The Butter Buttler

By Christian Leycam

Description of the Product:

Family reunions can often mean large extravagant meals at the dining table. One minor inconvenience that can beset these events is the act of passing around dishes and condiments while everyone is engaged in active conversation. There are a few products on the market to address this issue such as the rotating tray, but they all lack the use of modern automation. I sought out to address this issue and took the opportunity to learn how to utilize motor drivers and infrared sensors. I designed a system to tow tableside condiments and small dishes across the dining table through signals received by an infrared receiver module. The product is intended to be a small fun gadget to use among the family and can additionally be used as a fun toy for the children.

Figure 1: The Butter Buttler automation device. The device is composed of a microcontroller, an infrared receiver module, two brushed DC motors, and a motor driver. All these components are secured within the 3D printed housing.

Electromechanical Details:

Interfacing with condiments and small dishes: Fixed arms attached to the housing act as tow bars to position items such as butter to a desired location. Commands to direct motion are given by a small remote that sends infrared signals to the receiver located at the top of the housing. The microcontroller located within the housing then delivers signals on the appropriate PWM channel to direct the internally mounted motors. The complete housing can be observed in figure 1.

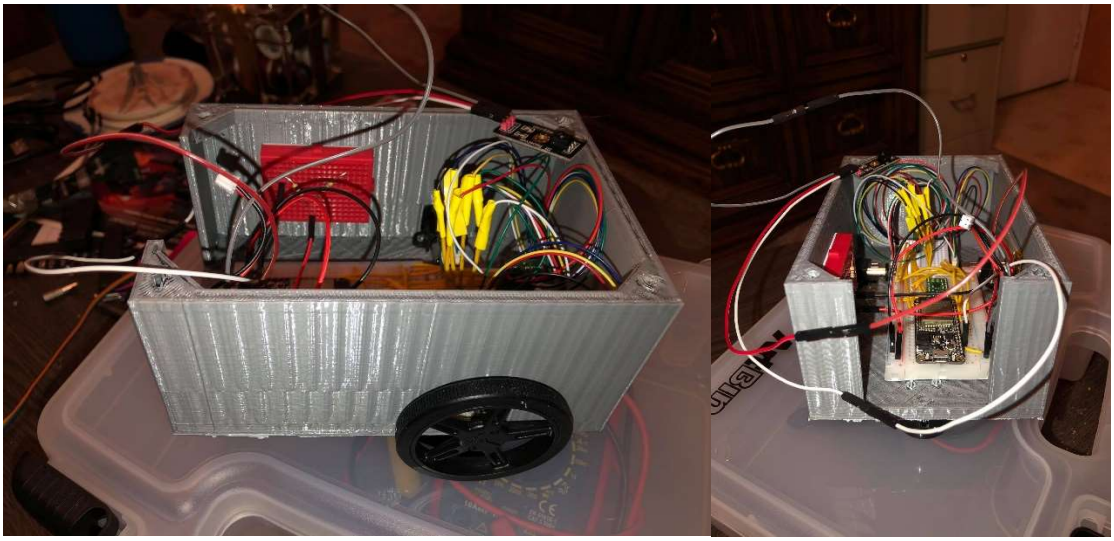

Figure 2: Main circuit located in the base of the housing (left). Exposed rear opening showing the ESP32 microcontroller, the circuit, and the DC motors in the rear (right).

Housing: The housing contains all the circuits. All housing components are secured to each other with #6-32 machine screws. The power cable and micro-USB port can be accessed through the rear side. The wheels attach to the DC motor shafts that are fitted through openings placed on the left and right sides. The motors are mounted internally within the bottom box. This can be observed in figure 2.

Circuit:

The intent of this project was to utilize a motor driver to control two brushed DC motors in addition to learning how to control this system remotely. There were three main elements: (1) The brushed DC motors, (2) The Motor Driver and (3) The IR receiver.

1. Brushed DC Motors: I had the brushed DC motors from my ME 100 lab kit. Both motors were fitted with quadrature encoders which provided 12 counts per revolution. Motor mounts were also provided by the ME100 lab kit.

2. DRV8833 Motor Driver: The motor driver utilized was provided in my ME102B lab kit. This driver allowed multidirectional control of the two DC motors. This capability enabled the system to perform left and right rotations necessary for the application.
3. OSEPP IR Receiver Module: To read infrared signals, I purchased an existing IR receiver module with an IR remote ($12.99) designed to be used in Arduino projects. I connected the module directly with the 5V power supply and took analog readings through the ESP32.
4. 6 Count AA Battery pack: The battery pack was provided in the ME100 lab kit. The battery pack was used to provide a portable power supply to the circuit.

A full circuit diagram is provided below. Careful considerations were taken to ensure that neither component was subjected to harmful voltages or current. Encoder count values were sent to the microcontroller to calculate radial velocity. The calculated speed was then used within a proportional-integral control scheme to account for the various loads that the device would have to tow.

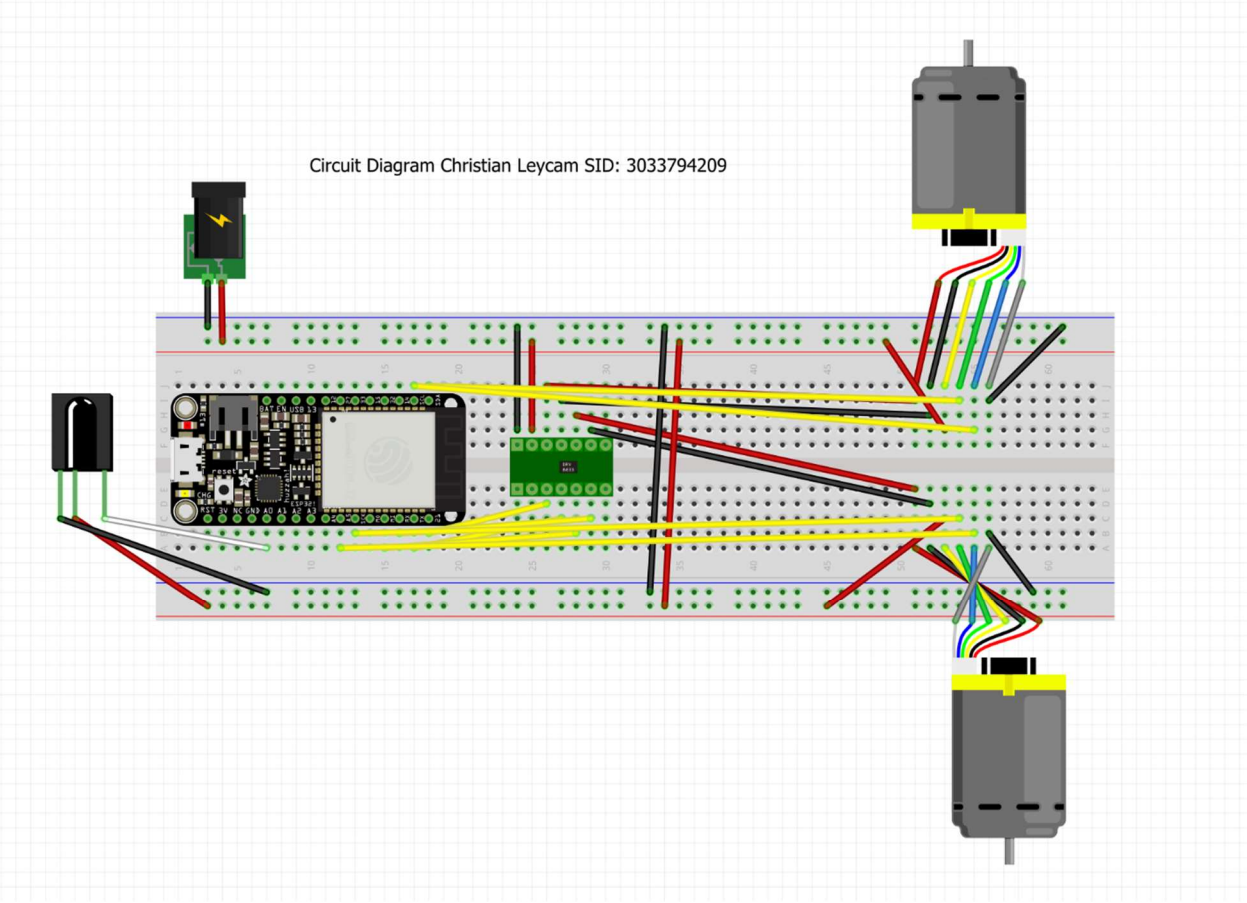

Figure 3: The Butter Buttler Circuit Diagram.

Finite State Diagram:

The states of motion can cause a numerous combination of state transitions as observed in the finite state diagram below. Corresponding signals from specific buttons off an IR remote were set as triggering mechanisms to begin specific motions. The logic was implemented in the Arduino sketch, which can be observed in Appendix I.

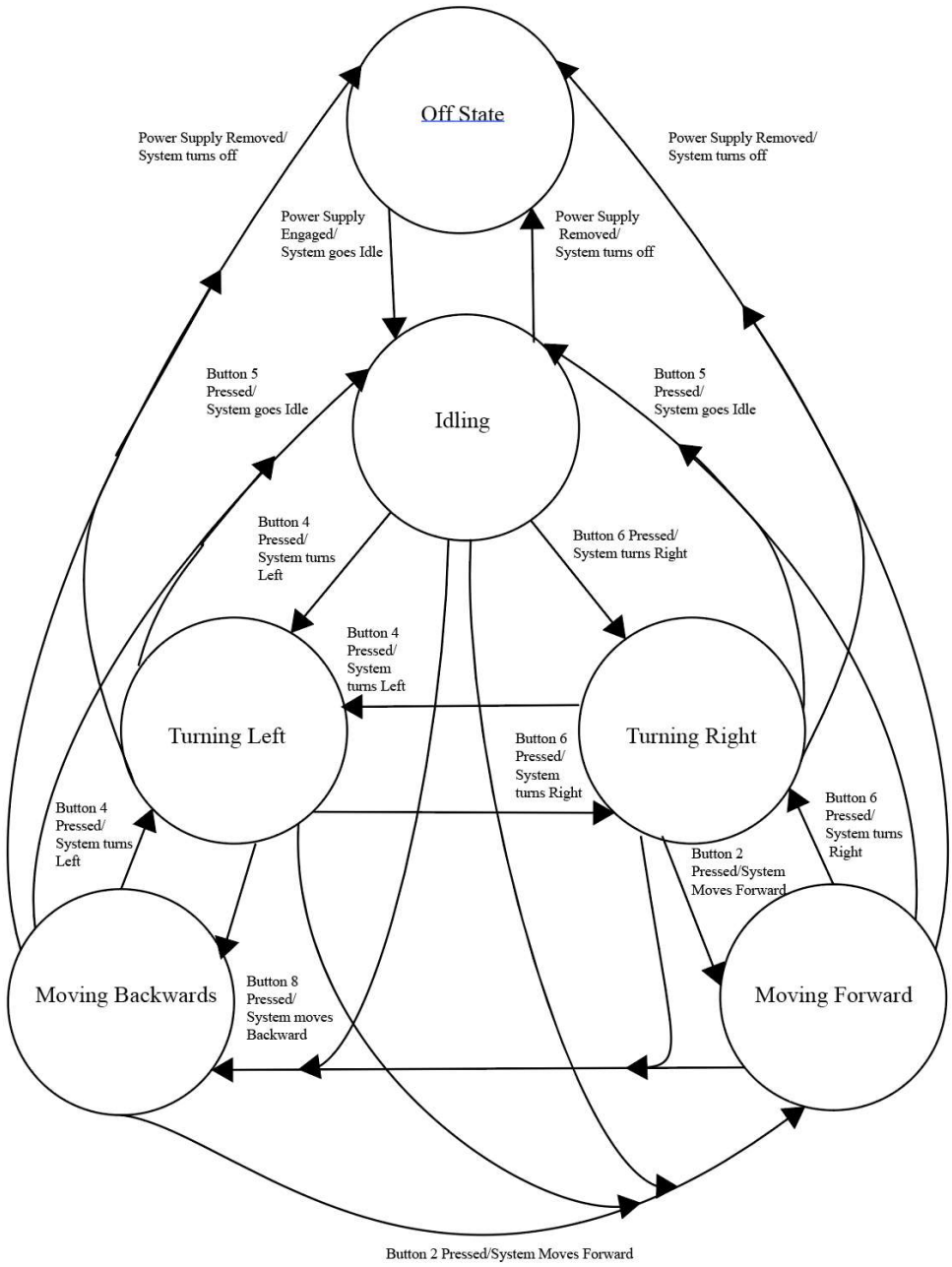

Figure 4: Finite State Diagram

APPENDIX I: Arduino Code

```
// Christian Leycam SID: 3033794209 Final Project
#include <ESP32Encoder.h>
#include <IRremote.h>

ESP32Encoder encoder;
ESP32Encoder encoder2;


// establish PWM variables
const int freq = 5000; // PWM resolution
const int Mot1Channel1 = 0; // 1st PWM channel for motor 1
const int Mot1Channel2 = 1; // 2nd PWM channel for motor 1
const int Mot2Channel1 = 2; // 1st PWM channel for motor 2
const int Mot2Channel2 = 3; // 2nd PWM channel for motor 2
// const int ServChannel1 = 4; // PWM channel for servo 1
// const int ServChannel2 = 5; // PWM channel for servo 2
const int resolution = 8; // 8 bit resolution
int Mot1Pin1 = 21;  // PWM pin for motor 1 control
int Mot1Pin2 = 17;  // PWM pin for motor 1 control
int Mot2Pin1 = 16;  // PWM pin for motor 2 control
int Mot2Pin2 = 19;  // PWM pin for motor 2 control

// establish servo variables
int ServPin1 = 13;  // PWM pin for motor 2 control
int ServPin2 = 39;  // PWM pin for motor 2 control

// Variables to control and/or print
volatile int spd1 = 0; // Speed in RPM for motor 1
volatile int spd2 = 0; // Speed in RPM for motor 1
volatile int desspd = 75; // Desired Speed in RPM
volatile int duty1 = 0; // duty cycle value for motor 1
volatile int duty2 = 0; // duty cycle value for motor 2
int tstep = 0; // t(k) (make it every 1 second)
int Iterm1;
int Iterm2;

// IR reciever variables
const int RECV_PIN = A0;
IRrecv irrecv(RECV_PIN);
decode_results results;
unsigned long key_value = 0;

// Timer/count variables
hw_timer_t * blinktimespd = NULL; // needed for timer
hw_timer_t * blinktime = NULL; // needed for timer
hw_timer_t * CTLtime = NULL; // needed for control calculations timer
volatile int tic1 = 0; // variable that stores the encoder count
volatile int tic2 = 0;
#define Debounce 200
volatile int buttontimer = 0;
volatile int buttontimer_prev = 0;
unsigned long ctlprev;



void blink() { // time ISR to set up blinking
 tstep = tstep + 1; // t(k) counter
```

```
  Serial.print(tstep); // Time step
  Serial.print('\t');
  Serial.print(spd1);
  Serial.print('\t');
  Serial.print(spd2);
  Serial.print('\t');
  Serial.print(desspd);
  Serial.print('\n');
  }

 void spdcalc() { // time ISR to set up blinking
  tic1 = encoder.getCount();
  tic2 = encoder2.getCount();
  spd1 = 0.006911*tic1*12000/(2*PI);
  spd2 = -0.006911*tic2*12000/(2*PI);
  encoder.clearCount();
  encoder2.clearCount();
  }

  void CTL() { // set up controller calculations
   Iterm1 += 0.012*(desspd-spd1);
   if(Iterm1 > 40)
   {
     Iterm1 = 40;
   }
   else if(Iterm1 < 25)
   {
      Iterm1 = 25;
   }

   duty1 = 1.2*((desspd-spd1)+Iterm1);
   if(duty1 > 150)
   {
     duty1 = 150;
   }
   else if( duty1 < 65)
   {
      duty1 = 65;
   }

   Iterm2 += 0.012*(desspd-spd2);
   if(Iterm2 > 40)
   {
     Iterm2 = 40;
   }
   else if(Iterm2 < 25)
   {
      Iterm2 = 25;
   }

   duty2 = 1.2*((desspd-spd2)+Iterm2);
   if(duty2 > 150)
   {
     duty2 = 150;
   }
   else if( duty1 < 65)
   {
      duty2 = 65;
   }

  }


 void setup() {
   Serial.begin(115200); //start serial connection
```

```
123     ESP32Encoder::useInternalWeakPullResistors=UP;
124     encoder.attachFullQuad(23, 22); // encoder for motor 1
125     encoder2.attachFullQuad(18, 5); // encoder for motor 2
126     // put your setup code here, to run once:
127     ledcSetup(Mot1Channel1, freq, resolution); // set freq and res
128     ledcSetup(Mot1Channel2, freq, resolution); // set freq and res
129     ledcSetup(Mot2Channel1, freq, resolution); // set freq and res
130     ledcSetup(Mot2Channel2, freq, resolution); // set freq and res
131     // ledcSetup(ServChannel1, freq, resolution); // set freq and res
132     // ledcSetup(ServChannel2, freq, resolution); // set freq and res
133     ledcAttachPin(Mot1Pin1, Mot1Channel1); // sets up PWM pin
134     ledcAttachPin(Mot1Pin2, Mot1Channel2); // sets up PWM pin
135     ledcAttachPin(Mot2Pin1, Mot2Channel1); // sets up PWM pin
136     ledcAttachPin(Mot2Pin2, Mot2Channel2); // sets up PWM pin
137     // ledcAttachPin(ServPin1, ServChannel1); // sets up PWM pin
138     // ledcAttachPin(ServPin2, ServChannel2); // sets up PWM pin
139
140     // Establish servo parameters
141     // Allow allocation of all timers
142
143     blinktimespd = timerBegin(1, 80, true); // set prescaler
144     timerAttachInterrupt(blinktimespd, &spdcalc, true);
145     timerAlarmWrite(blinktimespd, 5000, true); //check every 5 milliseconds
146     timerAlarmEnable(blinktimespd);// enable timer
147
148     blinktime = timerBegin(1, 80, true); // set prescaler
149     timerAttachInterrupt(blinktime, blink, true);
150     timerAlarmWrite(blinktime, 1000000, true); //check every second
151     timerAlarmEnable(blinktime);// enable timer
152
153
154     CTLtime = timerBegin(2, 80, true); // set prescaler
155     timerAttachInterrupt(CTLtime, &CTL, true);
156     timerAlarmWrite(CTLtime, 5000, true); //check every 5 milliseconds
157     timerAlarmEnable(CTLtime);// enable timer
158
159     // setup IR
160     irrecv.enableIRIn();
161     irrecv.blink13(true);
162
163
164   }
165
166   void loop() {
167     // put your main code here, to run repeatedly:
168
169       if (irrecv.decode(&results)){
170
171           if (results.value == 0XFFFFFFFF)
172             results.value = key_value;
173
174           switch(results.value){
175             case 0xFFE01F: // move arms down
176             Serial.println("-");
177             break ;
178             case 0xFFA857: // move arms up
179             Serial.println("+");
180             break ;
181             case 0xFF18E7: // move forward
182             Serial.println("2");
183             servicefor();
184             break ;
185             case 0xFF10EF: // move left
186             Serial.println("4");
187             serviceleft();
```

```
188             break ;
189             case 0xFF38C7: // stop DC motors
190             Serial.println("5");
191             servicestop();
192             break ;
193             case 0xFF5AA5: // move right
194             Serial.println("6");
195             serviceright();
196             break ;
197             case 0xFF4AB5: // move backwards
198             Serial.println("8");
199             serviceback();
200             break ;
201           }
202           key_value = results.value;
203           irrecv.resume();
204     }
205   }
206
207   void servicestop(){
208             ledcWrite(Mot1Channel2,0);
209             ledcWrite(Mot1Channel1,0);
210             ledcWrite(Mot2Channel1,0);
211             ledcWrite(Mot2Channel2,0);
212     }
213
214   void servicefor(){
215             if (spd1<desspd)
216             {
217               ledcWrite(Mot1Channel2,duty1);
218               ledcWrite(Mot1Channel1,0);
219             }
220             else if (spd1>desspd)
221             {
222               duty1 = -1*duty1;
223               ledcWrite(Mot1Channel2,0);
224               ledcWrite(Mot1Channel1,duty1);
225              }
226             if (spd2<desspd)
227             {
228               ledcWrite(Mot2Channel1,duty2);
229               ledcWrite(Mot2Channel2,0);
230              }
231              else if (spd2>desspd)
232               {
233               duty2 = -1*duty2;
234               ledcWrite(Mot2Channel1,0);
235               ledcWrite(Mot2Channel2,duty2);
236               }
237               }
238
239   void serviceback(){
240             if (spd1<desspd)
241             {
242               ledcWrite(Mot1Channel2,0);
243               ledcWrite(Mot1Channel1,duty1);
244             }
245             else if (spd1>desspd)
246             {
247               duty1 = -1*duty1;
248               ledcWrite(Mot1Channel2,duty1);
249               ledcWrite(Mot1Channel1,0);
250              }
251             if (spd2<desspd)
252             {
```

```
253              ledcWrite(Mot2Channel1,0);
254              ledcWrite(Mot2Channel2,duty2);
255             }
256             else if (spd2>desspd)
257              {
258              duty2 = -1*duty2;
259              ledcWrite(Mot2Channel1,duty2);
260              ledcWrite(Mot2Channel2,0);
261              }
262              }
263
264  void serviceleft(){
265            if (spd1<desspd)
266            {
267              ledcWrite(Mot1Channel2,0);
268              ledcWrite(Mot1Channel1,duty1);
269            }
270            else if (spd1>desspd)
271            {
272              duty1 = -1*duty1;
273              ledcWrite(Mot1Channel2,duty1);
274              ledcWrite(Mot1Channel1,0);
275             }
276            if (spd2<desspd)
277            {
278              ledcWrite(Mot2Channel1,duty2);
279              ledcWrite(Mot2Channel2,0);
280             }
281             else if (spd2>desspd)
282              {
283              duty2 = -1*duty2;
284              ledcWrite(Mot2Channel1,0);
285              ledcWrite(Mot2Channel2,duty2);
286              }
287              }
288
289  void serviceright(){
290            if (spd1<desspd)
291            {
292              ledcWrite(Mot1Channel2,duty1);
293              ledcWrite(Mot1Channel1,0);
294            }
295            else if (spd1>desspd)
296            {
297              duty1 = -1*duty1;
298              ledcWrite(Mot1Channel2,0);
299              ledcWrite(Mot1Channel1,duty1);
300             }
301            if (spd2<desspd)
302            {
303              ledcWrite(Mot2Channel1,0);
304              ledcWrite(Mot2Channel2,duty2);
305             }
306             else if (spd2>desspd)
307              {
308              duty2 = -1*duty2;
309              ledcWrite(Mot2Channel1,duty2);
310              ledcWrite(Mot2Channel2,0);
311              }
312              }
```