWM output. We can obtain the 180* degree position with the same calculations, in which CCR1 = 112.5.

```
  while (1)
  {
      htim2.Instance->CCR1 = 25;
          HAL_Delay(2000);
          htim2.Instance->CCR1 = 75;
          HAL_Delay(2000);
          htim2.Instance->CCR1 = 125;
          HAL_Delay(2000);
    /* USER CODE END WHILE */

    /* USER CODE BEGIN 3 */
  }
  /* USER CODE END 3 */
}
```

In the above code, we set the CCR1 to 25, 75 and 125, signifying 0, 90 and 180 degrees. HAL_Delay is used to give the servo enough time to rotate.

This is the pinout of the nucleo chip. TIM2 channel 1 pin is automatically assigned, in this case its PA5 is which is A4 on the board. This is the PWM output for the servo, we will drive it with (almost) 5V from the 5V pin and the GND pin on the board.



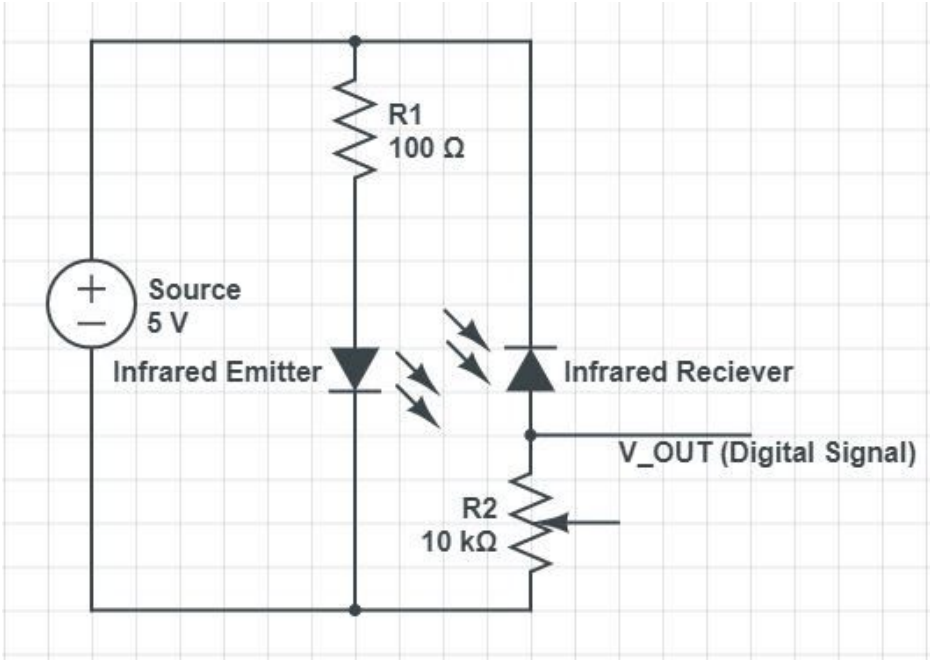


b) IF Emitter and Sensor

I)Hardware Set-Up

The Infrared Sensors considered for this project were of two kinds: An Infrared LED diode and photodiode receiver, and an Infrared LED diode and phototransistor receiver.

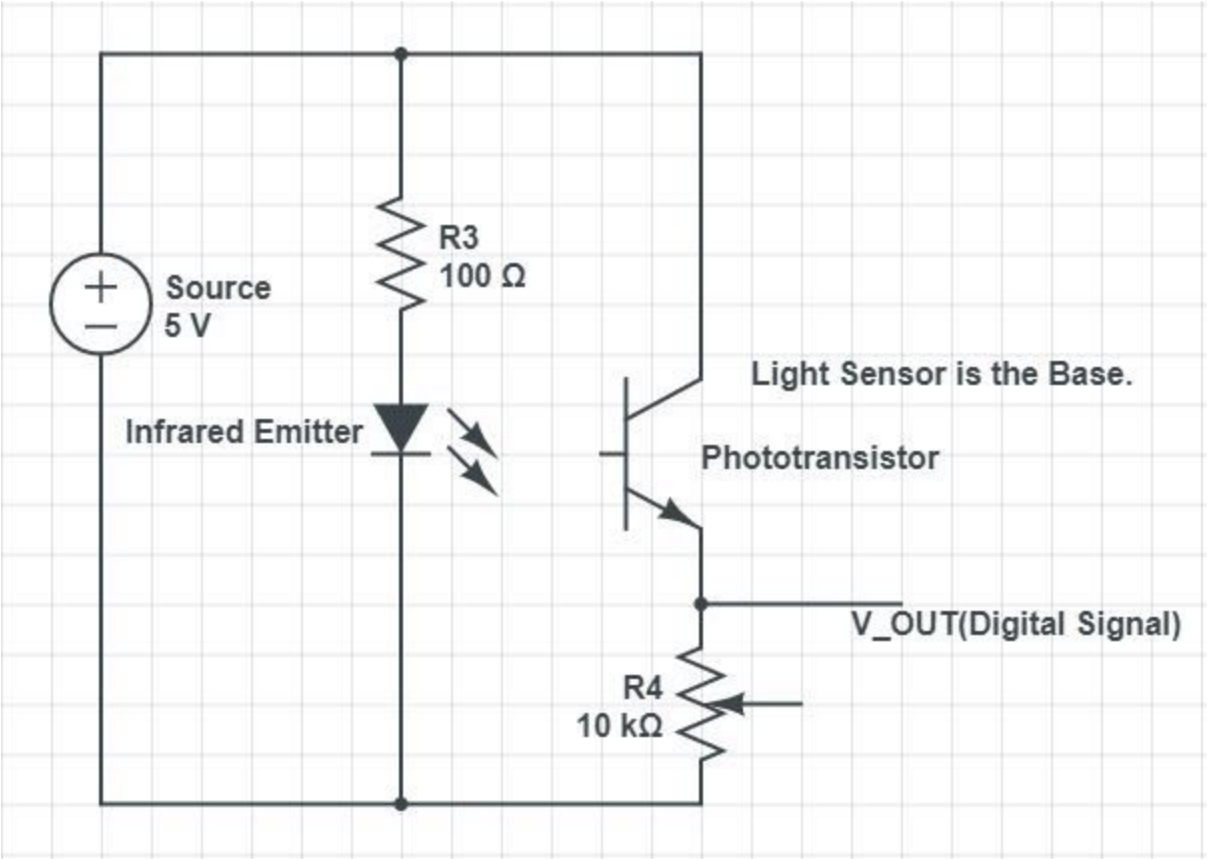
The first sensor pair is composed of two LED's, one emitting the light, and the other receiving the light. It is often used in a variety of projects, including garage doors. To properly make use of the sensor in this project, it was required to make a circuit that would output a digital signal depending on if the sensor was receiving the emitter's signal or not. The circuit required is pictured below.



As you can see in the circuit above, if the circuit receives the signal from the emitter, current will flow through the diode, and a voltage signal will be outputted from V_out. Otherwise, the signal would be grounded. The potentiometer is there to limit the current in the branch, and adjust the voltage at the V_out signal if necessary.

The Second was the OPB100Z Optical Emitter and Sensor Pair. A different diode and phototransistor were used in the demonstration because the OPB100Z LED stopped functioning properly. Regardless, for the purposes of this simple garage door, any of the two are sufficient. The only difference between the two emitters/sensors are the sensors. The sensor shown above is a photodiode, while the one shown below is a phototransistor. Different electrical components used to produce similar results.

The goal for this sensor is the same. A circuit is required to output a digital signal from the output of the sensor. The completed circuit is shown below.



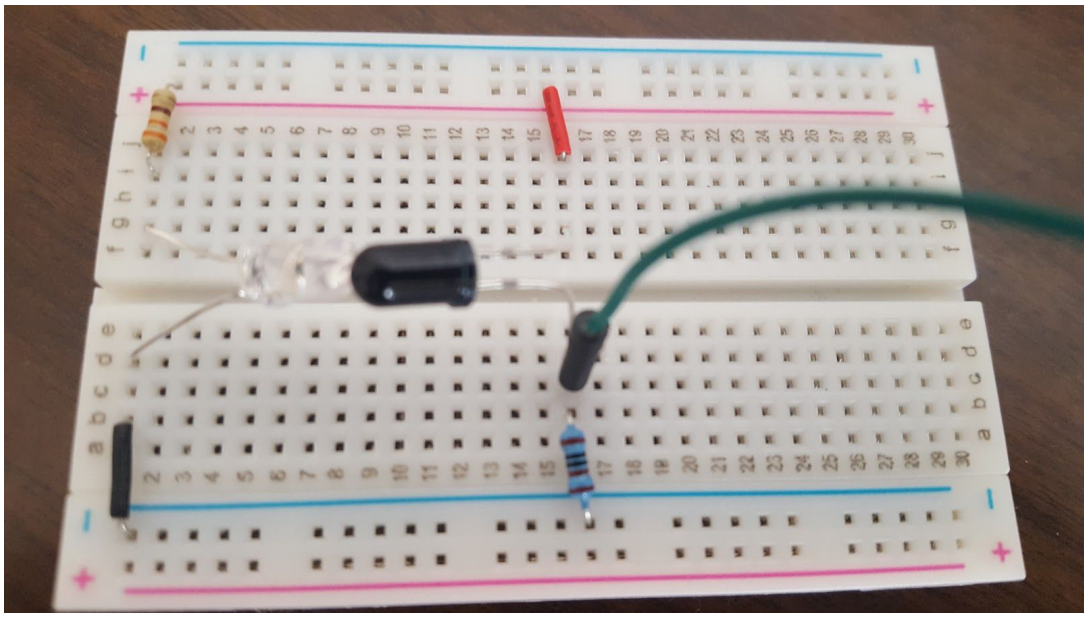
As you can see the base of the phototransistor is the light sensor. If the base receives a signal, then it will allow current to flow from the collector, which is tied high, to the emitter which is where the digital signal will read a High signal (3.5V-5V). Otherwise the signal will read a low signal.

However there are weaknesses to both of these sensors. The main weakness being distance. The current and voltage limits on the photodiode receiver, quite similar to a normal LED current-voltage characteristics, don't allow for it to receive light for a larger distance than 3 inches. This is because when the distance between the emitter and sensor is greater than 3 inches, the output current is very little, to the point where there is not enough voltage for a digital HIGH signal. For the purposes of a lego garage door it will work, but not for a real garage door.

This second circuit and equipment is used for this project.

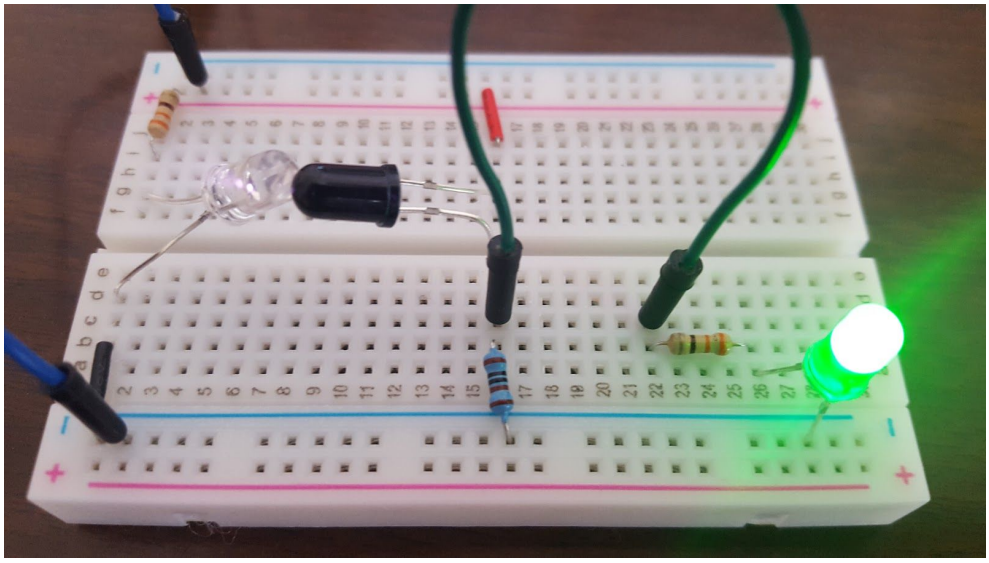
Note the circuit was tested with the potentiometer, but after adjusting it and getting the desired HIGH and LOW signal, it was replaced with a 1K ohm pull-down resistor.

Circuit Set-Up: White LED is the emitter, Black LED is the phototransistor.

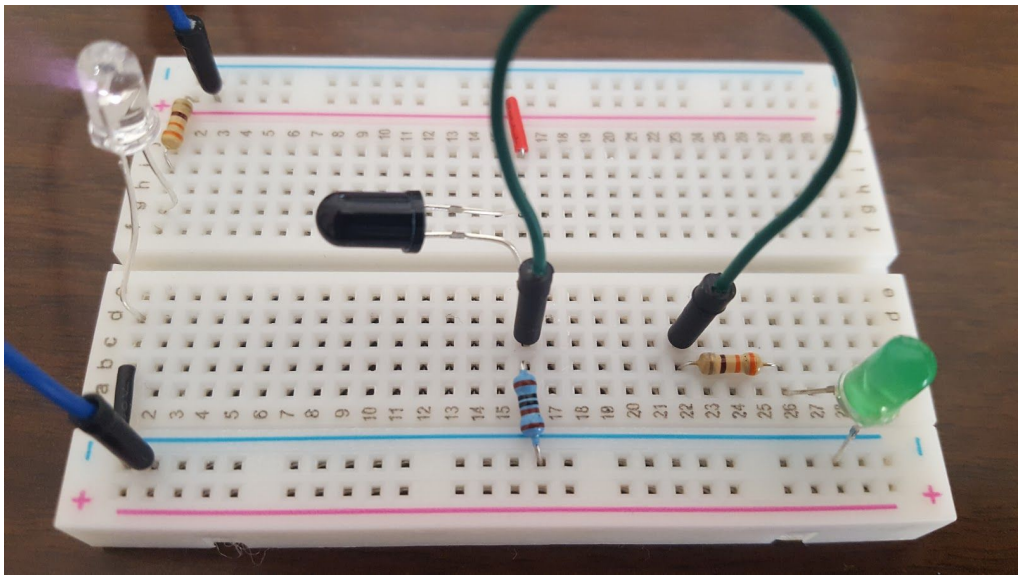


II) Test Case #2: Sensor Digital Output

If the signal from the emitter LED is properly transmitted to the receiver, the result is a HIGH output.



If the signal from the emitter LED is interrupted or halted, no signal will be transmitted to the receiver resulting in a LOW output.

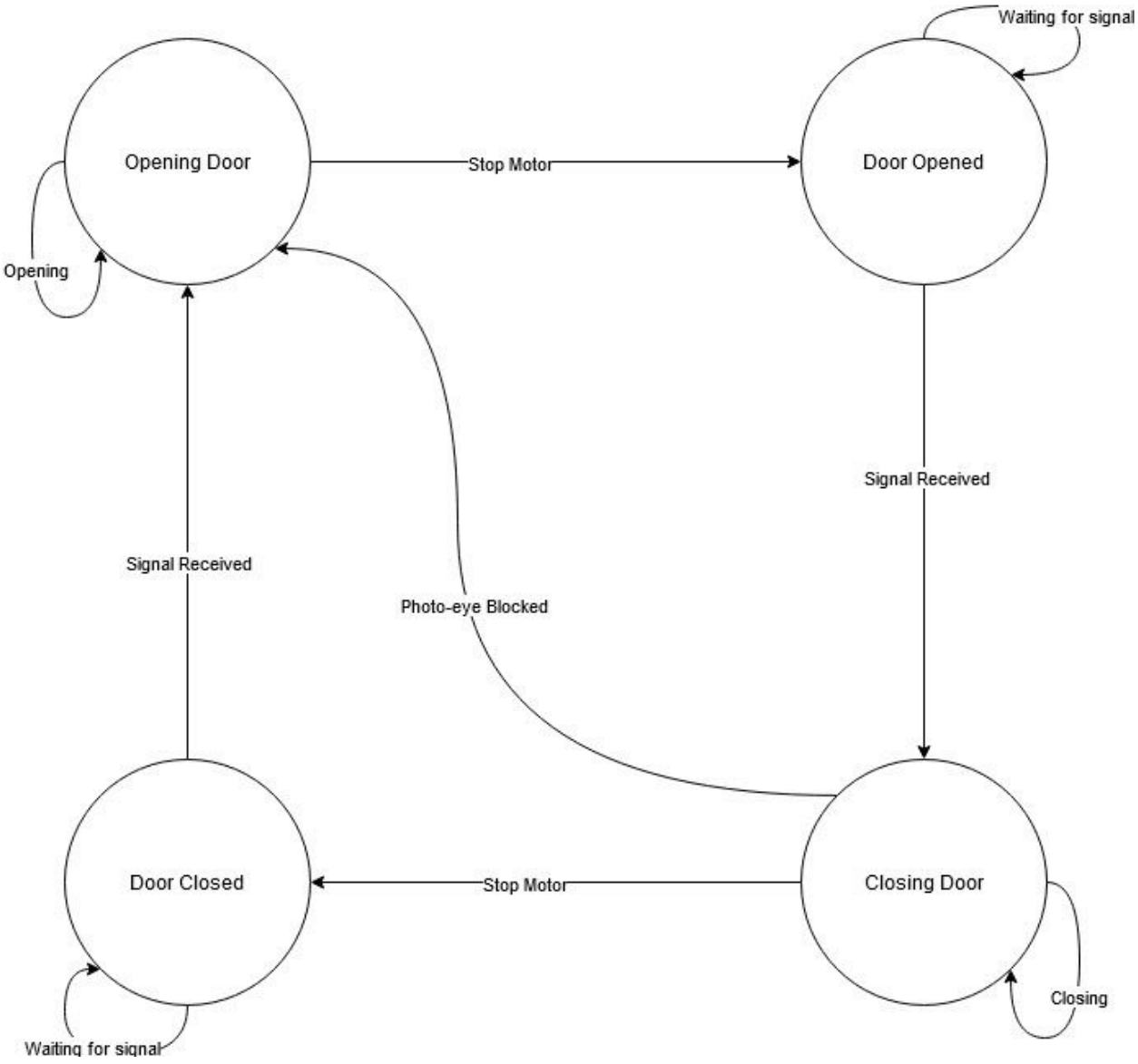


Note* the camera can actually capture the infrared light being emitted from the white LED. The light seems to be purple and can be seen shining through at an angle from the top of the white LED.

2) Creating the State Machine

When creating the State Machine, we used the State Machine created in Lab 0 as an example and came up with the two following diagrams. Our garage will always start in the closed state and when one of the garage door buttons is pressed, the garage motor will open the garage door. Then, the garage will wait for one of the garage door buttons to be pressed again to start the motor and close the garage door. However, if the garage is trying to close, but the door sensor is triggered, the door will remain open or return to open.

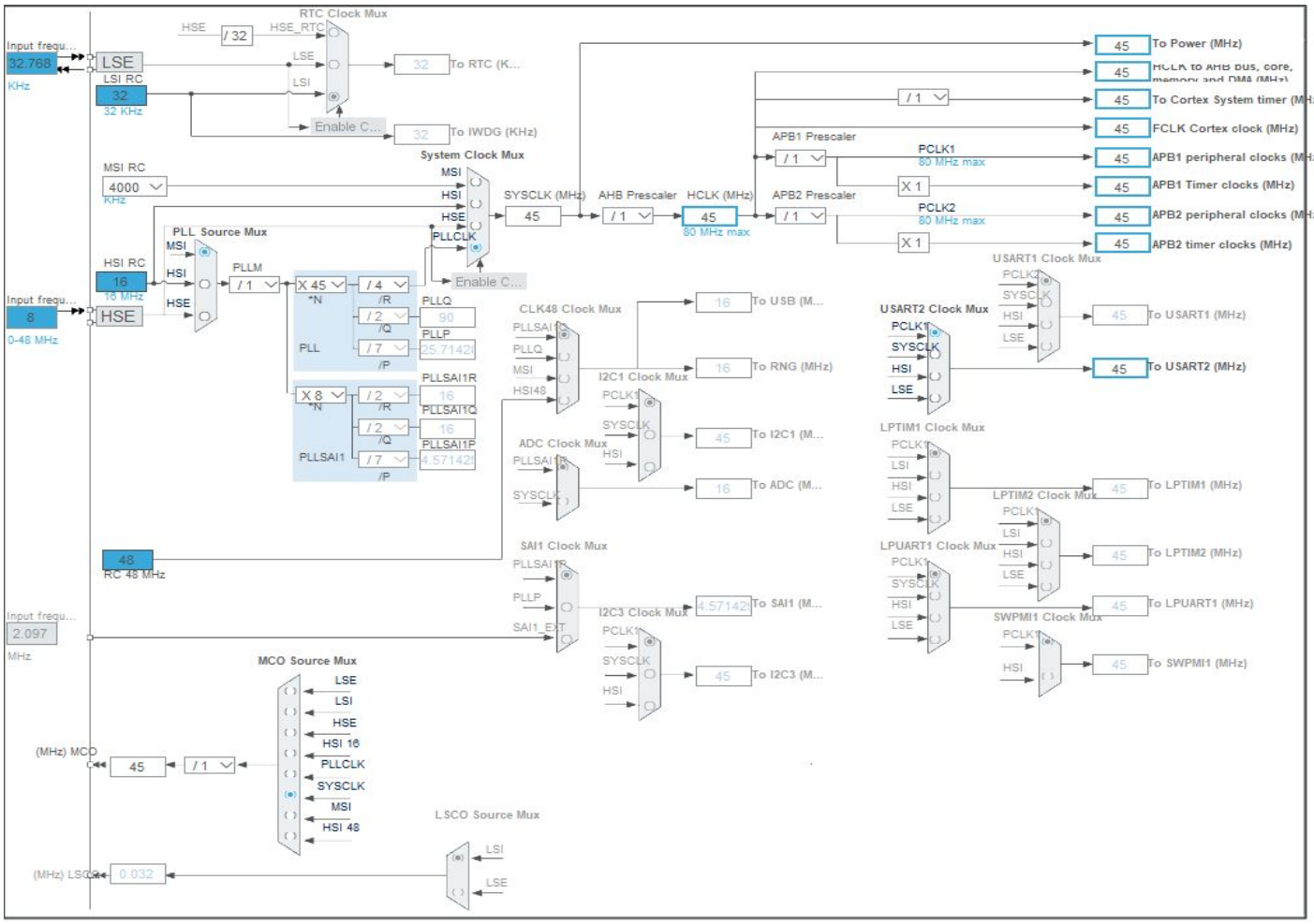
a) Finite State Machine Diagram:



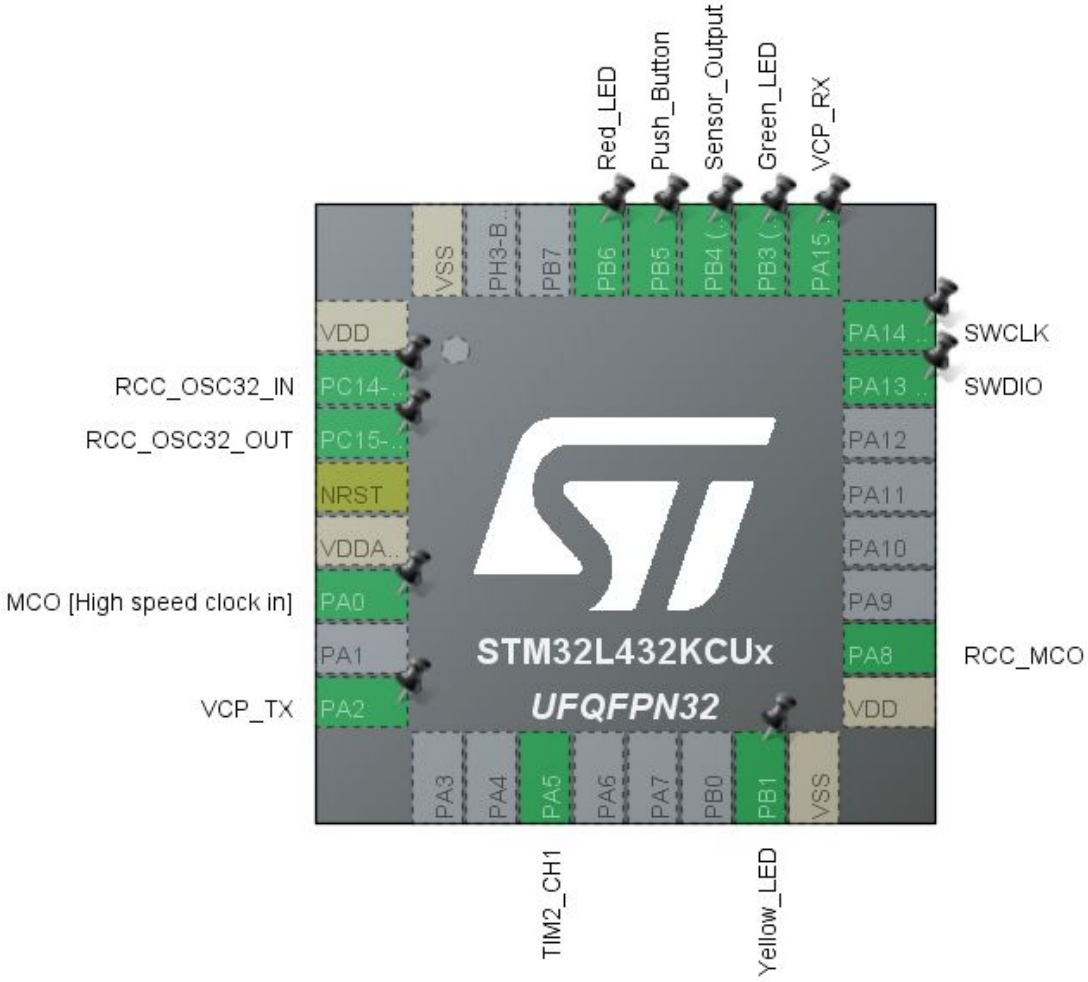
b) Finite State Machine Table:

Current State :	Input :	Next State :	Output :
Closed State	If signal is low → If signal is high →	- Return to Closed State - Go to Opening State	- Garage is closed - Garage starts to open
Opening State	Rotation = 180 degrees Motor will run for a preset rotation. No input.	Go to Opened State	Garage will be opened
Opened State	If signal is low → If signal is high →	Return to Opened State Go to Closing State	- Garage will remain open - Garage will begin to close
Closing State	If sensor is low → If sensor is high → (infrared emitter/ sensor)	- Go to Opening State - Rotation = - 180 degrees	- Garage will start to open - Garage will continue to close

c) Clock Configuration for Nucleo-L432KC



d) Pinout, GPIO, RCC, SYS, TIM, USART Configuration



GPIO | RCC | SYS | TIM | USART

Search Signals

Search (Crtl+F)

Show only Modified Pins

Pin Name	Signal on Pin	GPIO output l...	GPIO mode	GPIO Pull-up/...	Maximum out...	Fast Mode	User Label	Modified
PB1	n/a	Low	Output Push ...	No pull-up and...	Low	n/a	Yellow_LED	☑
PB3 (JTDO-T...	n/a	Low	Output Push ...	No pull-up and...	Low	n/a	Green_LED	☑
PB4 (NJTRST)	n/a	n/a	Input mode	No pull-up and...	n/a	n/a	Sensor_Output	☑
PB5	n/a	Low	Output Push ...	No pull-up and...	Low	n/a	Push_Button	☑
PB6	n/a	Low	Output Push ...	No pull-up and...	Low	Disable	Red_LED	☑

GPIO | RCC | SYS | TIM | USART

Search Signals

Search (Crtl+F)

Show only Modified Pins

Pin Name	Signal on Pin	GPIO output l...	GPIO mode	GPIO Pull-up/...	Maximum out...	Fast Mode	User Label	Modified
PA0	RCC_CK_IN	n/a	n/a	n/a	n/a	n/a	MCO [High sp...	☑
PA8	RCC_MCO	n/a	Alternate Fun...	No pull-up and...	Low	n/a		☐
PC14-OSC32...	RCC_OSC32_IN	n/a	n/a	n/a	n/a	n/a		☐
PC15-OSC32...	RCC_OSC32_...	n/a	n/a	n/a	n/a	n/a		☐

GPIO | RCC | SYS | TIM | USART

Search Signals

Search (Crtl+F)

Show only Modified Pins

Pin Name	Signal on Pin	GPIO output l...	GPIO mode	GPIO Pull-up/...	Maximum out...	Fast Mode	User Label	Modified
PA13 (JTMS-...	SYS_JTMS-S...	n/a	n/a	n/a	n/a	n/a	SWDIO	☑
PA14 (JTCK-...	SYS_JTCK-S...	n/a	n/a	n/a	n/a	n/a	SWCLK	☑

GPIO | RCC | SYS | TIM | USART

Search Signals

Search (Crtl+F)

Show only Modified Pins

Pin Name	Signal on Pin	GPIO output l...	GPIO mode	GPIO Pull-up/...	Maximum out...	Fast Mode	User Label	Modified
PA5	TIM2_CH1	n/a	Alternate Fun...	No pull-up and...	Low	n/a		☐

GPIO | RCC | SYS | TIM | USART

Search Signals

Search (Crtl+F)

Show only Modified Pins

Pin Name	Signal on Pin	GPIO output l...	GPIO mode	GPIO Pull-up/...	Maximum out...	Fast Mode	User Label	Modified
PA2	USART2_TX	n/a	Alternate Fun...	No pull-up and...	Very High	n/a	VCP_TX	☑
PA15 (JTDI)	USART2_RX	n/a	Alternate Fun...	No pull-up and...	Very High	n/a	VCP_RX	☑

e) C code to program Nucleo-L432KC (main.c file)

```c
/* USER CODE BEGIN Header */
/**
  ******************************************************************************
  * @file           : main.c
  * @brief          : Main program body
  ******************************************************************************
  * @attention
  *
  * <h2><center>&copy; Copyright (c) 2020 STMicroelectronics.
  * All rights reserved.</center></h2>
  *
  * This software component is licensed by ST under BSD 3-Clause license,
  * the "License"; You may not use this file except in compliance with the
  * License. You may obtain a copy of the License at:
  *                        opensource.org/licenses/BSD-3-Clause
  *
  ******************************************************************************
  */
/* USER CODE END Header */
/* Includes ------------------------------------------------------------------*/
#include "main.h"

/* Private includes ----------------------------------------------------------*/
/* USER CODE BEGIN Includes */

/* USER CODE END Includes */

/* Private typedef -----------------------------------------------------------*/
/* USER CODE BEGIN PTD */
typedef enum
{
    Door_Opened_State,
    Door_Opening_State,
    Door_Closing_State,
    Door_Closed_State,
} eSystemState;

/* USER CODE END PTD */

/* Private define ------------------------------------------------------------*/
/* USER CODE BEGIN PD */
/* USER CODE END PD */

/* Private macro -------------------------------------------------------------*/
/* USER CODE BEGIN PM */

/* USER CODE END PM */

/* Private variables ---------------------------------------------------------*/
TIM_HandleTypeDef htim2;

UART_HandleTypeDef huart2;
```

```
53
54  /* USER CODE BEGIN PV */
55
56  /* USER CODE END PV */
57
58  /* Private function prototypes -----------------------------------------------*/
59  void SystemClock_Config(void);
60  static void MX_GPIO_Init(void);
61  static void MX_USART2_UART_Init(void);
62  static void MX_TIM2_Init(void);
63  /* USER CODE BEGIN PFP */
64
65  /* USER CODE END PFP */
66
67  /* Private user code ---------------------------------------------------------*/
68  /* USER CODE BEGIN 0 */
69  int counter = 25;   // Motor Position Counter, 25 = 0 degrees, 125 = 180 degrees
70
71  eSystemState Door_Opened(void)
72  {
73      HAL_GPIO_WritePin(GPIOB, Green_LED_Pin, GPIO_PIN_SET);      // Green LED "ON", signify door opened
74      return Door_Opened_State;
75  }
76
77  eSystemState Door_Opening(void)
78  {
79      HAL_GPIO_WritePin(GPIOB, Red_LED_Pin, GPIO_PIN_RESET);      // Red LED "OFF", signify door closed
80      HAL_Delay(1000);    // Delay 1 second
81      HAL_GPIO_WritePin(GPIOB, Yellow_LED_Pin, GPIO_PIN_SET);     // Yellow LED "ON", signify door opening/closing
82      HAL_Delay(1000);    // Delay 1 second
83      if(counter == 25)
84      {
85          htim2.Instance->CCR1 = 125;             // Turn motor 125 degrees, fully opened
86          counter = 125;                          // Update Motor Position 180 degrees
87      }
88      else if(counter != 25)
89      {
90          while (counter != 125) {
91              counter = counter + 5;              // Inc Motor Position Counter
92              htim2.Instance->CCR1 = counter;     // Returning door to fully opened if sensor tripped
93          }
94      }
95      HAL_Delay(2000);    // Delay 2 seconds
96      HAL_GPIO_WritePin(GPIOB, Yellow_LED_Pin, GPIO_PIN_RESET);   // Yellow LED "OFF", signify door not opening/closing
97      return Door_Opened_State;
98  }
99
100 eSystemState Door_Closing(void)
101 {
102     HAL_GPIO_WritePin(GPIOB, Green_LED_Pin, GPIO_PIN_RESET);    // Green LED "OFF", signify door not opened
103     HAL_Delay(1000);    // Delay 1 second
104     HAL_GPIO_WritePin(GPIOB, Yellow_LED_Pin, GPIO_PIN_SET);     // Yellow LED "ON", signify door opening/closing
```

```
105     HAL_Delay(1000);    // Delay 1 second
106     while (counter != 25) {     // Motor control with sensor
107         if(HAL_GPIO_ReadPin(GPIOB, Sensor_Output_Pin) == GPIO_PIN_RESET)  // if sensor is tripped, open door
108             return Door_Opening_State;
109         counter = counter - 5;  // Decr Motor Position Counter
110         htim2.Instance->CCR1 = counter; // close door by 5 degrees
111         HAL_Delay(300); // Delay 300 milliseconds
112     }
113     HAL_Delay(2000);    // Delay 2 seconds
114     HAL_GPIO_WritePin(GPIOB, Yellow_LED_Pin, GPIO_PIN_RESET);   // Yellow LED "OFF", signify door done opening/closing
115     return Door_Closed_State;
116 }
117
118 eSystemState Door_Closed(void)
119 {
120     HAL_GPIO_WritePin(GPIOB, Red_LED_Pin, GPIO_PIN_SET);    // Red LED "ON", signify door closed
121     return Door_Closed_State;
122 }
123 /* USER CODE END 0 */
124
125 /**
126   * @brief  The application entry point.
127   * @retval int
128   */
129 int main(void)
130 {
131   /* USER CODE BEGIN 1 */
132
133   /* USER CODE END 1 */
134
135   /* MCU Configuration--------------------------------------------------------*/
136
137   /* Reset of all peripherals, Initializes the Flash interface and the Systick. */
138   HAL_Init();
139
140   /* USER CODE BEGIN Init */
141
142   /* USER CODE END Init */
143
144   /* Configure the system clock */
145   SystemClock_Config();
146
147   /* USER CODE BEGIN SysInit */
148
149   /* USER CODE END SysInit */
150
151   /* Initialize all configured peripherals */
152   MX_GPIO_Init();
153   MX_USART2_UART_Init();
154   MX_TIM2_Init();
155   /* USER CODE BEGIN 2 */
156   HAL_TIM_PWM_Start (&htim2, TIM_CHANNEL_1);
```

```
157   htim2.Instance->CCR1 = 25;
158   eSystemState eNextState = Door_Closed();
159   /* USER CODE END 2 */
160
161   /* Infinite loop */
162   /* USER CODE BEGIN WHILE */
163   while (1)
164   {
165       switch(eNextState)
166       {
167           case Door_Closed_State:
168               if(HAL_GPIO_ReadPin(GPIOB, Push_Button_Pin)) {
169                   eNextState = Door_Opening();          // Door button pressed
170                   break;
171               }
172               eNextState = Door_Closed();               // Door button not pressed
173               break;
174           case Door_Opening_State:
175               eNextState = Door_Opened();
176               break;
177           case Door_Opened_State:
178               if(HAL_GPIO_ReadPin(GPIOB, Push_Button_Pin)) {
179                   eNextState = Door_Closing();          // Door button pressed
180                   break;
181               }
182               eNextState = Door_Opened();               // Door button not pressed
183               break;
184           case Door_Closing_State:
185               eNextState = Door_Closed();
186               break;
187           default:
188               eNextState = Door_Closed();
189               break;
190       }
191     /* USER CODE END WHILE */
192
193     /* USER CODE BEGIN 3 */
194   }
195   /* USER CODE END 3 */
196 }
197
198 /**
199   * @brief System Clock Configuration
200   * @retval None
201   */
202 void SystemClock_Config(void)
203 {
204   RCC_OscInitTypeDef RCC_OscInitStruct = {0};
205   RCC_ClkInitTypeDef RCC_ClkInitStruct = {0};
206   RCC_PeriphCLKInitTypeDef PeriphClkInit = {0};
207
208   /** Configure LSE Drive Capability
209   */
```

```
210   HAL_PWR_EnableBkUpAccess();
211   __HAL_RCC_LSEDRIVE_CONFIG(RCC_LSEDRIVE_LOW);
212   /** Initializes the RCC Oscillators according to the specified parameters
213   * in the RCC_OscInitTypeDef structure.
214   */
215   RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_LSE|RCC_OSCILLATORTYPE_MSI;
216   RCC_OscInitStruct.LSEState = RCC_LSE_ON;
217   RCC_OscInitStruct.MSIState = RCC_MSI_ON;
218   RCC_OscInitStruct.MSICalibrationValue = 0;
219   RCC_OscInitStruct.MSIClockRange = RCC_MSIRANGE_6;
220   RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
221   RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_MSI;
222   RCC_OscInitStruct.PLL.PLLM = 1;
223   RCC_OscInitStruct.PLL.PLLN = 45;
224   RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV7;
225   RCC_OscInitStruct.PLL.PLLQ = RCC_PLLQ_DIV2;
226   RCC_OscInitStruct.PLL.PLLR = RCC_PLLR_DIV4;
227   if (HAL_RCC_OscConfig(&RCC_OscInitStruct) != HAL_OK)
228   {
229     Error_Handler();
230   }
231   /** Initializes the CPU, AHB and APB buses clocks
232   */
233   RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_HCLK|RCC_CLOCKTYPE_SYSCLK
234                                 |RCC_CLOCKTYPE_PCLK1|RCC_CLOCKTYPE_PCLK2;
235   RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
236   RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
237   RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV1;
238   RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV1;
239
240   if (HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_2) != HAL_OK)
241   {
242     Error_Handler();
243   }
244   PeriphClkInit.PeriphClockSelection = RCC_PERIPHCLK_USART2;
245   PeriphClkInit.Usart2ClockSelection = RCC_USART2CLKSOURCE_PCLK1;
246   if (HAL_RCCEx_PeriphCLKConfig(&PeriphClkInit) != HAL_OK)
247   {
248     Error_Handler();
249   }
250   HAL_RCC_MCOConfig(RCC_MCO1, RCC_MCO1SOURCE_SYSCLK, RCC_MCODIV_1);
251   /** Configure the main internal regulator output voltage
252   */
253   if (HAL_PWREx_ControlVoltageScaling(PWR_REGULATOR_VOLTAGE_SCALE1) != HAL_OK)
254   {
255     Error_Handler();
256   }
257   /** Enable MSI Auto calibration
258   */
259   HAL_RCCEx_EnableMSIPLLMode();
260 }
261
262 /**
```

```
263   * @brief TIM2 Initialization Function
264   * @param None
265   * @retval None
266   */
267 static void MX_TIM2_Init(void)
268 {
269
270   /* USER CODE BEGIN TIM2_Init 0 */
271
272   /* USER CODE END TIM2_Init 0 */
273
274   TIM_ClockConfigTypeDef sClockSourceConfig = {0};
275   TIM_MasterConfigTypeDef sMasterConfig = {0};
276   TIM_OC_InitTypeDef sConfigOC = {0};
277
278   /* USER CODE BEGIN TIM2_Init 1 */
279
280   /* USER CODE END TIM2_Init 1 */
281   htim2.Instance = TIM2;
282   htim2.Init.Prescaler = 900-1;
283   htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
284   htim2.Init.Period = 1000-1;
285   htim2.Init.ClockDivision = TIM_CLOCKDIVISION_DIV1;
286   htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
287   if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
288   {
289     Error_Handler();
290   }
291   sClockSourceConfig.ClockSource = TIM_CLOCKSOURCE_INTERNAL;
292   if (HAL_TIM_ConfigClockSource(&htim2, &sClockSourceConfig) != HAL_OK)
293   {
294     Error_Handler();
295   }
296   if (HAL_TIM_PWM_Init(&htim2) != HAL_OK)
297   {
298     Error_Handler();
299   }
300   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
301   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
302   if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
303   {
304     Error_Handler();
305   }
306   sConfigOC.OCMode = TIM_OCMODE_PWM1;
307   sConfigOC.Pulse = 0;
308   sConfigOC.OCPolarity = TIM_OCPOLARITY_HIGH;
309   sConfigOC.OCFastMode = TIM_OCFAST_DISABLE;
310   if (HAL_TIM_PWM_ConfigChannel(&htim2, &sConfigOC, TIM_CHANNEL_1) != HAL_OK)
311   {
312     Error_Handler();
313   }
314   /* USER CODE BEGIN TIM2_Init 2 */
315
```

```
316   /* USER CODE END TIM2_Init 2 */
317   HAL_TIM_MspPostInit(&htim2);
318
319 }
320
321 /**
322   * @brief USART2 Initialization Function
323   * @param None
324   * @retval None
325   */
326 static void MX_USART2_UART_Init(void)
327 {
328
329   /* USER CODE BEGIN USART2_Init 0 */
330
331   /* USER CODE END USART2_Init 0 */
332
333   /* USER CODE BEGIN USART2_Init 1 */
334
335   /* USER CODE END USART2_Init 1 */
336   huart2.Instance = USART2;
337   huart2.Init.BaudRate = 115200;
338   huart2.Init.WordLength = UART_WORDLENGTH_8B;
339   huart2.Init.StopBits = UART_STOPBITS_1;
340   huart2.Init.Parity = UART_PARITY_NONE;
341   huart2.Init.Mode = UART_MODE_TX_RX;
342   huart2.Init.HwFlowCtl = UART_HWCONTROL_NONE;
343   huart2.Init.OverSampling = UART_OVERSAMPLING_16;
344   huart2.Init.OneBitSampling = UART_ONE_BIT_SAMPLE_DISABLE;
345   huart2.AdvancedInit.AdvFeatureInit = UART_ADVFEATURE_NO_INIT;
346   if (HAL_UART_Init(&huart2) != HAL_OK)
347   {
348     Error_Handler();
349   }
350   /* USER CODE BEGIN USART2_Init 2 */
351
352   /* USER CODE END USART2_Init 2 */
353
354 }
355
356 /**
357   * @brief GPIO Initialization Function
358   * @param None
359   * @retval None
360   */
361 static void MX_GPIO_Init(void)
362 {
363   GPIO_InitTypeDef GPIO_InitStruct = {0};
364
365   /* GPIO Ports Clock Enable */
366   __HAL_RCC_GPIOC_CLK_ENABLE();
367   __HAL_RCC_GPIOA_CLK_ENABLE();
368   __HAL_RCC_GPIOB_CLK_ENABLE();
```

```
369
370   /*Configure GPIO pin Output Level */
371   HAL_GPIO_WritePin(GPIOB, Yellow_LED_Pin|Green_LED_Pin|Push_Button_Pin|Red_LED_Pin, GPIO_PIN_RESET);
372
373   /*Configure GPIO pins : Yellow_LED_Pin Green_LED_Pin Push_Button_Pin Red_LED_Pin */
374   GPIO_InitStruct.Pin = Yellow_LED_Pin|Green_LED_Pin|Push_Button_Pin|Red_LED_Pin;
375   GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
376   GPIO_InitStruct.Pull = GPIO_NOPULL;
377   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
378   HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
379
380   /*Configure GPIO pin : PA8 */
381   GPIO_InitStruct.Pin = GPIO_PIN_8;
382   GPIO_InitStruct.Mode = GPIO_MODE_AF_PP;
383   GPIO_InitStruct.Pull = GPIO_NOPULL;
384   GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
385   GPIO_InitStruct.Alternate = GPIO_AF0_MCO;
386   HAL_GPIO_Init(GPIOA, &GPIO_InitStruct);
387
388   /*Configure GPIO pin : Sensor_Output_Pin */
389   GPIO_InitStruct.Pin = Sensor_Output_Pin;
390   GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
391   GPIO_InitStruct.Pull = GPIO_NOPULL;
392   HAL_GPIO_Init(Sensor_Output_GPIO_Port, &GPIO_InitStruct);
393
394 }
395
396 /* USER CODE BEGIN 4 */
397
398 /* USER CODE END 4 */
399
400 /**
401   * @brief  This function is executed in case of error occurrence.
402   * @retval None
403   */
404 void Error_Handler(void)
405 {
406   /* USER CODE BEGIN Error_Handler_Debug */
407   /* User can add his own implementation to report the HAL error return state */
408
409   /* USER CODE END Error_Handler_Debug */
410 }
411
412 #ifdef  USE_FULL_ASSERT
413 /**
414   * @brief  Reports the name of the source file and the source line number
415   *         where the assert_param error has occurred.
416   * @param  file: pointer to the source file name
417   * @param  line: assert_param error line source number
418   * @retval None
419   */
420 void assert_failed(uint8_t *file, uint32_t line)
421 {
422   /* USER CODE BEGIN 6 */
423   /* User can add his own implementation to report the file name and line number,
424      tex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
425   /* USER CODE END 6 */
426 }
427 #endif /* USE_FULL_ASSERT */
428
429 /************************ (C) COPYRIGHT STMicroelectronics *****END OF FILE****/
430
```

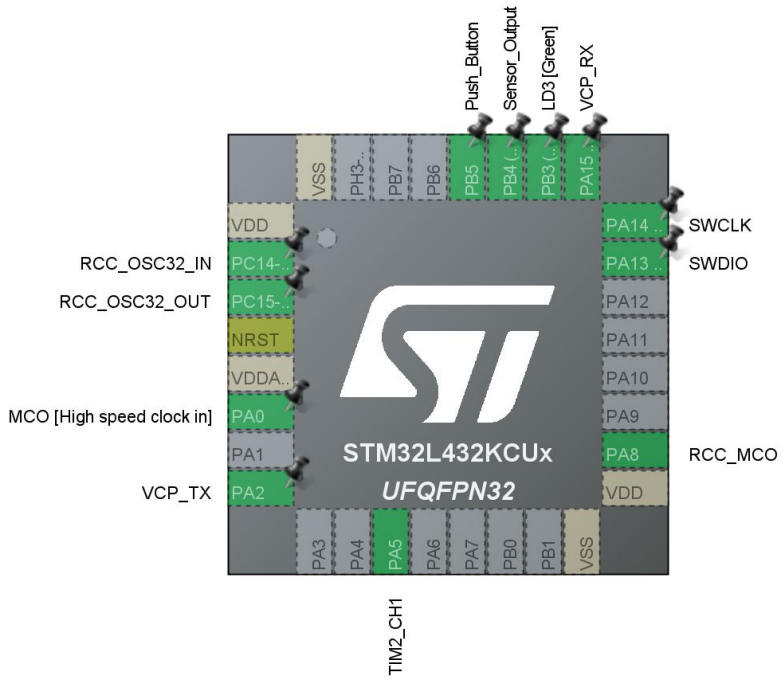
3) Connecting the Hardware Components

a) Test Case #3: Garage Door - Motor - Sensor - Switch

All of the hardware was connected to the Nucleo-L432KC.

Both the switch and Sensor were connected to the motor using GPIO pins on the Nucleo board. The idea was to have them as conditions in order for the motor to run using nested if statements. So if the sensor input was high, and the button input was high, then the motor would turn. Otherwise the motor would not turn.

The pinout included the PWM timer output on PA5 for the motor, the Sensor input on PB4, and the switch input on PB5.



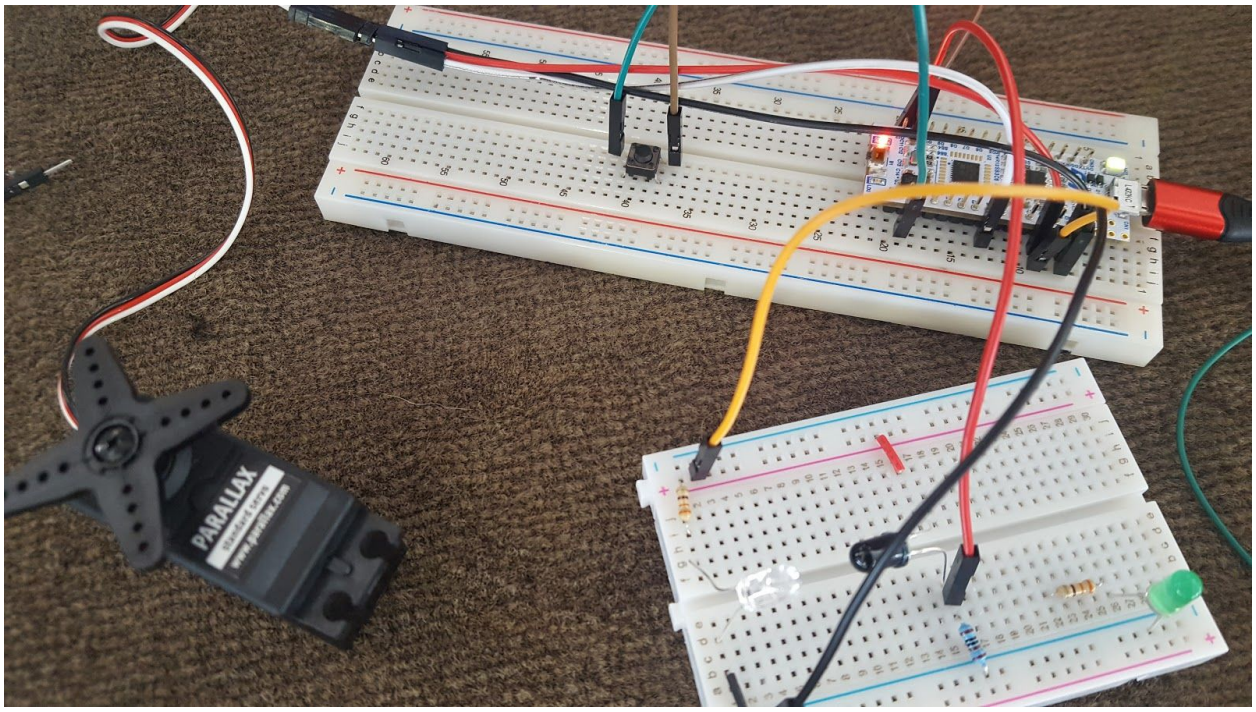
The code was kept all in the while loop, and included a nested if block with two if statements, along with commands to rotate the motor.

```
104   while (1)
105   {
106     /* USER CODE END WHILE */
107       if(HAL_GPIO_ReadPin(GPIOB, Read_Sensor_Pin))
108       {
109           if(HAL_GPIO_ReadPin(GPIOB, Push_Button_Pin))
110               {
111               htim2.Instance->CCR1 = 25;
112               HAL_Delay(2000);
113               htim2.Instance->CCR1 = 75;
114               HAL_Delay(2000);
115               htim2.Instance->CCR1 = 125;
116               HAL_Delay(2000);
117               // Yellow pin on
118               }
119         }
120       else
121       {
122           // Red pin on
123       }
124
125     /* USER CODE BEGIN 3 */
126   }
```

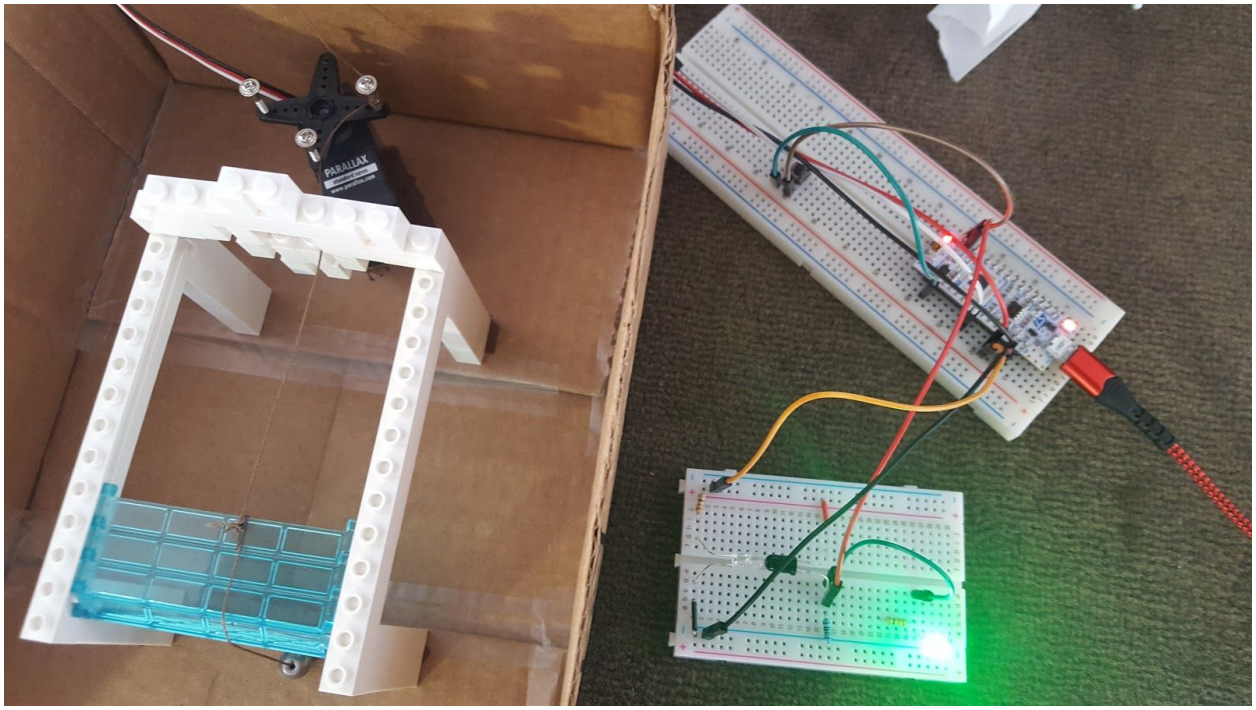
The most difficult and unreliable connection was the mechanical connection between the motor and Lego garage door. Without having enough time to order the proper equipment, we were forced to innovate methods in order to have a decent connection between the motor and door. The goal was for the door to open and close when the motor ran.

This was accomplished using the following tools: counter-weight, twine, nails, glue, and a cardboard box. The cardboard box was to help glue all of the components in place. The glue helped keep all of the components in place and was necessary in order for the force exerted on the garage door not to pull the whole Lego Garage itself. The nails were attached to the gear which was attached to the motor. Then the twine was coiled around the nails so when the gear moved, the nails moved, and the twine pulled and pushed whatever it was connected to. The twine was then connected to the counter-weight which was connected to the garage by tying a knot between the door rungs. Now when the motor moved, the nails moved, which turned the twine and pushed or pulled the garage door. The counter-weight is used in modern garage doors, and assists the garage to close because the force of gravity pulls it down more.

Below is the connection between the Nucleo board, sensor, switch, and motor.



The connection between all of the components in this test case are displayed below including the mechanical connections of the counter-weight, twine, nails, motor, and garage door.

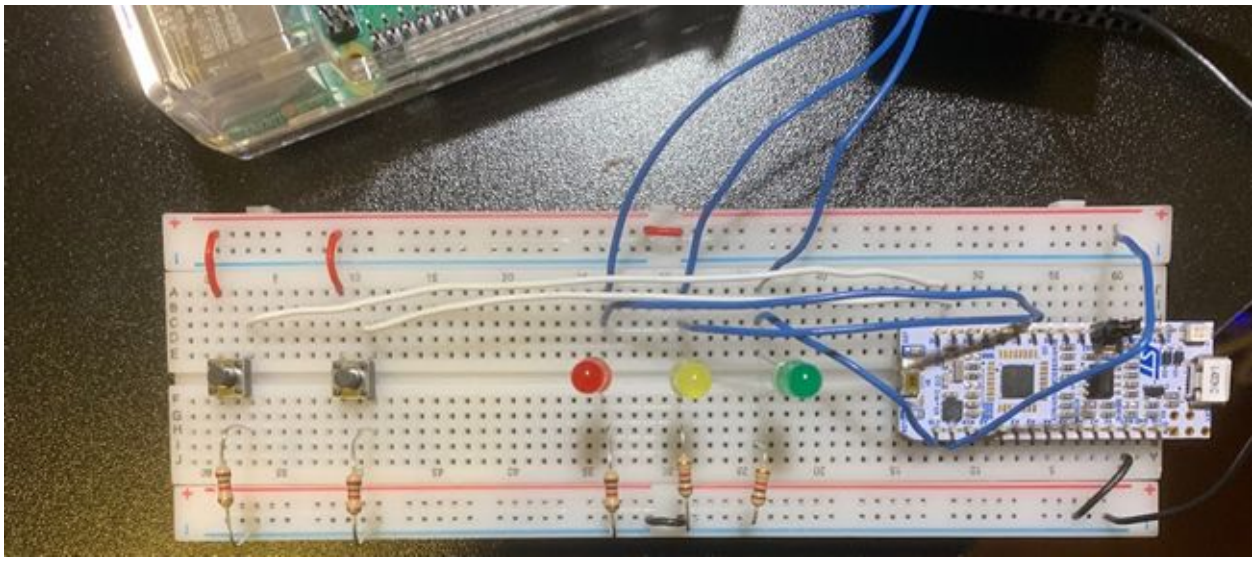


Here is a Youtube link to demonstrate Test Case #3:
https://www.youtube.com/watch?v=S4iiti6QSUQ

4) Raspberry-Pi Data Logging

For the data logging portion of the project, we used the raspberry pi to keep track of the garage door's state. One program called 'log.py' is used to display the state of the garage in the terminal and write to a text file which can be accessed through the web server. The web server simply shows the state of the garage using pictures of a closed garage when in the closed state, a question mark when in the closing/opening state, and an opened garage when in the opened state. However, the user must press the F5 button on the keyboard to refresh the web page to get the current garage door state. Please note that Chrome has a cache bug, so we used Mozilla Firefox. There is also a link to access the data log text file created by the log.py program.

a) Hardware Set-up

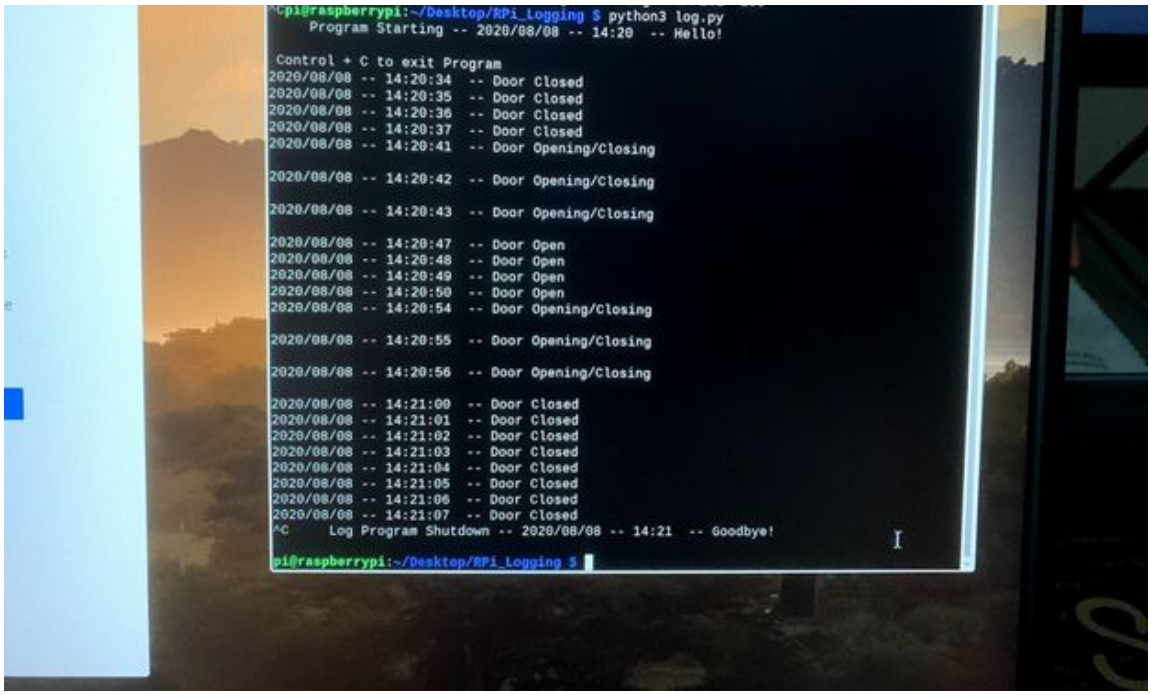


b) Software Set-up

```python
import os
import RPi.GPIO as GPIO
import time
from datetime import datetime


logfile = open("/home/pi/Desktop/RPi_Logging/static/log.txt","a")
logfile.write(datetime.now().strftime("     Program Starting -- %Y/%m/%d -- %H:%M  -- Hello! \n"))
logfile.close()
print(datetime.now().strftime("     Program Starting -- %Y/%m/%d -- %H:%M  -- Hello! \n"))

print (" Control + C to exit Program")

GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)

GPIO.setup(17, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)   # Green LED
GPIO.setup(27, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)   # Yellow LED
GPIO.setup(22, GPIO.IN, pull_up_down=GPIO.PUD_DOWN)   # Red LED
time.sleep(1)

try:
        while 1 >= 0:
         time.sleep(1)

         if GPIO.input(17) == GPIO.LOW and GPIO.input(27) == GPIO.HIGH and GPIO.input(22) == GPIO.LOW:  # Door Closing/Opening
           logfile = open("/home/pi/Desktop/RPi_Logging/static/log.txt","a")
           logfile.write(datetime.now().strftime("%Y/%m/%d -- %H:%M:%S  -- Door Opening/Closing \n"))
           logfile.close()
           print(datetime.now().strftime("%Y/%m/%d -- %H:%M:%S  -- Door Opening/Closing \n"))

         elif GPIO.input(17) == GPIO.LOW and GPIO.input(27) == GPIO.LOW and GPIO.input(22) == GPIO.HIGH:  # Door is Closed
           logfile = open("/home/pi/Desktop/RPi_Logging/static/log.txt","a")
           logfile.write(datetime.now().strftime("%Y/%m/%d -- %H:%M:%S  -- Door Closed \n"))
           logfile.close()
           print(datetime.now().strftime("%Y/%m/%d -- %H:%M:%S  -- Door Closed"))

         elif GPIO.input(17) == GPIO.HIGH and GPIO.input(27) == GPIO.LOW and GPIO.input(22) == GPIO.LOW:  # Door is Open
           logfile = open("/home/pi/Desktop/RPi_Logging/static/log.txt","a")
           logfile.write(datetime.now().strftime("%Y/%m/%d -- %H:%M:%S  -- Door Open \n"))
           logfile.close()
           print(datetime.now().strftime("%Y/%m/%d -- %H:%M:%S  -- Door Open"))


except KeyboardInterrupt:   # Handle Ctrl + C
        logfile = open("/home/pi/Desktop/RPi_Logging/static/log.txt","a")
        logfile.write(datetime.now().strftime("     Log Program Shutdown -- %Y/%m/%d -- %H:%M  -- Goodbye! \n"))
```

```python
           print(datetime.now().strftime("%Y/%m/%d -- %H:%M:%S  -- Door Open"))


except KeyboardInterrupt:   # Handle Ctrl + C
        logfile = open("/home/pi/Desktop/RPi_Logging/static/log.txt","a")
        logfile.write(datetime.now().strftime("     Log Program Shutdown -- %Y/%m/%d -- %H:%M  -- Goodbye! \n"))
        logfile.close()
        print(datetime.now().strftime("     Log Program Shutdown -- %Y/%m/%d -- %H:%M  -- Goodbye! \n"))
        GPIO.cleanup()
```

log.py program, logs status of garage door in text file, can be seen from webserver



Example of log.py program running using the Nucleo Test code in Part C

```python
import time
from datetime import datetime
from flask import Flask, render_template, request

import RPi.GPIO as GPIO
GPIO.setmode(GPIO.BCM)
GPIO.setwarnings(False)
GPIO.setup(17, GPIO.IN, GPIO.PUD_DOWN) # set up pin 17 as an input with a pull-down resistor
GPIO.setup(27, GPIO.IN, GPIO.PUD_DOWN) # set up pin 27 as an input with a pull-down resistor
GPIO.setup(22, GPIO.IN, GPIO.PUD_DOWN) # set up pin 22 as an input with a pull-down resistor

app = Flask(__name__)

@app.route('/', methods=['GET', 'POST'])
def index():

    if GPIO.input(17) == GPIO.LOW and GPIO.input(27) == GPIO.LOW and GPIO.input(22) == GPIO.HIGH:
        print ("Garage is Closed")
        return app.send_static_file('Closed.html')
    elif GPIO.input(17) == GPIO.LOW and GPIO.input(27) == GPIO.HIGH and GPIO.input(22) == GPIO.LOW:
        print("Garage is Opening/Closing")
        return app.send_static_file('Closing-Opening.html')
    elif GPIO.input(17) == GPIO.HIGH and GPIO.input(27) == GPIO.LOW and GPIO.input(22) == GPIO.LOW:
        print ("Garage is Open")
        return app.send_static_file('Open.html')

@app.route('/stylesheet.css')
def stylesheet():
        return app.send_static_file('stylesheet.css')

@app.route('/Log')
def logfile():
        return app.send_static_file('log.txt')

@app.route('/images/<picture>')
def images(picture):
        return app.send_static_file('images/' + picture)

if __name__ == '__main__':
        app.run(debug=True, host='0.0.0.0', port=5000)
```

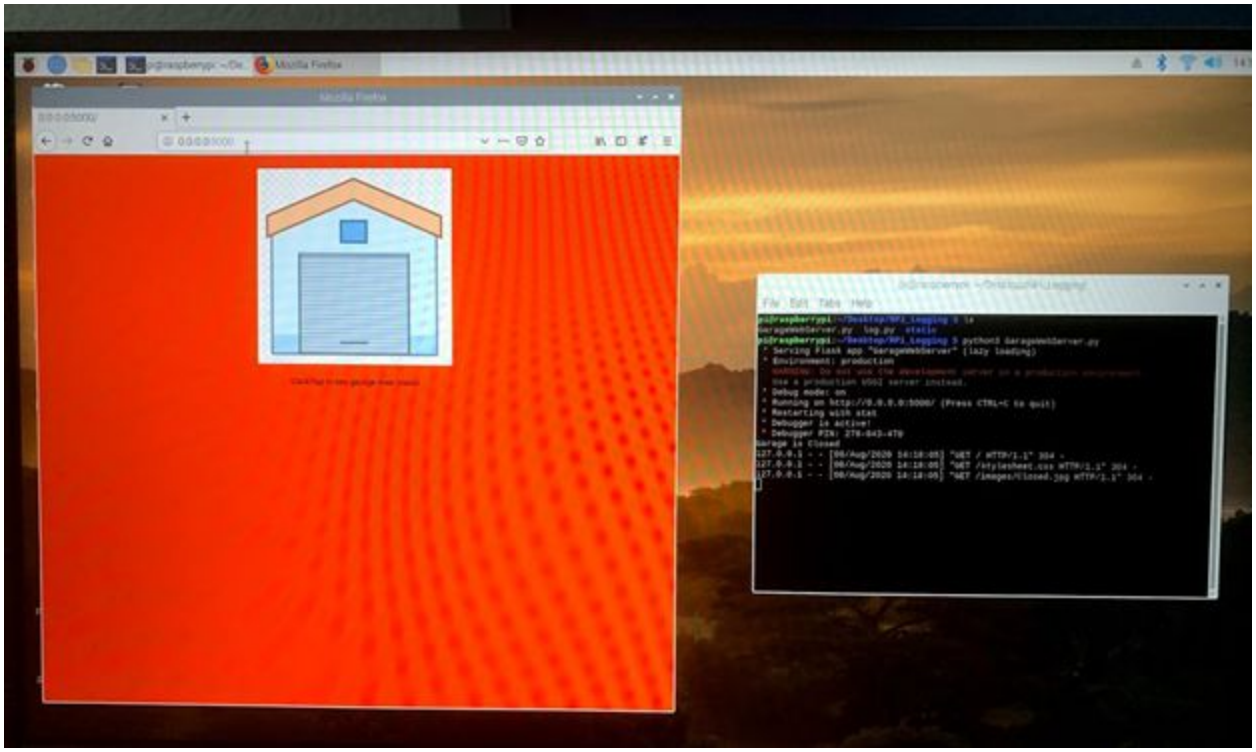
GarageWebServer.py program, Web Server is hosted when running this program

```css
#container {
margin-left: auto;
margin-right: auto;
}

#backtomain {
vertical-align:text-middle;
text-align: center;
float: left;
width: 40px;
height: 40px;
background-color: #88B655;
-moz-border-radius: 15px;
-webkit-border-radius: 15px;
padding: 5px;
}

table, td {
border: 0px;
padding: 0px;
border-spacing: 0px;
}

body {
cursor:crosshair
margin-left: 0pt;
margin-left: 0px;
margin-right: 0px;
margin-top: 0px;
margin-bottom: 0px;
font-family: Trebuchet MS, Verdana, Arial, Helvetica, sans-serif;
font-size:18px;
}

a:link   {font-family:Arial; font-size:9pt; font-weight:bold; color:#1D1D1D; text-decoration: none}
a:visited {font-family:Arial; font-size:9pt; font-weight:bold; color:#1D1D1D; text-decoration: none}
a:active  {font-family:Arial; font-size:9pt; font-weight:bold; color:#1D1D1D; text-decoration: none}
a:hover   {font-family:Arial; font-size:9pt; font-weight:bold; color:#6E6E6E; text-decoration: none;cursor:hand}

container {
    width: 399px;
    border: 1px solid black;
    display: inline-block;
}

block {
    display: inline-block;
    width: 123px;
    height: 100px;
    margin: 5px;
    background: #6c6969;
    float: right;
    font-size:35px;
}

block[h2] {
    width: 256px;
}
```

stylesheet.css, controls the layout of the Web Page

```
<html><head>
<meta name="viewport" content="width=device-width">
<link rel="stylesheet" type="text/css" href="stylesheet.css">

</head>
<body bgcolor="red">
<center>
<br>
<img src="/images/Closed.jpg" width="30%">
<br><br><a href="/Log">Click/Tap to see garage door status</a><br><br>
</center>
</body></html>
```

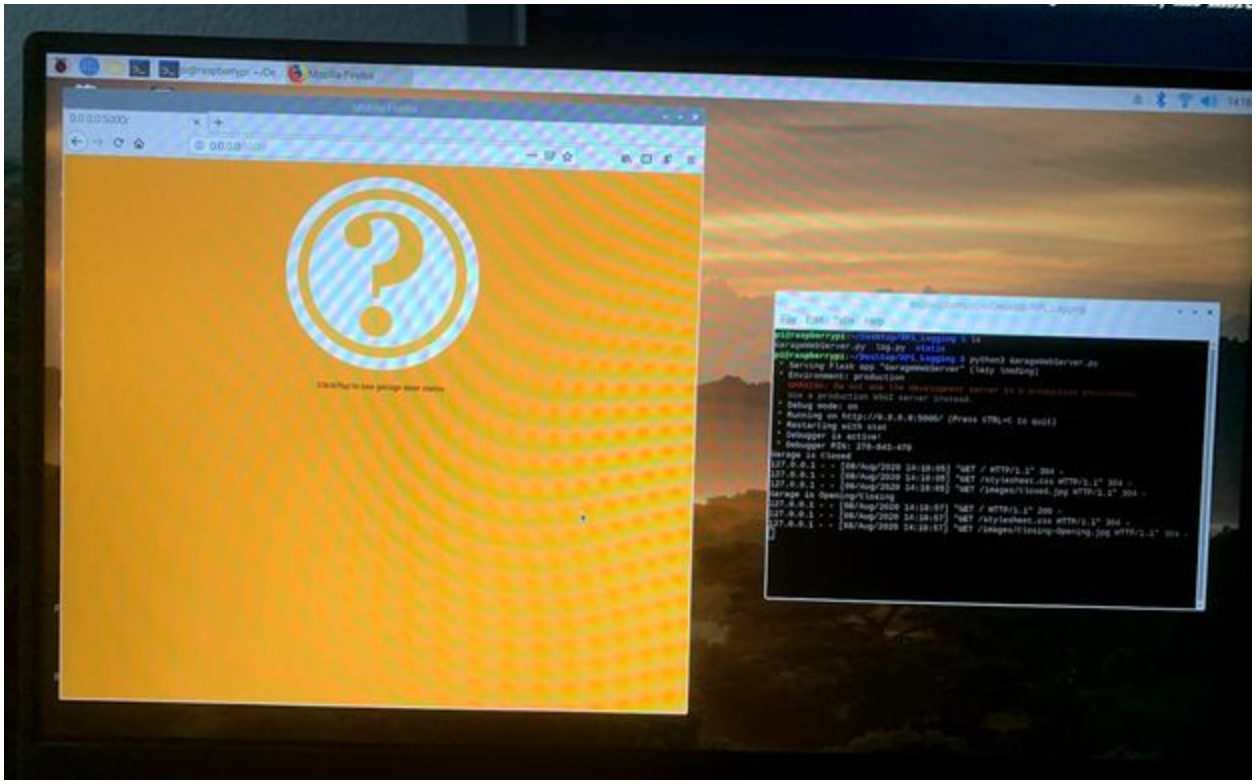
Closed.html, controls the layout of Web Page when garage is closed



GarageWebServer closed state

```
<html><head>
<meta name="viewport" content="width=device-width">
<link rel="stylesheet" type="text/css" href="stylesheet.css">

</head>
<body bgcolor="orange">
<center>
<br>
<img src="/images/Closing-Opening.jpg" width="30%">
<br><br><a href="/Log">Click/Tap to see garage door status</a><br><br>
</center>
</body></html>
```

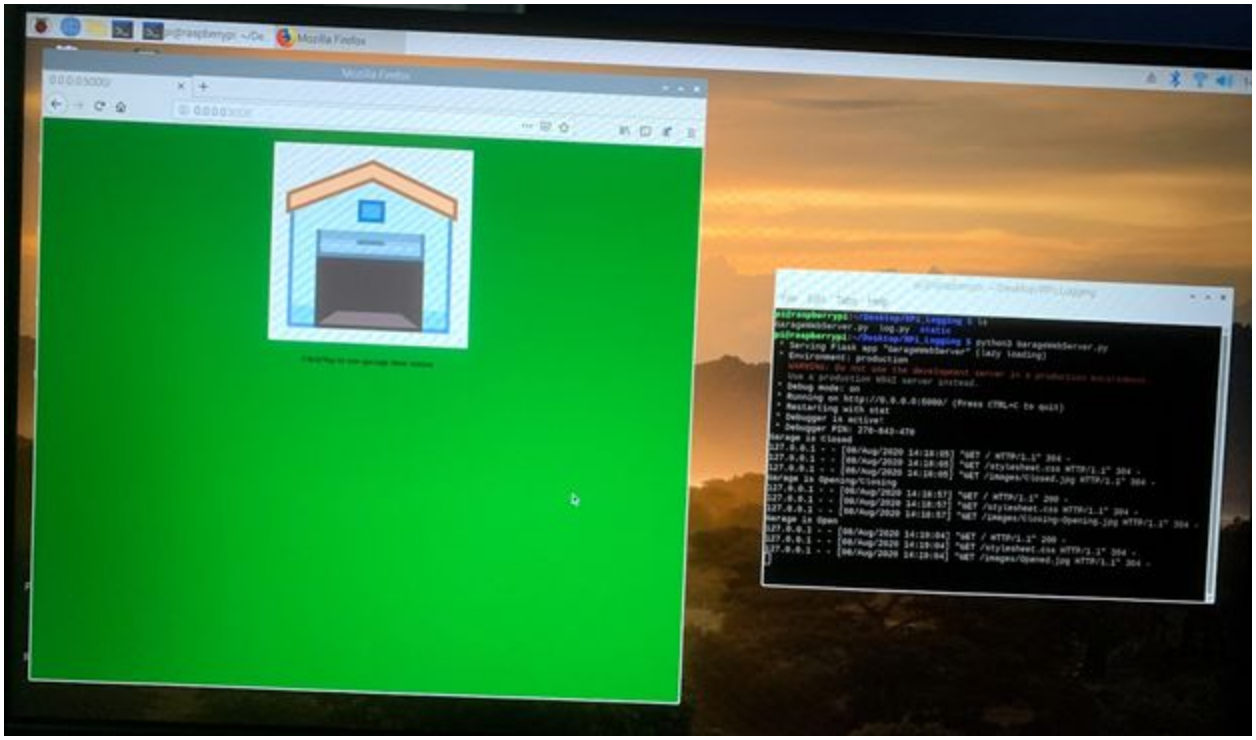
Closing-Opening.html, controls the layout of Web Page when garage is closing/opening



GarageWebServer closing/opening state

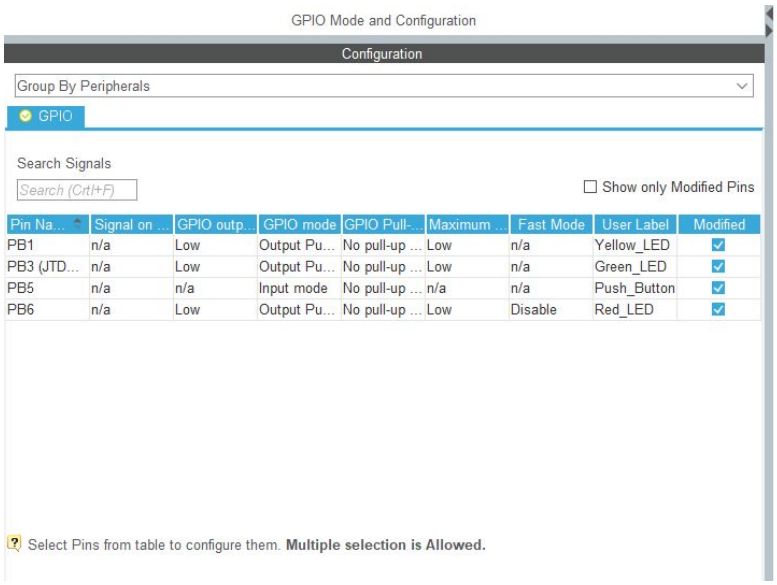
```
<html><head>
<meta name="viewport" content="width=device-width">
<link rel="stylesheet" type="text/css" href="stylesheet.css">

</head>
<body bgcolor="green">
<center>
<br>
<img src="/images/Opened.jpg" width="30%">
<br><br><a href="/Log">Click/Tap to see garage door status</a><br><br>
</center>
</body></html>
```

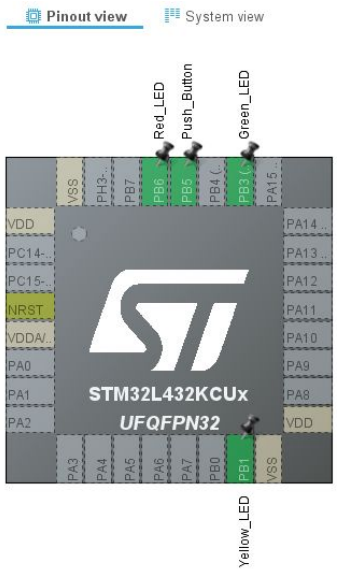
Open.html, controls the layout of Web Page when garage opened



GarageWebServer opened state

c) Test Code





GPIO and Pinout Configuration

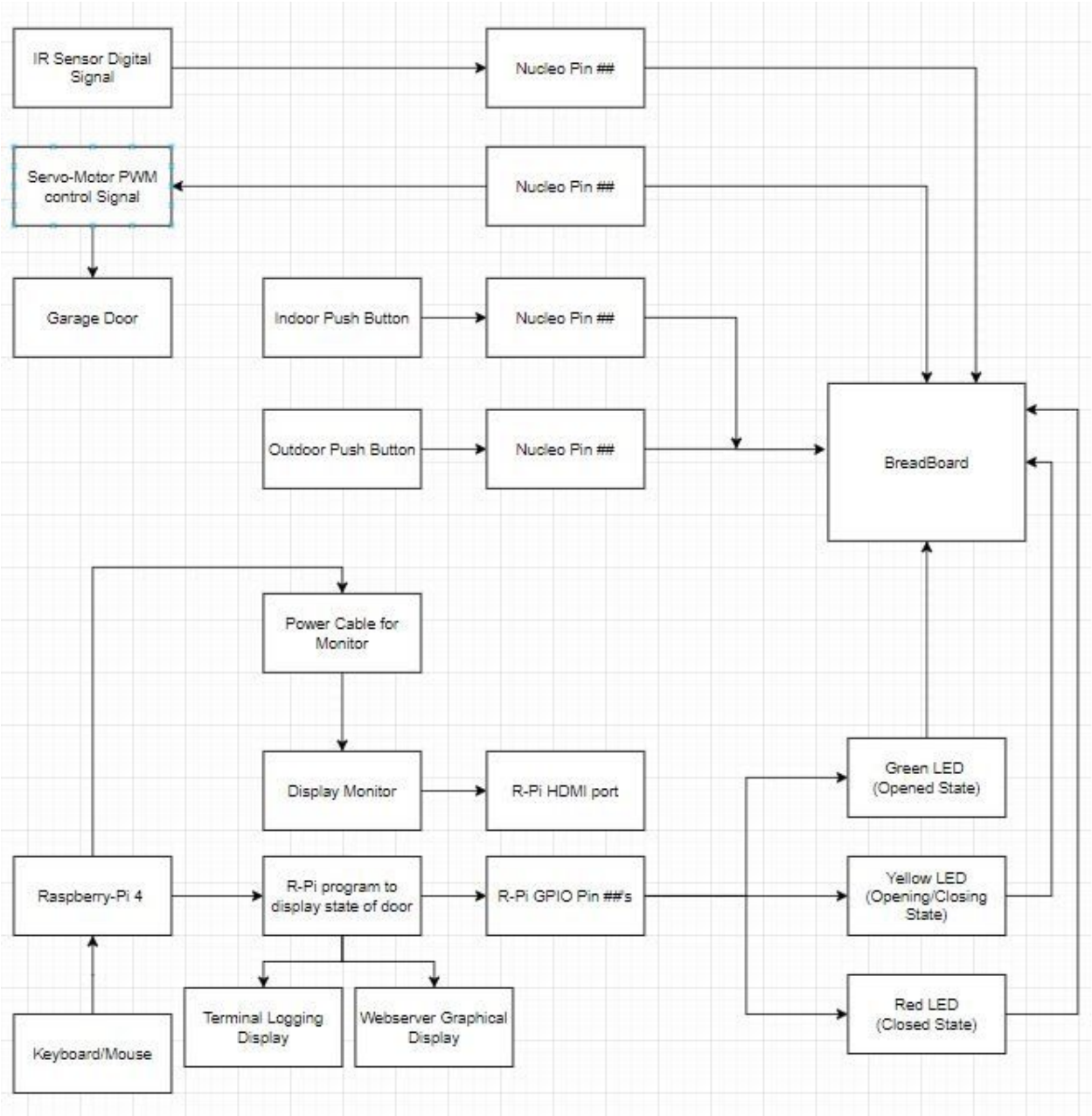
```c
while (1)
{
    if(HAL_GPIO_ReadPin(GPIOB, Push_Button_Pin)) {
        HAL_GPIO_WritePin(GPIOB, Red_LED_Pin, GPIO_PIN_SET);          // Red LED "ON", signify door closed
        HAL_Delay(3000);  // Delay 3 seconds
        HAL_GPIO_WritePin(GPIOB, Red_LED_Pin, GPIO_PIN_RESET);        // Red LED "OFF", signify door closed
        HAL_Delay(3000);  // Delay 3 second
        HAL_GPIO_WritePin(GPIOB, Yellow_LED_Pin, GPIO_PIN_SET);       // Yellow LED "ON", signify door opening/closing
        HAL_Delay(3000);  // Delay 3 seconds
        HAL_GPIO_WritePin(GPIOB, Yellow_LED_Pin, GPIO_PIN_RESET);     // Yellow LED "OFF", signify door not opening/closing
        HAL_Delay(3000);  // Delay 3 seconds
        HAL_GPIO_WritePin(GPIOB, Green_LED_Pin, GPIO_PIN_SET);        // Green LED "ON", signify door opened
        HAL_Delay(3000);  // Delay 3 seconds
        HAL_GPIO_WritePin(GPIOB, Green_LED_Pin, GPIO_PIN_RESET);      // Green LED "OFF", signify door no opened
        HAL_Delay(3000);  // Delay 3 seconds
        HAL_GPIO_WritePin(GPIOB, Yellow_LED_Pin, GPIO_PIN_SET);       // Yellow LED "ON", signify door opening/closing
        HAL_Delay(3000);  // Delay 3 seconds
        HAL_GPIO_WritePin(GPIOB, Yellow_LED_Pin, GPIO_PIN_RESET);     // Yellow LED "OFF", signify door done opening/closing
        HAL_Delay(3000);  // Delay 3 seconds
        HAL_GPIO_WritePin(GPIOB, Red_LED_Pin, GPIO_PIN_SET);          // Red LED "ON", signify door closed
    }
    else{}
```

Added to main.c in infinite loop

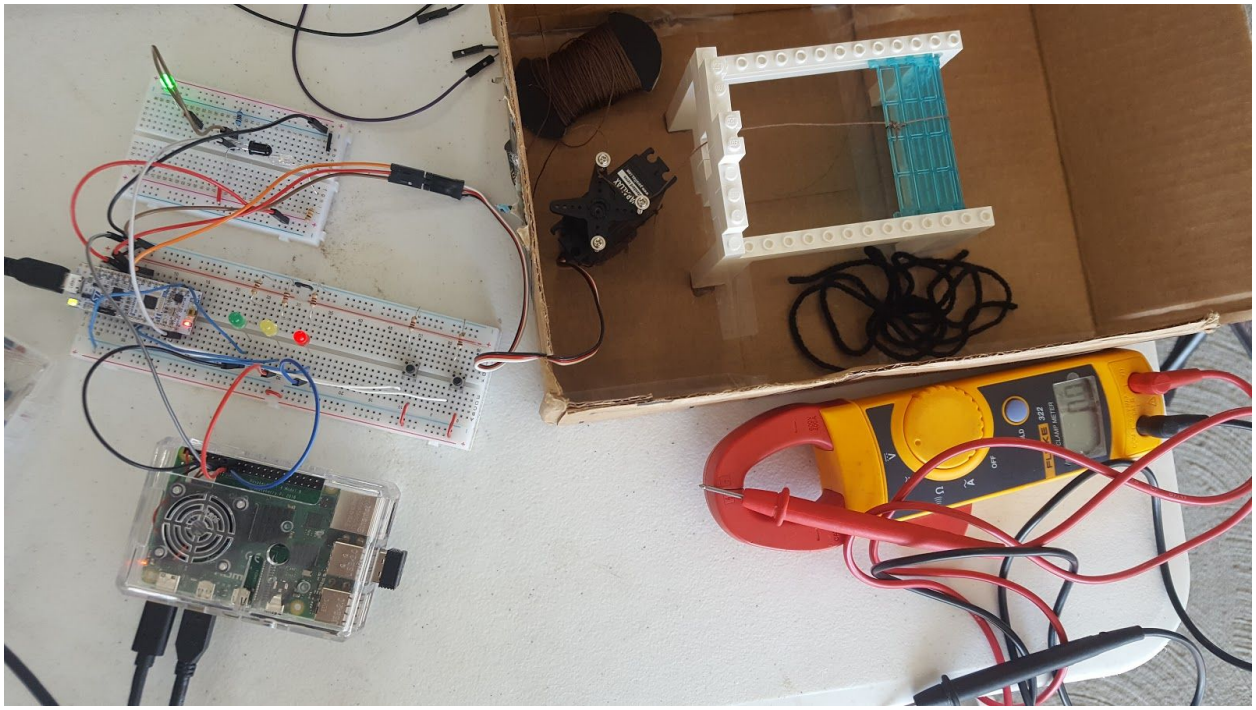
5) Completed Design Set-up and Demonstration

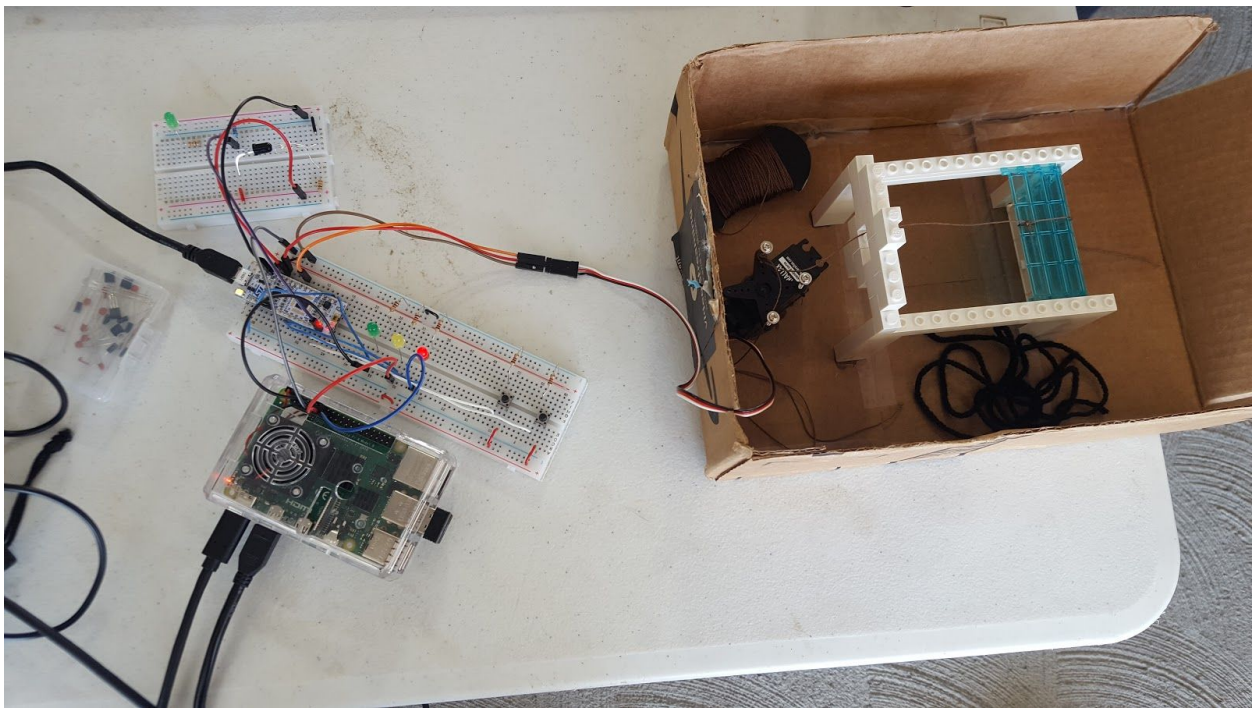
We recorded a video for the final demonstration that was posted on Moodle. We made a presentation that goes through the final set-up, and demo.

For the final demonstration, we connected all of the components exactly as shown in the block diagram below. A similar block diagram was also included in the final project proposal.



The hardware Set-Up was meticulously done yielding the results below:





Once the hardware was set-up, we recorded the function of our state-machine.

Here is the link of the private Youtube video of our Final Demo:

https://youtube/BtZ7_FBKz7s

Sources Cited:

LED/Phototransistor

https://www.ttelectronics.com/TTElectronics/media/ProductFiles/Optoelectronics/Datasheets/OPB100.pdf

http://www.gikfun.com/leds-c-6_51/5mm-940nm-leds-infrared-emitter-and-ir-receiver-diode-p-519.html (sensor used in the project.

https://www.electronics-notes.com/articles/electronic_components/transistor/phototransistor-circuits-applications.php

Data Logging

https://github.com/shrocky2/GarageWeb

https://pinout.xyz/#

https://os.mbed.com/platforms/ST-Nucleo-L432KC/

Motor:

https://people.ece.cornell.edu/land/courses/ece4760/FinalProjects/s2012/xz227_gm348/xz227_gm348/900-00005-StdServo-v2.0.pdf

https://controllerstech.com/servo-motor-with-stm32/

https://www.engineersgarage.com/stm32/interfacing-servo-motor-with-stm32/