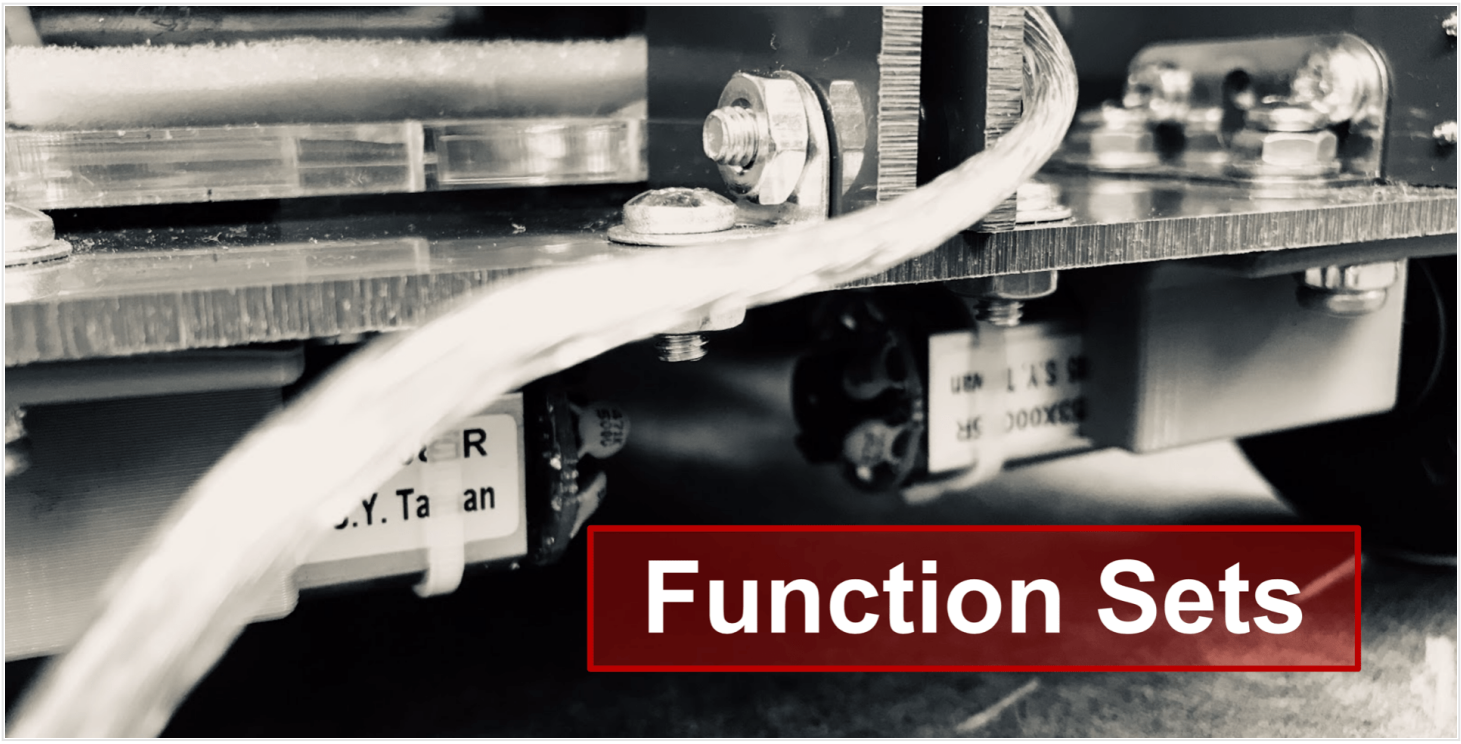
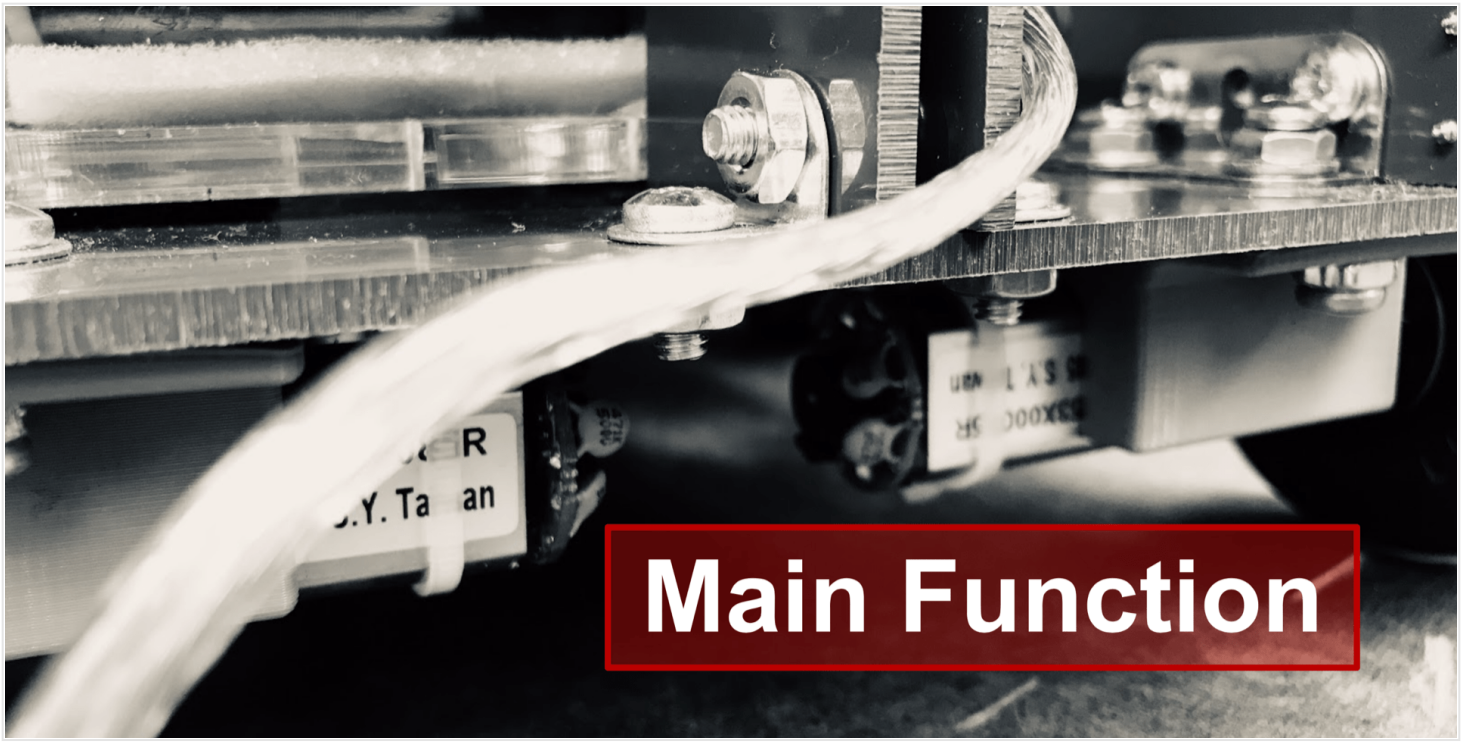


The team focused on modular code so that functions that may be reused, such as line following, driving, and rotation, were separate functions that must be called. This made the main function to be easier to read. Not only was the main function significantly shorter, but it focused on overall robot movement instead of the individual lines of code.

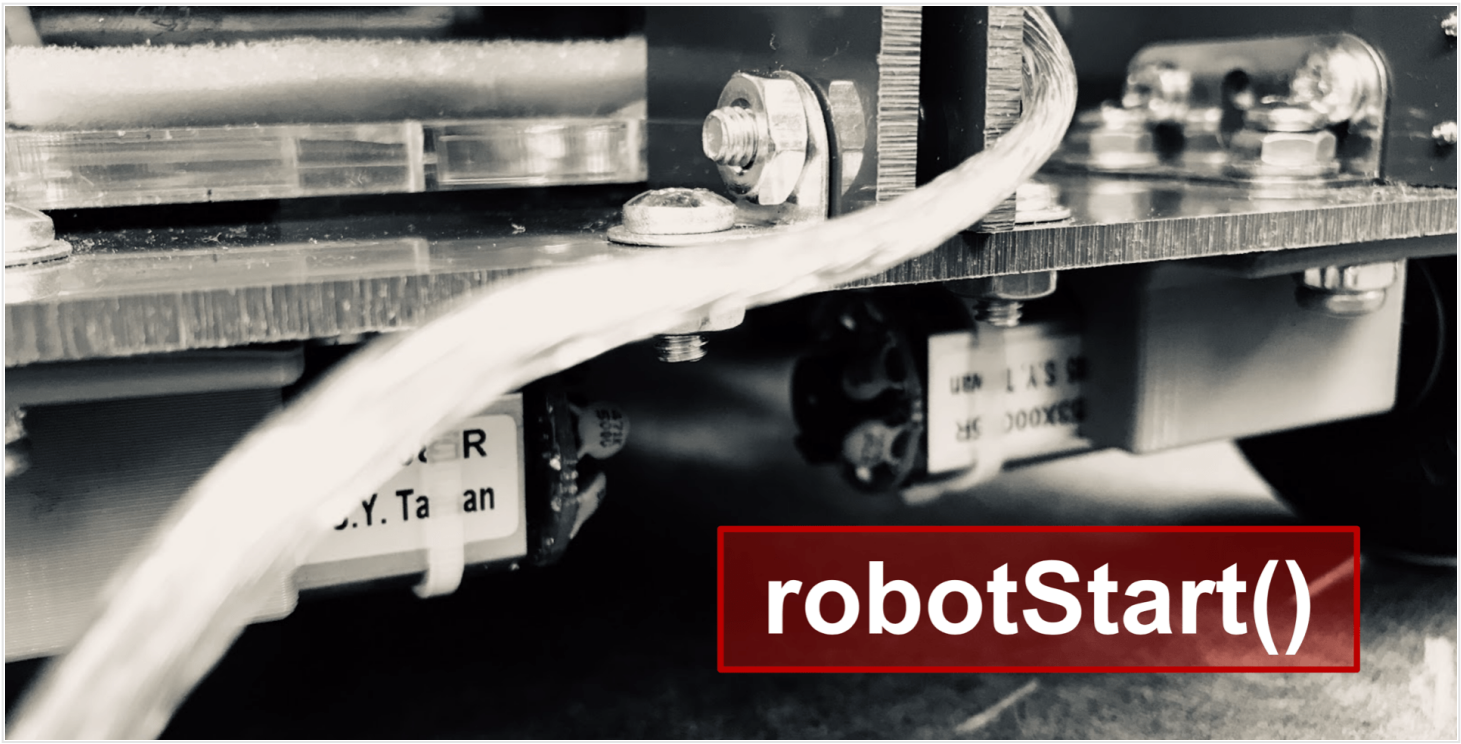
For example, the code in the main function for moving the robot to the trash consisted of a list of functions with each line directing a different movement. It called a function to drive forward for a given number of counts, a function to rotate counterclockwise for a given number of counts, a function to drive forward until the microswitches hit the wall, a function to rotate for a given number of counts, and a function to drive backward until the microswitches hit the trashcan. This enabled function inputs such as motor percentage and the number of shaft encoder rotations to be easily updated while each line showed one full movement, making the code for the robot's path easy to follow.

The specific structure of the main function is described here. These reference additional functions. Most notably, the maneuverability function set that contains the driving forward/backward and rotation functions.



The following functions are the finalized functions in the main function. Please click on each of their names for an explanation of their logic, a video clip of the executed code, and the finished code. Note: many of these functions call functions from the maneuverability set. More information about those can be found here.

void robotStart();
void trayDump();
void jukeboxComplete();
void driveUpRamp();
void ticketSlideFinal();
void hotPlateFinal();



Overall Function:

This function declares the screen colors, initializes the robot, and sets the robot to start when a light turns on after the screen is pressed.

```
01 void robotStart() {
02     LCD.Clear( FEHLCD::Black );
03     LCD.SetFontColor( FEHLCD::White );
04     RPS.InitializeTouchMenu();
05 
06     initializeRobot();
07     /*
08      * Start with light
09      */
10     touchScreen();
11     lightStart();
12 }
```

Functions used:

initializeRobot()

Sets screen to black with white text. Sets minimum and maximum values for each motor and sets them to an initial degree. Finally, it updates the encoder counts.

```
01 void initializeRobot() {
02 
03     LCD.Clear(FEHLCD::Black);
04     LCD.SetFontColor(FEHLCD::White);
05 
```

```
    burgerFlipServo.SetMin(BURGER_SERVO_MIN);
    burgerFlipServo.SetMax(BURGER_SERVO_MAX);
    burgerFlipServo.SetDegree(BURGER_SLIDE_DEGREE);

    iceCreamServo.SetMin(ICECREAM_SERVO_MIN);
    iceCreamServo.SetMax(ICECREAM_SERVO_MAX);
    iceCreamServo.SetDegree(145);

    ticketSlideServo.SetMin(TICKET_SERVO_MIN);
    ticketSlideServo.SetMax(TICKET_SERVO_MAX);
    ticketSlideServo.SetDegree(0);
    updateEncoder();
}
```

touchScreen()

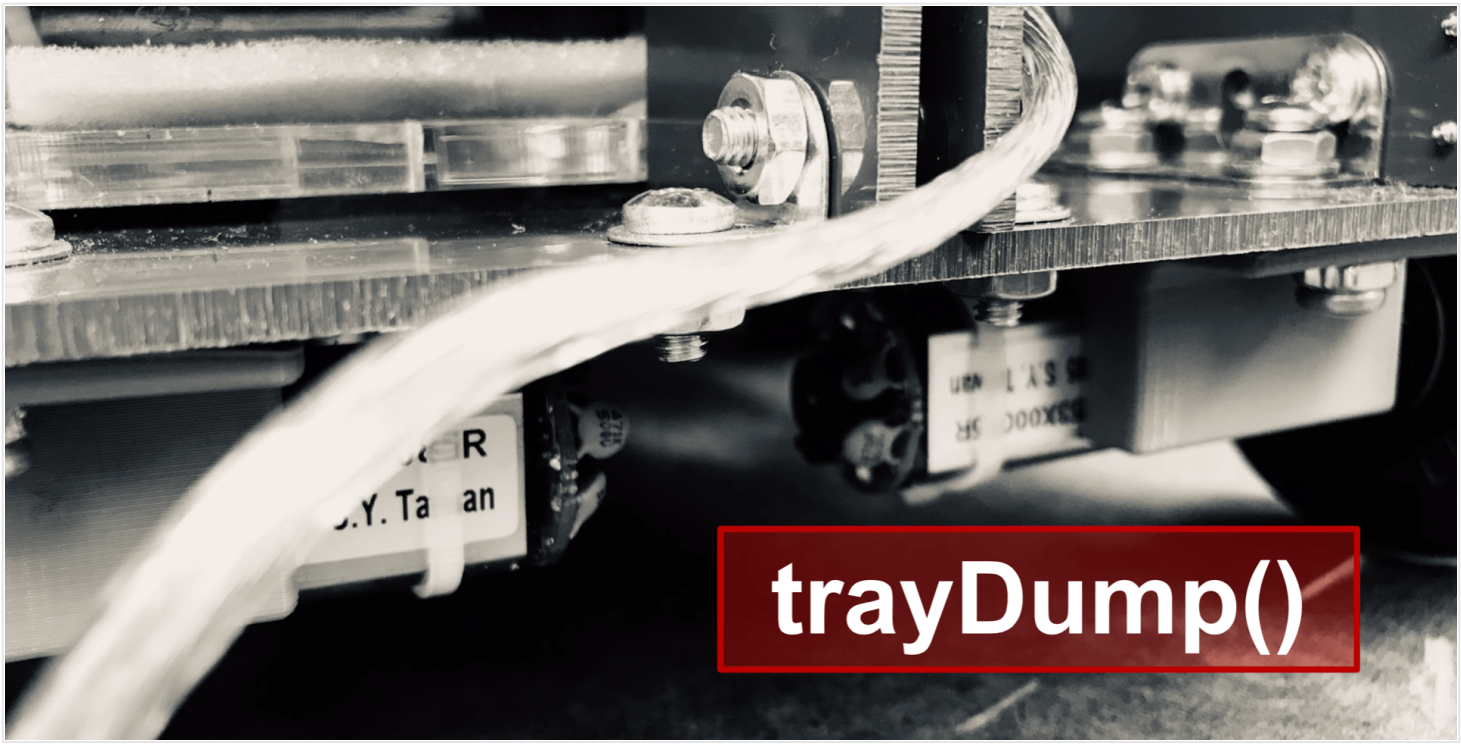
Waits for user to touch screen, then clears screen.

```
void touchScreen() {
    float x, y;
    LCD.WriteLine("Next test:");
    LCD.ClearBuffer(); //reset values
    while(!LCD.Touch(&x,&y)); //Wait for screen to be pressed
    while(LCD.Touch(&x,&y)); //Wait for screen to be unpressed
    LCD.Clear(); //clear screent
}
```

lightStart()

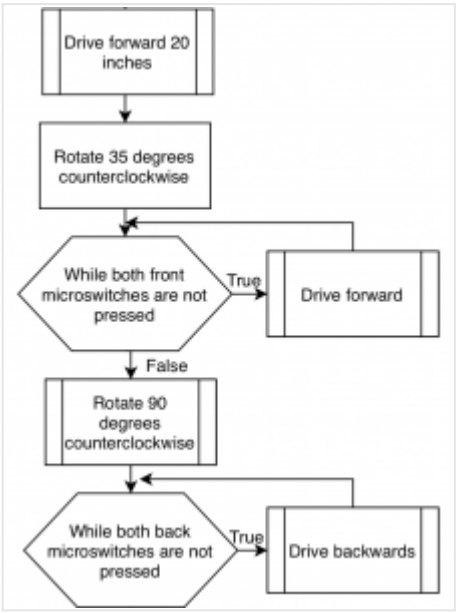
Robot sleeps while light from CdS cell is not detected.

```
void lightStart() {
    while (cdsCell.Value() > CdS_BLUE_MAX) {
        Sleep(.1);
    }
}
```



Overall Function:

This function maneuvers the robot from starting position to dumping the tray off. The logic is as follows:



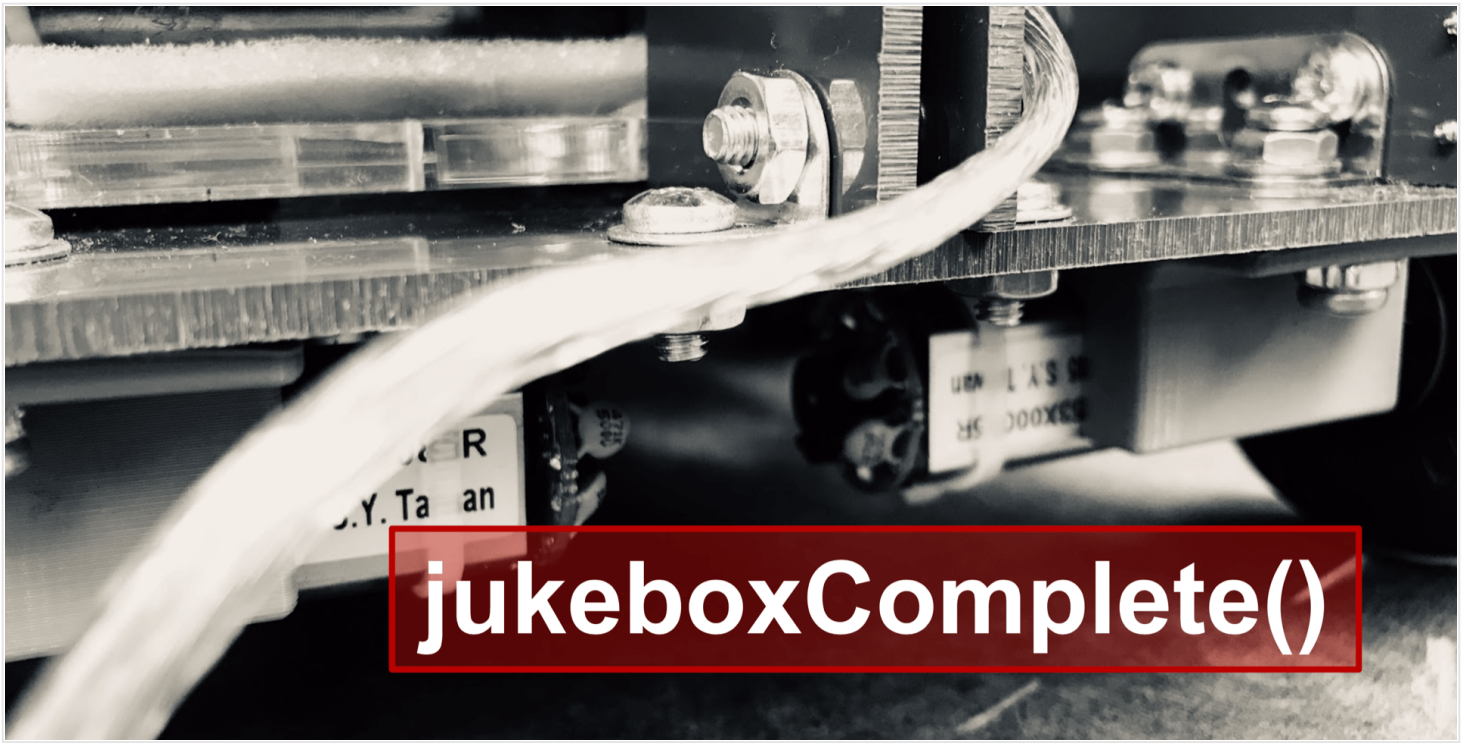
Code:

```
01 void trayDump() {
02     /*
03      * Drive forward, turn, then drive forward until
   front bump switches pressed
04      */
05     driveShaft(25, 300);
```

```
06      counterClockWiseTurnShaft(25, 50);
07      driveForwardStopFrontBumpBoth();
08
09      /*
10       * 90 degree turn counterclockwise - using shaft
    encoders
11       */
12      leftMotor.Stop();
13      rightMotor.Stop();
14      Sleep(1.0);
15      driveShaft(-25, 25);
16       counterClockWiseTurnShaft(25, 145);
17
18       /*
19        * Drive backward until back bump switches pressed.
20        */
21       leftMotor.Stop();
22       rightMotor.Stop();
23       Sleep(1.0);
24       driveBackwardStopBackBumpOne(-75);
25  }
```

Functions used:

driveShaft(int percentage, int encoderCounts);
counterClockWiseTurnShaft(int percentage, intEncoderCounts);
driveForwardStopFrontBumpBoth()
driveBackwardStopBackBumpOne(int percentage);



Maneuvering to the jukebox is broken down into three sections: line following to the colored light, reading the color of the light, and pressing the correct button. These are described in more detail below.

Line Following to Colored Light

These three functions executes the line following algorithm for three optosensors while at least one of the optosensors is on the line. It does so by calling the updateEncoder function which updates the analogGlobal global array with the condition of each of the optosensors.

```
01 void findLine() {
02     driveShaft(25, 10);
03 }
04
05 void updateEncoder() {
06     if (left_input.Value() > LEFT_LINE_READER_MID_VAL) {
07         analogGlobal[0] = 1;
08     }
09     else {
10         analogGlobal[0] = 0;
11     }
12
13     if (center_input.Value() >
   CENTER_LINE_READER_MID_VAL) {
14         analogGlobal[1] = 1;
15     } else {
16         analogGlobal[1] = 0;
17     }
18
19     if (right_input.Value() > RIGHT_LINE_READER_MID_VAL)
   {
20         analogGlobal[2] = 1;
```

```
21      } else {
22          analogGlobal[2] = 0;
23      }
24      LCD.Write(analogGlobal[0]);
25      LCD.Write(analogGlobal[1]);
26      LCD.WriteLine(analogGlobal[2]);
27      printValues();
28  }
29
30
31  void followLineToJukeBox() {
32
33      /*
34      leftMotor.SetPercent(25);
35      Sleep(.5);
36      rightMotor.SetPercent(25);
37      leftMotor.SetPercent(0);
38
39      while (analogGlobal[0] + analogGlobal[1] +
    analogGlobal[2] < 2) {
40          updateEncoder();
41          Sleep(.02);
42      }
43
44      leftMotor.SetPercent(0);
45      rightMotor.SetPercent(0);
46
47      leftMotor.SetPercent(25);
48      rightMotor.SetPercent(25);
49      Sleep(.2);
50      leftMotor.SetPercent(0);
51      rightMotor.SetPercent(0);
52      Sleep(.1);
53      */
54      while (analogGlobal[0] != 0 || analogGlobal[1] != 0
    || analogGlobal[2] != 0) {
55          printValues();
56          updateEncoder();
57          if (analogGlobal[0] == 0 && analogGlobal[1] == 0
    && analogGlobal[2] == 1) {
58              turnRight(2);
59          } else if (analogGlobal[0] == 0 &&
    analogGlobal[1] == 1 && analogGlobal[2] == 0) {
60              driveStraight(10);
61          } else if (analogGlobal[0] == 1 &&
    analogGlobal[1] == 0 && analogGlobal[2] == 0) {
62              turnLeft(2);
63          } else if (analogGlobal[0] == 1 &&
    analogGlobal[1] == 1 && analogGlobal[2] == 0) {
64              turnLeft(1);
65          } else if (analogGlobal[0] == 0 &&
    analogGlobal[1] == 1 && analogGlobal[2] == 1) {
66              turnRight(1);
67          } else if (analogGlobal[0] == 1 &&
    analogGlobal[1] == 0 && analogGlobal[2] == 1) {
68              driveStraight(10);
69          } else if (analogGlobal[0] == 1 &&
```

```
    analogGlobal[1] == 1 && analogGlobal[2] == 1) {
70              driveStraight(10);
71          }
72      }
73  }
```

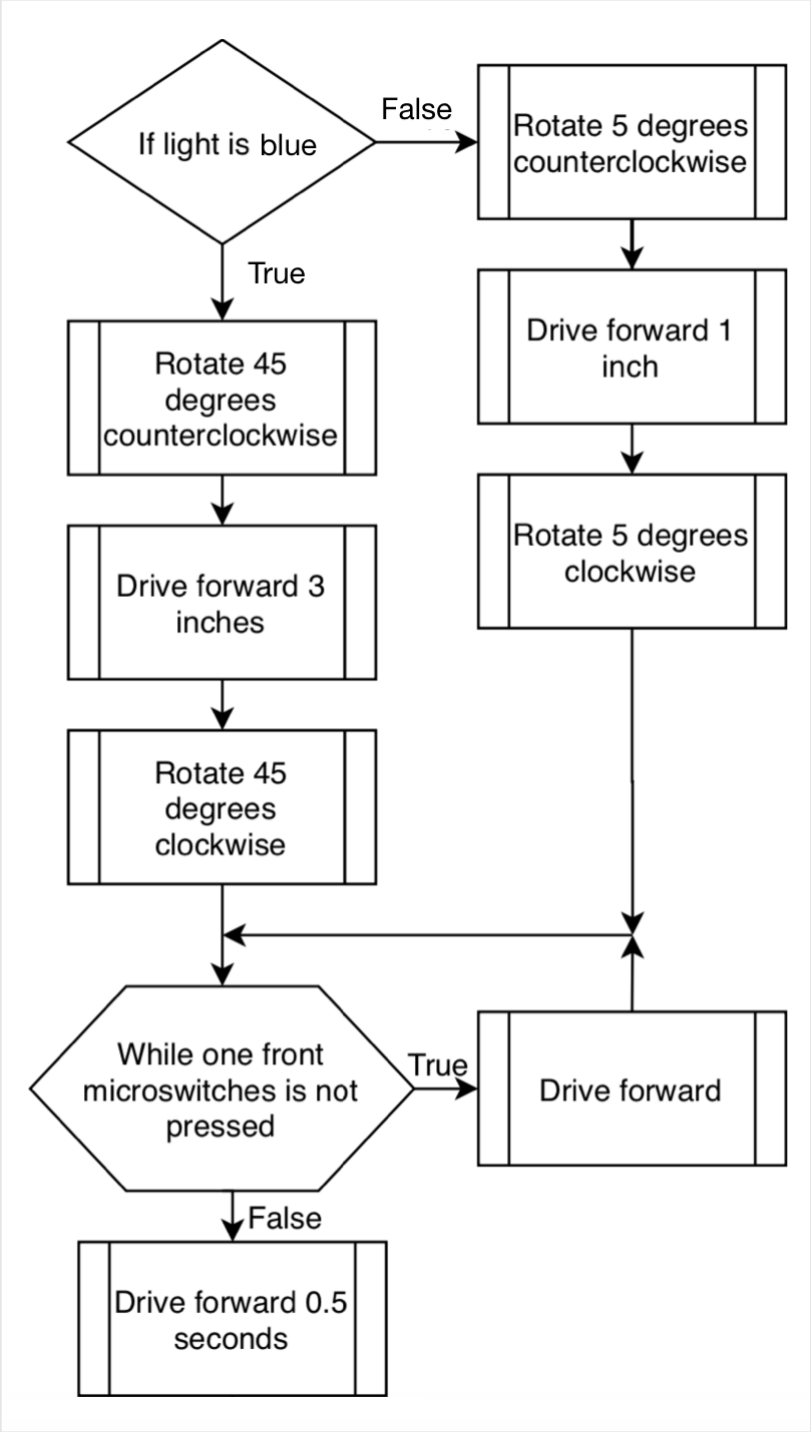
Reading the Button Color:

This function has the robot drive forward until a light is detected. The robot then stops, sleeps (to ensure robot is still), and tests for the light color. If the CdS cell value is less then the minimum value for blue, then it is a red light. If not, then the light is blue. This system guarantees that the robot will always think it is either a red light or blue light, even if no light is detected, so the robot will always be able to hit one of the buttons, even if it is not the correct one.

```
01  void jukeboxLightReading() {
02
03      while (cdsCell.Value() > CdS_BLUE_MAX && fr.Value()
    == 1 && fl.Value() == 1) {
04          //drive forward
05          rightMotor.SetPercent(25);
06          leftMotor.SetPercent(25);
07      }
08
09      rightMotor.SetPercent(0);
10      leftMotor.SetPercent(0);
11
12      Sleep(.1);
13
14      //test for light
15      if (cdsCell.Value() < CdS_BLUE_MIN) {
16          redLight = true;
17      } else {
18          blueLight = true;   //fingers crossed it's
    Annie's favorite color
19      }
20
21      //print color of light ot screen
22      if (redLight) {
23          LCD.WriteLine("Red light.");
24      } else if (blueLight) {
25          LCD.WriteLine("Blue light.");
26      } else {
27          LCD.WriteLine("No light detected. This is
    bad.");
28      }
29  }
```

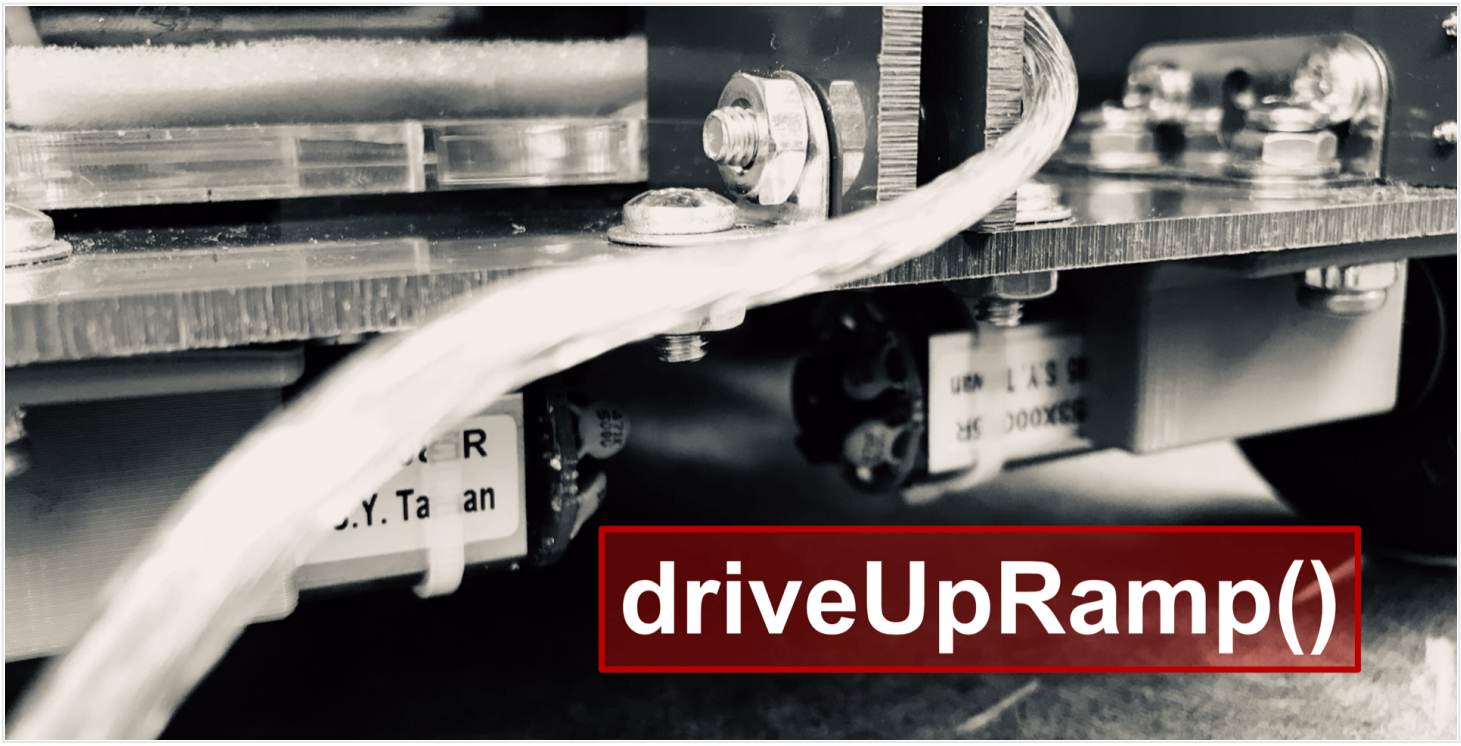
Press Jukebox Button:

This function directs the robot to hit the blue button if the blueLight variable is true. If not, then it defaults to hitting the red button. The logic behind this algorithm is described in the flowchart below.



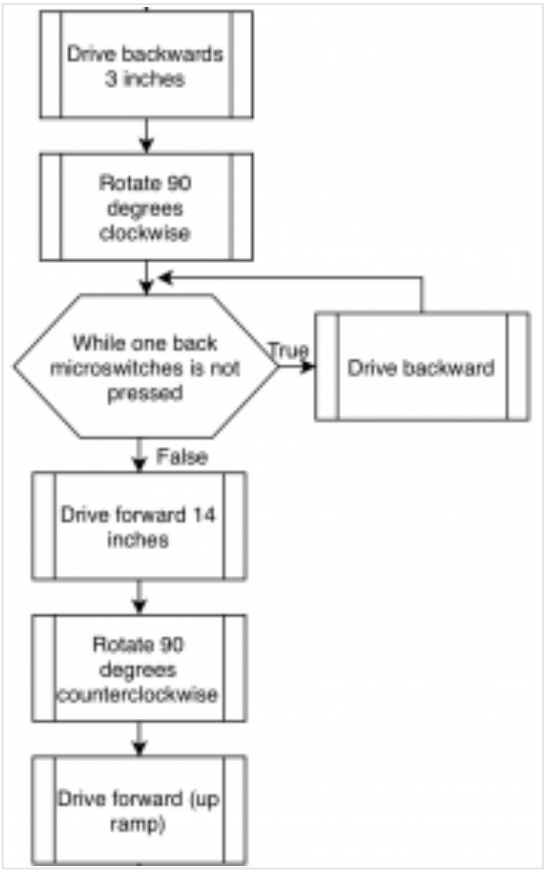

```
01 void jukeboxButton() {
02
03     if (blueLight) {
04       counterClockWiseTurnShaft(25, 50);
05       driveShaft(25, 115);
06       clockwiseTurnShaft(25, 50);
07       rightMotor.SetPercent(0);
08       leftMotor.SetPercent(0);
09       driveForwardStopFrontBumpOne();
10
11       rightMotor.SetPercent(0);
12       leftMotor.SetPercent(25);
13       Sleep(.5);
```

```
14
15        rightMotor.SetPercent(0);
16        leftMotor.SetPercent(0);
17
18      } else {
19
20          counterClockWiseTurnShaft(25, 7);
21          driveShaft(25, 10);
22          clockwiseTurnShaft(25, 7);
23          driveForwardStopFrontBumpOne();
24
25          rightMotor.SetPercent(0);
26          leftMotor.SetPercent(25);
27          Sleep(.5);
28
29          rightMotor.SetPercent(0);
30          leftMotor.SetPercent(0);
31      }
32  }
```



Overall Function:

This function maneuvers the robot to drive up the ramp from the jukebox. The logic is as follows:

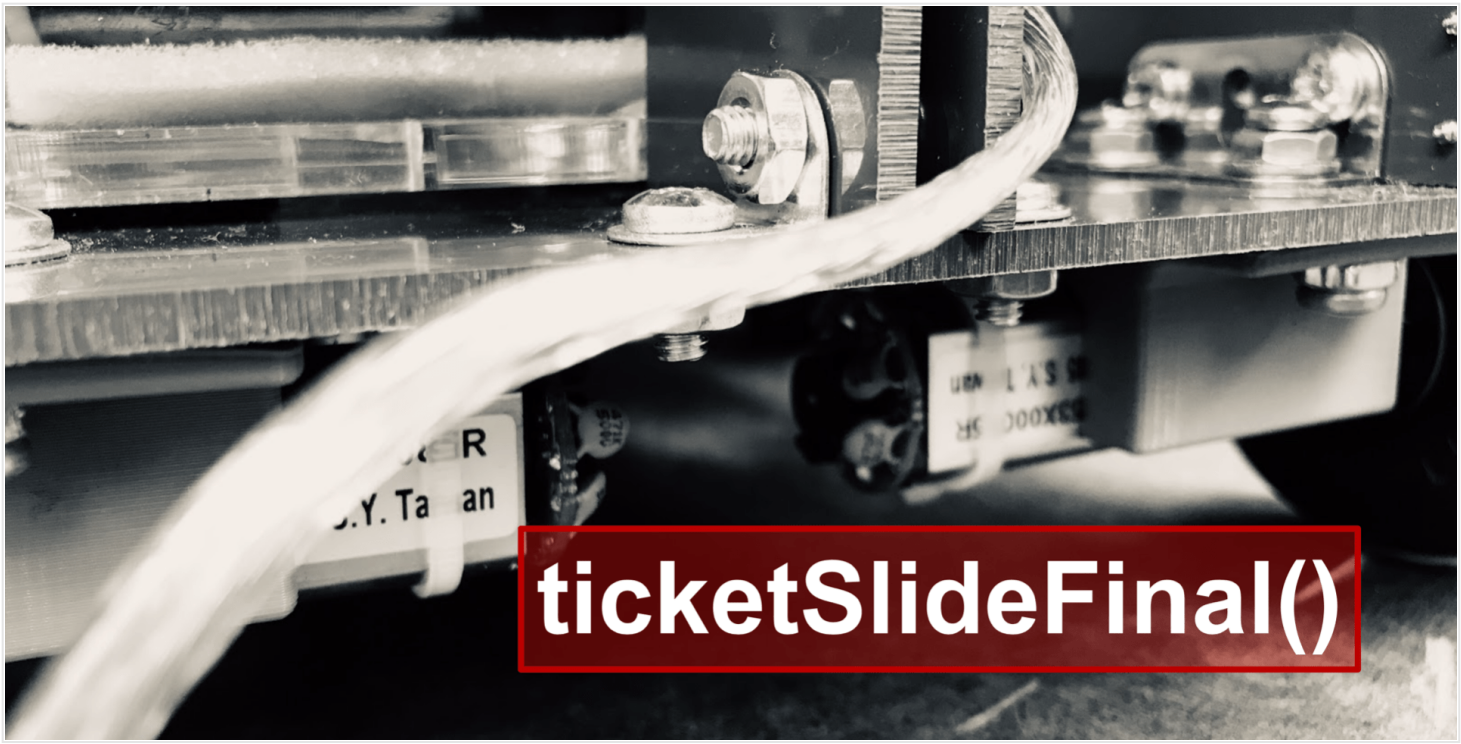


Code:

```
01 void driveUpRamp() {
02     /*
03      * Align with ramp
04      */
05     driveShaft(-25, 40);
06     counterClockWiseTurnShaft(25, 125);
07     Sleep(.2);
08     driveBackwardStopBackBumpBeforeRamp(-25);
09     Sleep(.2);
10     driveShaft(25, 340);
11     counterClockWiseTurnShaft(25, 120);
12
13
14     batteryVoltage();
15
16     /*
17      * Drive up ramp
18      */
19     driveShaft(50, 540);
20 }
```

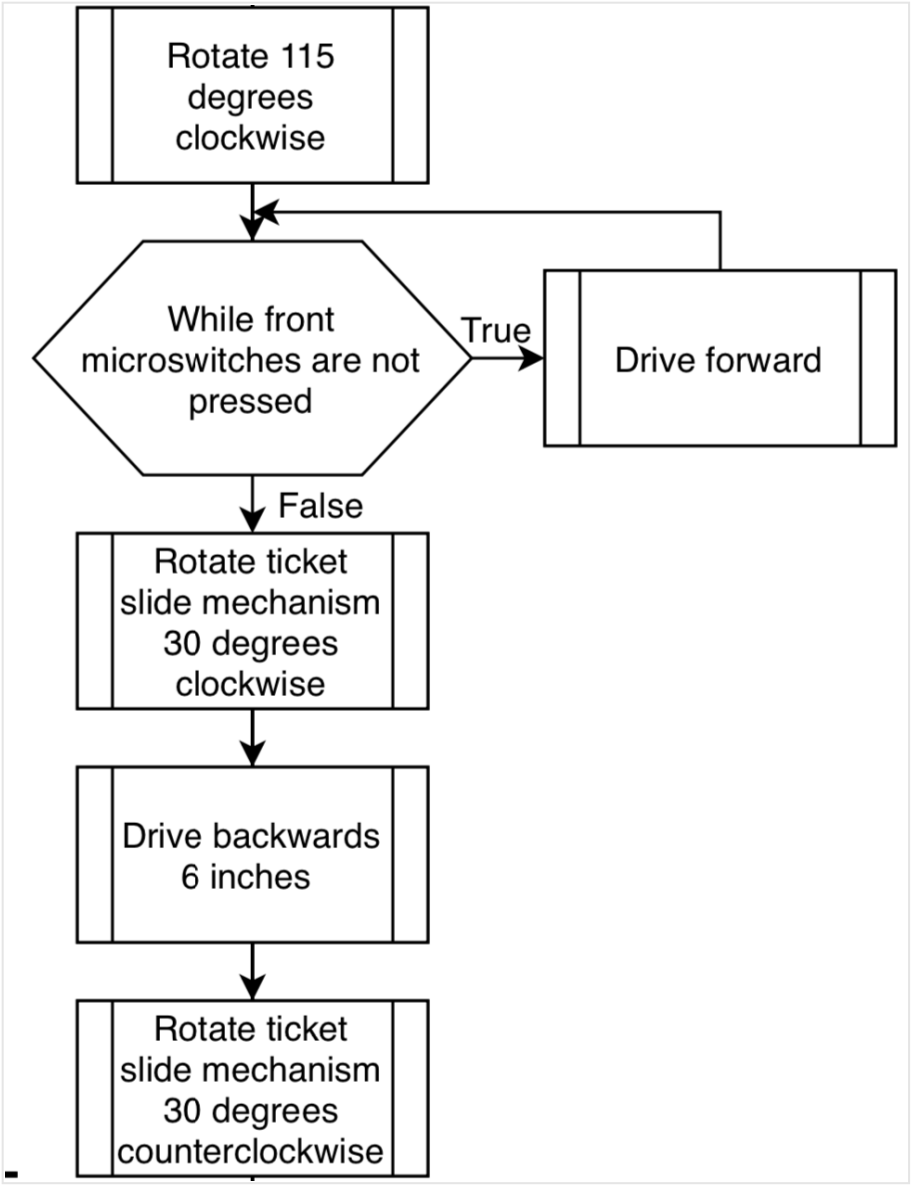
Functions used:

driveShaft(int percentage, int encoderCounts);
counterClockWiseTurnShaft(int percentage, intEncoderCounts);
driveForwardStopFrontBumpBoth()
driveBackwardStopBackBumpBeforeRamp()
driveBackwardStopBackBumpOne(int percentage);



Overall Function:

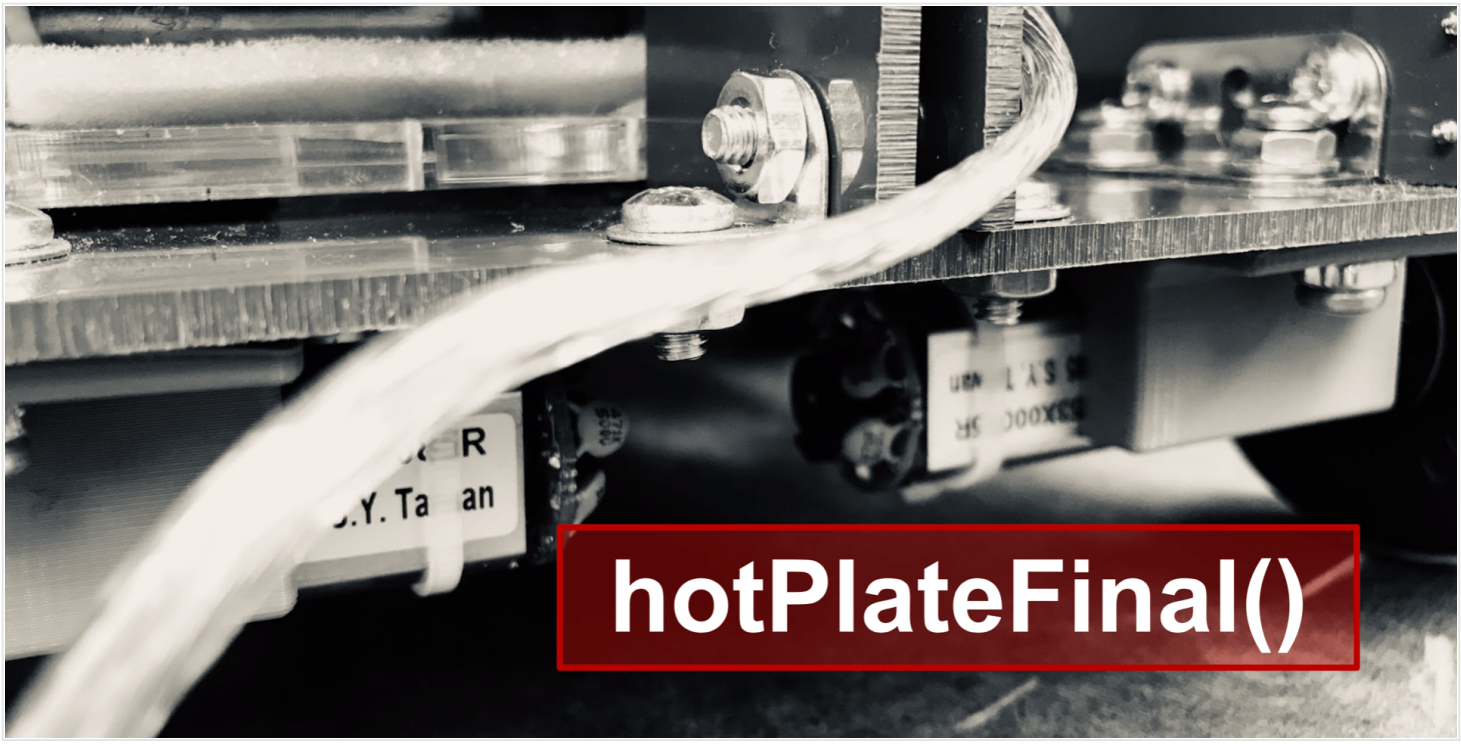
These two functions set the robot up to slide the ticket and slides the ticket. The logic is as follows:



Code:

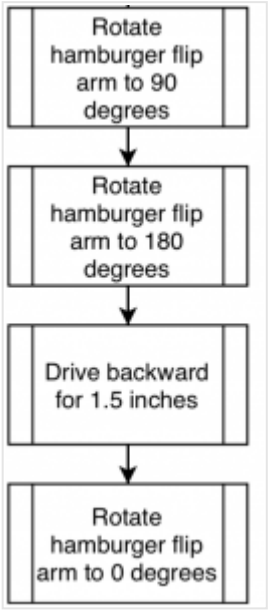
```
01 void ticketSlide() {
02     driveForwardStopFrontBumpBoth();
03 
04     ticketSlideServo.SetMin(TICKET_SERVO_MIN);
05     ticketSlideServo.SetMax(TICKET_SERVO_MAX);
06 
07     ticketSlideServo.SetDegree(TICKET_SLIDE_DEGREE);
08     Sleep(.5);
09 
10     //Reset encoder counts
11     rightShaftEncoder.ResetCounts();
12     leftShaftEncoder.ResetCounts();
13 
14     float currentTime = TimeNow();
15     while (TimeNow() - currentTime < 2) {
16         rightMotor.SetPercent(-25);
17         leftMotor.SetPercent(-20);
18     }
```

```
19
20      //Set both motors to stop, then sleep
21      rightMotor.SetPercent(0);
22      leftMotor.SetPercent(0);
23      Sleep(.1);
24
25      //release hook from ticket slide
26      driveShaft(25, 25);
27      Sleep(.1);
28      ticketSlideServo.SetDegree(-1.0*TICKET_SLIDE_DEGREE);
29      Sleep(.1);
30
31  }
32
33  void ticketSlideFinal() {
34      /*
35       * Ticket slide
36       */
37      clockwiseTurnShaftTicket(25, 120); //90 degree turn
38      driveShaft(50, 20);
39      clockwiseTurnShaftTicket(25, 35); //90 degree turn
40      ticketSlide();
41  }
```



Overall Function:

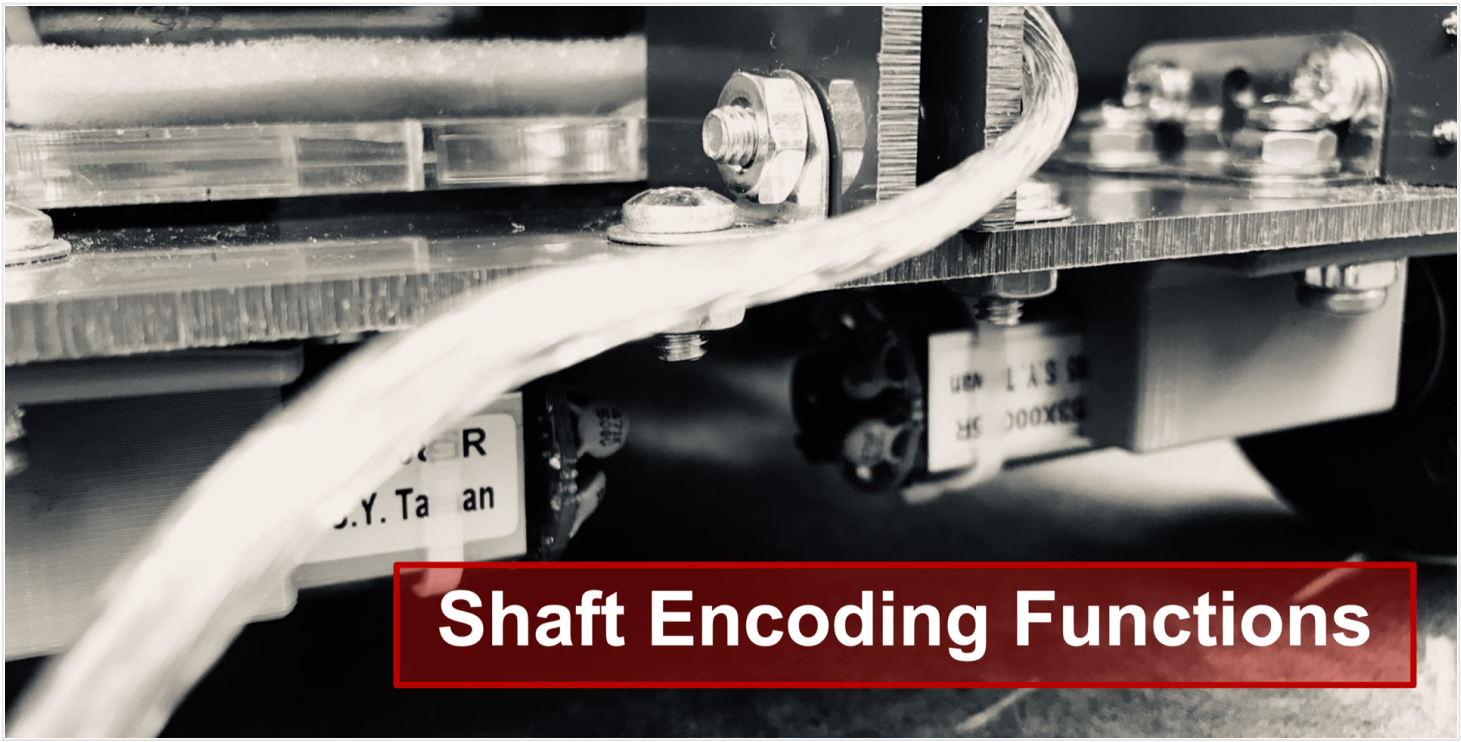
This function drives the robot from the ticket to the hotplate, flips the hotplate, and returns it to its initial position. The logic is as follows:




```
01 void hotPlateFinal() {
02     /*
03      * Align with hot plate
04      */
05     counterClockWiseTurnShaft(25, 50);
06     burgerFlipServo.SetDegree(0);
07     driveForwardStopFrontBumpOne();
```

```
08
09      /*
10       * Turn hot plate
11       */
12      burgerFlipServo.SetDegree(95);
13      Sleep (.5);
14      burgerFlipServo.SetDegree(0);
15  }
```

One of the main instances of modular coding practices was in the creation of reusable functions for each case of robot maneuverability. Sets: Shaft Encoding, Microswitch Movement, and RPS Functions.



Overall Function Set:

This function set contains three functions for robot movement based on shaft encoding. One is for driving in a straight line (driveShaft) and two are for rotating clockwise (clockwiseTurnShaft) and counterclockwise (counterClockWiseTurnShaft).

Individual Functions:

driveShaft(int percent, int counts)

Drives the robot at “percent variable” percent of its motor for a total of “count variable” encoder counts.

```
01 void driveShaft(int percent, int counts) //using
   encoders
02 {
03     //Reset encoder counts
04     rightShaftEncoder.ResetCounts();
05     leftShaftEncoder.ResetCounts();
06
07     //Set both motors to desired percent
08     rightMotor.SetPercent(percent);
09     leftMotor.SetPercent(percent);
10
11     //While the average of the left and right encoder is
   less than counts,
12     //keep running motors
13     while((leftShaftEncoder.Counts() +
   rightShaftEncoder.Counts()) / 2. < counts);
14
15     //Turn off motors
```

```
    rightMotor.Stop();
    leftMotor.Stop();
}
```

clockwiseTurnShaft()

Robot uses the percent variable input to run the right motor at negative percent (-1*percent) and the left motor at percent to rotate over an average of “variable counts” encoder counts.

```
void clockwiseTurnShaft(int percent, int counts) //
using encoders
{
    //Resent encoder counts
    rightShaftEncoder.ResetCounts();
    leftShaftEncoder.ResetCounts();

    //Set both motors to desired percent
    rightMotor.SetPercent(percent * -1.0);
    leftMotor.SetPercent(percent);

    //While average of left and right encoder is less
than counts, keep running motors
    while ((leftShaftEncoder.Counts() +
rightShaftEncoder.Counts()) / 2.0 < counts) {
        Sleep(100);
    }

    //Turn off motors
    rightMotor.Stop();
    leftMotor.Stop();
}
```

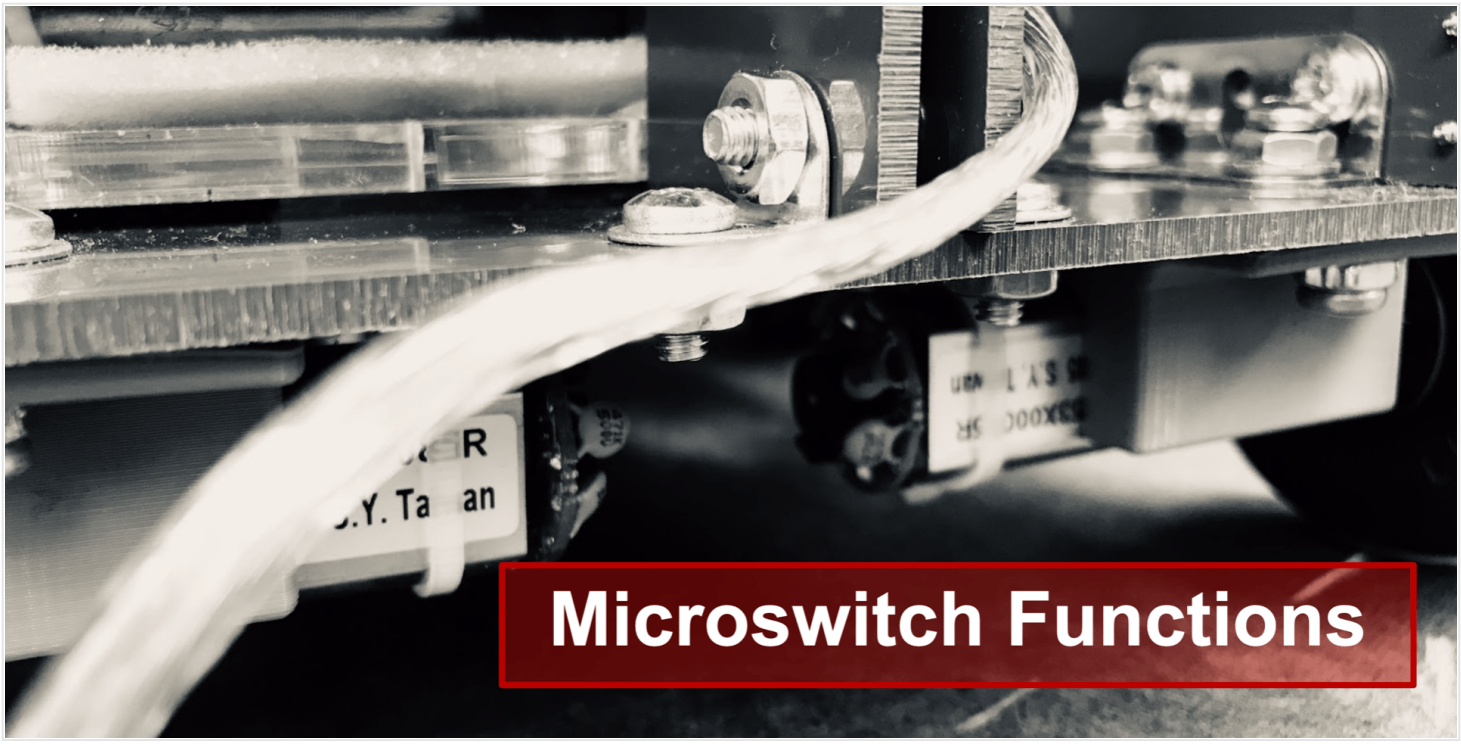
counterClockWiseTurnShaft()

Robot uses the percent variable input to run the left motor at negative percent (-1*percent) and the right motor at percent to rotate over an average of “variable counts” encoder counts.

```
void counterClockWiseTurnShaft(int percent, int counts)//
using encoders
{
    //Resent encoder counts
    rightShaftEncoder.ResetCounts();
    leftShaftEncoder.ResetCounts();

    //Set both motors to desired percent
    leftMotor.SetPercent(percent * -1.0);
    rightMotor.SetPercent(percent);

    //While average of left and right encoder is less
than counts, keep running motors
    while ((leftShaftEncoder.Counts() +
rightShaftEncoder.Counts()) / 2.0 < counts) {
        Sleep(100);
    }
```



Overall Function Set:

This function set contains five functions for robot movement based on microswitch sensor value. The differences between these functions are shown in the table below and then expanded upon in the Individual Functions section below.

Function Name	Direction	Number of Microswitches Pressed	Parameters
driveForwardStopFrontBumpOne()	forward	1	none
driveForwardStopFrontBumpBoth()	forward	2	none
driveBackwardStopBackBumpOne()	backward	1	int percent
driveBackwardStopBackBumpBeforeRamp()	backward	1	int percent
driveBackwardStopBackBumpBoth()	backward	2	none

Individual Functions:

driveForwardStopFrontBumpOne()

Drive until one front bump sensor is hit

```
01 void driveForwardStopFrontBumpOne()
   {                                        //drive until one
   front bump sensor is hit
02     //drives forward until wall is hit, then it stops
03
04     while(fl.Value() == 1 && fr.Value() == 1) {
05         leftMotor.SetPercent(50);
06         rightMotor.SetPercent(50);
07     }
08
09     leftMotor.SetPercent(0);
10     rightMotor.SetPercent(0);
11
12     Sleep(1);
13 }
```

driveForwardStopFrontBumpBoth()

Drive until front bump sensors are hit

```
01 void driveForwardStopFrontBumpBoth()
   {                                        //drive until front
   bump sensors are hit
02     //drives forward until wall is hit, then it stops
03
04     while(fl.Value() == 1 || fr.Value() == 1) {
05         leftMotor.SetPercent(25);
06         rightMotor.SetPercent(25);
07     }
08
09     leftMotor.SetPercent(0);
10     rightMotor.SetPercent(0);
11
12     Sleep(1);
13 }
```

driveBackwardStopBackBumpOne(int percent)

Drive at “percent variable” motor percent until either back bump sensors is hit

```
01 void driveBackwardStopBackBumpOne(int percent)
   {                                        //drive until
   either back bump sensors is hit
02     //drives backward until wall is hit, then it stops
03
04     while(bl.Value() == 1 && br.Value() == 1) {
05         leftMotor.SetPercent(percent);
06         rightMotor.SetPercent(percent);
07     }
08
09     leftMotor.SetPercent(0);
10     rightMotor.SetPercent(0);
11
12     Sleep(1);
13 }
```

driveBackwardStopBackBumpBoth()

Drive until both back bump sensors are hit

```
01 void driveBackwardStopBackBumpBoth()
   {                                        //drive until both
   back bump sensors are hit
02     //drives backward until wall is hit, then it stops
03
04     while(bl.Value() == 1 || br.Value() == 1) {
05         leftMotor.SetPercent(-25);
06         rightMotor.SetPercent(-25);
07     }
08
09     leftMotor.SetPercent(0);
10     rightMotor.SetPercent(0);
11
12     Sleep(1);
13 }
```

driveBackwardStopBackBumpBeforeRamp(int percent)

Drives at “percent variable” motor percent until either back bump sensor is hit

```
01 void driveBackwardStopBackBumpBeforeRamp(int percent)
   {                        //drive until either back bump
   sensors is hit
02     //drives backward until wall is hit, then it stops
03
04     while(bl.Value() == 1 && br.Value() == 1 /*&&
   trayBump.Value() == 0*/) {
05         leftMotor.SetPercent(percent);
06         rightMotor.SetPercent(percent);
07     }
08
09     leftMotor.SetPercent(0);
10     rightMotor.SetPercent(0);
11
12     Sleep(1);
13 }
```