To learn about the overall logic of our code, please click here to view our Logic page. The final version of the code is broken down by function here.

The code's overall structure is discussed here.

Our current version of the code is found here. Past versions of the code are located in Performance Test 1, Performance Test 2, and Performance Test 3.

Our code logic is broken down into multiple parts: sensor hierarchy, the structure of the code itself (found here), and the overall code functionality algorithm.

Sensor Hierarchy:

The strategy the group developed to accomplish the set tasks focused on sensor input to control robot movement. The team focused on using microswitches to allow the robot to align with walls or obstacles, standardizing the robot's position throughout runs. After microswitches, the general order of sensor preference was to use optosensors for line following and finally, shaft encoding for general movement. The CdS cell was used when available to begin/end robot movement based on light detection. RPS was used to travel longer distances across the course. Figure B1 in Appendix B shows the proposed breakdown for which sensors would be used for each section of the course.

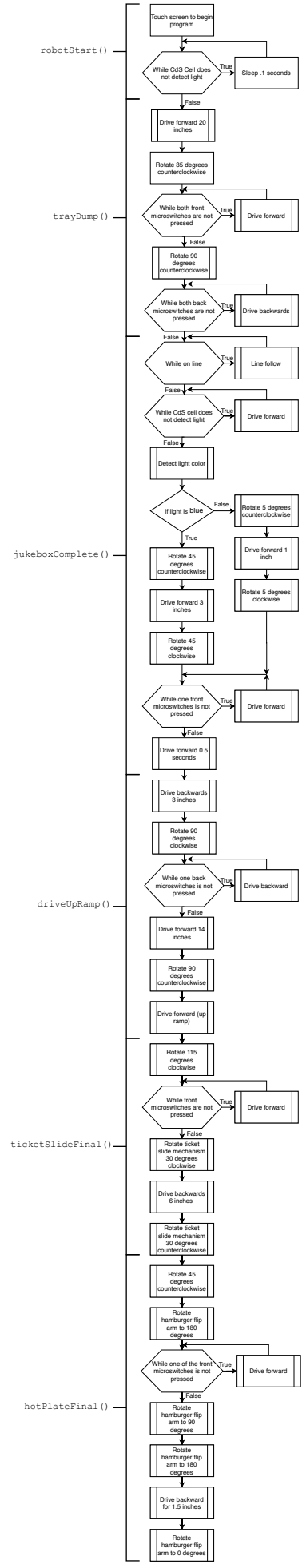
The group decided to use shaft encoding instead of powering a motor for a set amount of time because it was possible to get a more accurate reading for how far the robot had travelled at any point. Figure B1 in Appendix B also shows the areas where shaft encoding would be implemented. Most notably, shaft encoding was to be used to drive off of and onto the welcome mat, press the jukebox button, and drive up the ramp, as well as for turning.

Preliminary Algorithm

We originally wrote a preliminary algorithm, found here. An algorithm was originally chosen over pseudocode for its readability and over a flowchart so that it could be updated easier when finalized.

Final Code Logic (Flowchart)

Our final code logic was largely based on our preliminary algorithm. Some changes were made, most notably the hotplate flip functionality due to changed mechanism design. Please note, this flowchart only contains the finalized logic for the tray drop, jukebox button, drive up ramp, ticket slide, and hamburger flip. Please refer to the preliminary algorithm (above) for the logic for maneuvering to the lever flip and back to press the final button.



Assumptions

All variables are declared and initialized. All necessary C++ libraries have been imported.

Functions in Main Function

For the finalized code, the main function only contains six functions. These are the functions that break the flowchart into sections on the left. The complete list is:

`robotStart()`

`trayDump()`

`jukeboxComplete()`

`driveUpRamp()`

`ticketSlideFinal()`

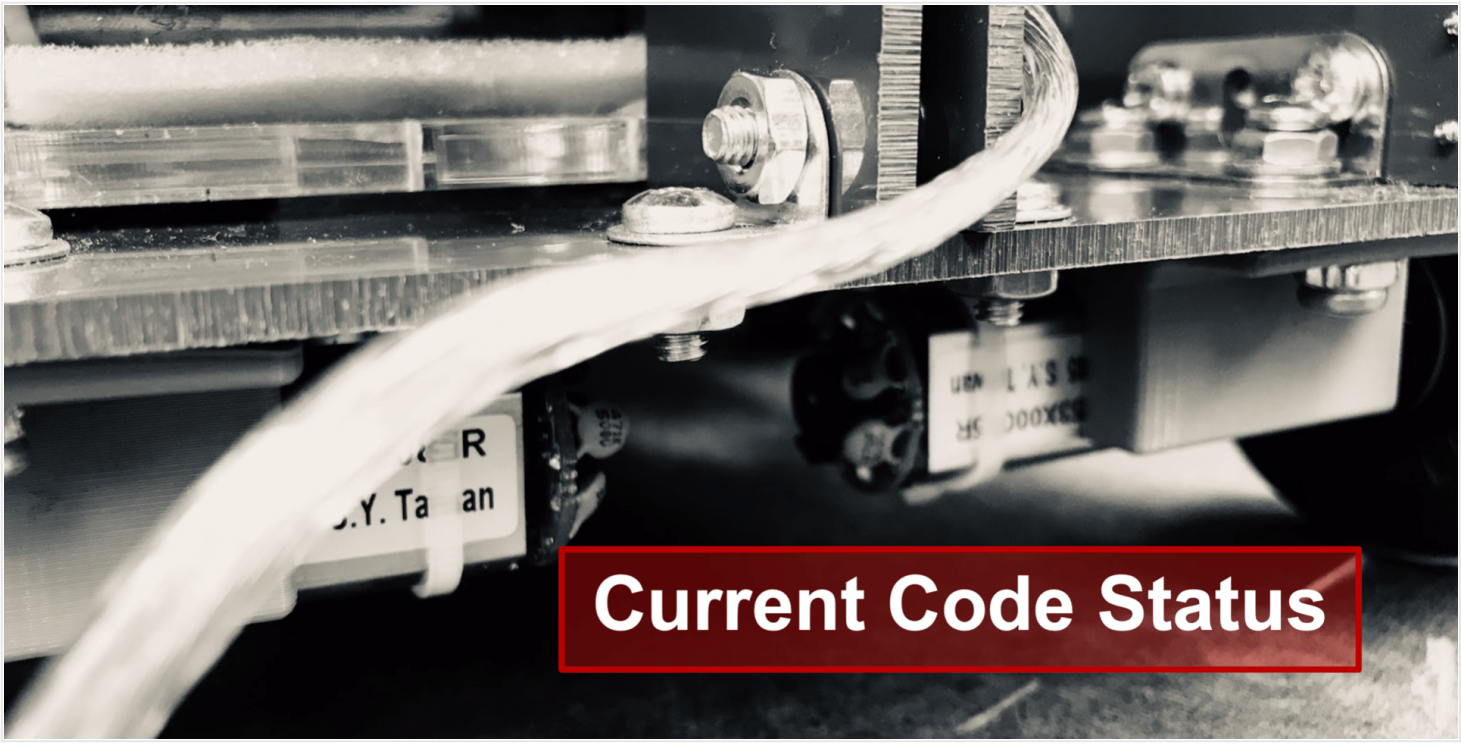
`hotPlateFinal()`

These functions are explained in more detail in "Function Sets" -> "Main Function"

Subprograms Referenced

Throughout the flowchart there are several subprograms referenced for driving forward/background, rotating, and line following. These are the most used functions and are explained in more detail in "Function Sets."

AMB 04/22/20



Overview:

The current code is similar to the version of the code in Performance Test 3 except that the hamburger flip was updated due to its mechanism revision. This means that the burger flip code no longer relies on RPS for navigation to the hotplate. To learn about how the structure of the code itself differs from the code shown in Performance Test 3, please click here.

Code:

View our full code below, or open as a PDF by clicking here. Hovering over code below provides the option (in the top right corner) to open it as a text file.

```
0001 //Include preprocessor directives
0002 #include <FEHLCD.h>
0003 #include <FEHIO.h>
0004 #include <FEHUtility.h>
0005 #include <FEHMotor.h>
0006 #include <FEHServo.h>
0007 #include <FEHAccel.h>
0008 #include <FEHBattery.h>
0009 #include <FEHBuzzer.h>
0010 #include <FEHRPS.h>
0011 #include <FEHSD.h>
0012 #include <string.h>
0013 #include <stdio.h>
0014
0015 AnalogInputPin left_input(FEHIO::P0_0);          //line
     followers - called encoders throughout code
0016 AnalogInputPin center_input(FEHIO::P0_1);
0017 AnalogInputPin right_input(FEHIO::P0_2);
0018
0019 AnalogInputPin cdsCell(FEHIO::P2_0);         //CdS Cell
0020
0021 FEHMotor leftMotor(FEHMotor::Motor0, 9);         //motors
```

```
0022 FEHMotor rightMotor (FEHMotor::Motor1, 9);
0023
0024 DigitalInputPin fl(FEHIO::P1_2);                //bump
     switches
0025 DigitalInputPin fr(FEHIO::P0_3);
0026 DigitalInputPin bl(FEHIO::P1_3);
0027 DigitalInputPin br(FEHIO::P1_1);
0028 DigitalInputPin trayBump(FEHIO::P3_7);
0029
0030 DigitalEncoder
     rightShaftEncoder(FEHIO::P1_0);        //shaft encoders
0031 DigitalEncoder leftShaftEncoder(FEHIO::P3_7);
0032
0033 FEHServo ticketSlideServo(FEHServo::Servo0); //servo
     motors
0034 FEHServo iceCreamServo(FEHServo::Servo1);
0035 FEHServo burgerFlipServo(FEHServo::Servo2);
0036
0037 #define TICKET_SERVO_MIN 1300       //servo motor mins,
     maxes, and angles
0038 #define TICKET_SERVO_MAX 1573
0039 #define TICKET_SLIDE_DEGREE 180.0
0040
0041 #define ICECREAM_SERVO_MIN 1121       //servo motor
     mins, maxes, and angles
0042 #define ICECREAM_SERVO_MAX 1954
0043 #define ICECREAM_SLIDE_DEGREE 180.0
0044
0045 #define BURGER_SERVO_MIN 800       //servo motor mins,
     maxes, and angles
0046 #define BURGER_SERVO_MAX 2369
0047 #define BURGER_SLIDE_DEGREE 180.0
0048
0049 #define CdS_CELL_NO_LIGHT_VALUE 2.0
     //CdS cell values for jukebox
0050 #define LINE_FOLLOWER_TO_CDS_READING_JUKEBOX 3.5
0051 #define CdS_RED_MIN 0.0
0052 #define CdS_RED_MAX 0.5
0053 #define CdS_BLUE_MIN 0.55
0054 #define CdS_BLUE_MAX 1.5
0055
0056 #define LEFT_LINE_READER_MID_VAL 2.0
     //line readers middle values
0057 #define CENTER_LINE_READER_MID_VAL 2.0
0058 #define RIGHT_LINE_READER_MID_VAL 2.75
0059
0060
0061
0062 /*
0063  * Global variables
0064  */
0065 int analogGlobal [3] = {0, 0, 0};   //global array for
     line reading
0066 bool redLight = false;
0067 bool blueLight = false;
0068
0069 /*
0070  * Functions
0071  */
0072 void printValues() {
0073     LCD.Write("");
```

```
0074     LCD.Write(analogGlobal[0]);
0075     LCD.Write(", ");
0076     LCD.Write(analogGlobal[1]);
0077     LCD.Write(", ");
0078     LCD.WriteLine(analogGlobal[2]);
0079 }
0080
0081 void updateEncoder() {
0082     if (left_input.Value() > LEFT_LINE_READER_MID_VAL) {
0083         analogGlobal[0] = 1;
0084     }
0085     else {
0086         analogGlobal[0] = 0;
0087     }
0088
0089     if (center_input.Value() >
     CENTER_LINE_READER_MID_VAL) {
0090         analogGlobal[1] = 1;
0091     } else {
0092         analogGlobal[1] = 0;
0093     }
0094
0095     if (right_input.Value() > RIGHT_LINE_READER_MID_VAL)
     {
0096         analogGlobal[2] = 1;
0097     } else {
0098         analogGlobal[2] = 0;
0099     }
0100     LCD.Write(analogGlobal[0]);
0101     LCD.Write(analogGlobal[1]);
0102     LCD.WriteLine(analogGlobal[2]);
0103     printValues();
0104 }
0105
0106 void initializeRobot() {
0107
0108     LCD.Clear(FEHLCD::Black);
0109     LCD.SetFontColor(FEHLCD::White);
0110
0111     burgerFlipServo.SetMin(BURGER_SERVO_MIN);
0112     burgerFlipServo.SetMax(BURGER_SERVO_MAX);
0113     burgerFlipServo.SetDegree(BURGER_SLIDE_DEGREE);
0114
0115     iceCreamServo.SetMin(ICECREAM_SERVO_MIN);
0116     iceCreamServo.SetMax(ICECREAM_SERVO_MAX);
0117     iceCreamServo.SetDegree(145);
0118
0119     ticketSlideServo.SetMin(TICKET_SERVO_MIN);
0120     ticketSlideServo.SetMax(TICKET_SERVO_MAX);
0121     ticketSlideServo.SetDegree(0);
0122
0123
0124     updateEncoder();
0125 }
0126
0127 void touchScreen() {
0128     float x, y;
0129     LCD.WriteLine("Next test:");
0130     LCD.ClearBuffer(); //reset values
0131     while(!LCD.Touch(&x,&y)); //Wait for screen to be
     pressed
```

```
0132     while(LCD.Touch(&x,&y)); //Wait for screen to be
     unpressed
0133     LCD.Clear(); //clear screent
0134 }
0135
0136 void turnRight(int i)
     {                                  //using only motor power
0137     rightMotor.SetPercent(0);
0138     leftMotor.SetPercent(12.5*i);
0139     Sleep(10);
0140 }
0141
0142 void turnLeft(int i)
     {                                   //using only motor
     power
0143     leftMotor.SetPercent(0);
0144     rightMotor.SetPercent(12.5*i);
0145     Sleep(10);
0146 }
0147
0148 void driveStraight(float time) {
0149     leftMotor.SetPercent(25);
0150     rightMotor.SetPercent(25);
0151     Sleep(time);
0152 }
0153
0154 /*
0155  * Line following
0156  */
0157 void followLineToJukeBox() {
0158
0159     /*
0160     leftMotor.SetPercent(25);
0161     Sleep(.5);
0162     rightMotor.SetPercent(25);
0163     leftMotor.SetPercent(0);
0164
0165     while (analogGlobal[0] + analogGlobal[1] +
     analogGlobal[2] < 2) {
0166         updateEncoder();
0167         Sleep(.02);
0168     }
0169
0170     leftMotor.SetPercent(0);
0171     rightMotor.SetPercent(0);
0172
0173     leftMotor.SetPercent(25);
0174     rightMotor.SetPercent(25);
0175     Sleep(.2);
0176     leftMotor.SetPercent(0);
0177     rightMotor.SetPercent(0);
0178     Sleep(.1);
0179     */
0180     while (analogGlobal[0] != 0 || analogGlobal[1] != 0
     || analogGlobal[2] != 0) {
0181         printValues();
0182         updateEncoder();
0183         if (analogGlobal[0] == 0 && analogGlobal[1] == 0
     && analogGlobal[2] == 1) {
0184             turnRight(2);
0185         } else if (analogGlobal[0] == 0 &&
```

```
    analogGlobal[1] == 1 && analogGlobal[2] == 0) {
0186            driveStraight(10);
0187        } else if (analogGlobal[0] == 1 &&
    analogGlobal[1] == 0 && analogGlobal[2] == 0) {
0188            turnLeft(2);
0189        } else if (analogGlobal[0] == 1 &&
    analogGlobal[1] == 1 && analogGlobal[2] == 0) {
0190            turnLeft(1);
0191        } else if (analogGlobal[0] == 0 &&
    analogGlobal[1] == 1 && analogGlobal[2] == 1) {
0192            turnRight(1);
0193        } else if (analogGlobal[0] == 1 &&
    analogGlobal[1] == 0 && analogGlobal[2] == 1) {
0194            driveStraight(10);
0195        } else if (analogGlobal[0] == 1 &&
    analogGlobal[1] == 1 && analogGlobal[2] == 1) {
0196            driveStraight(10);
0197        }
0198    }
0199 }
0200
0201 void driveForwardStopFrontBumpBoth()
    {                                       //drive until front
    bump sensors are hit
0202    //drives forward until wall is hit, then it stops
0203
0204    while(fl.Value() == 1 || fr.Value() == 1) {
0205        leftMotor.SetPercent(25);
0206        rightMotor.SetPercent(25);
0207    }
0208
0209    leftMotor.SetPercent(0);
0210    rightMotor.SetPercent(0);
0211
0212    Sleep(1);
0213 }
0214
0215 void driveLeftForwardStopFrontBump()
    {                                       //drive until front
    bump sensors are hit
0216    //drives forward until wall is hit, then it stops
0217
0218    while(fl.Value() == 1) {
0219        leftMotor.SetPercent(30);
0220        rightMotor.SetPercent(0);
0221    }
0222
0223    leftMotor.SetPercent(0);
0224    rightMotor.SetPercent(0);
0225
0226    Sleep(1);
0227 }
0228
0229 void driveForwardStopFrontBumpOne()
    {                                       //drive until one
    front bump sensor is hit
0230    //drives forward until wall is hit, then it stops
0231
0232    while(fl.Value() == 1 && fr.Value() == 1) {
0233        leftMotor.SetPercent(50);
0234        rightMotor.SetPercent(50);
```

```
0235     }
0236
0237     leftMotor.SetPercent(0);
0238     rightMotor.SetPercent(0);
0239
0240     Sleep(1);
0241 }
0242
0243 void driveBackwardStopBackBumpOne(int percent)
     {                                    //drive until
     either back bump sensors is hit
0244     //drives backward until wall is hit, then it stops
0245
0246     while(bl.Value() == 1 && br.Value() == 1) {
0247         leftMotor.SetPercent(percent);
0248         rightMotor.SetPercent(percent);
0249     }
0250
0251     leftMotor.SetPercent(0);
0252     rightMotor.SetPercent(0);
0253
0254     Sleep(1);
0255 }
0256
0257 void driveBackwardStopBackBumpBeforeRamp(int percent)
     {                     //drive until either back bump
     sensors is hit
0258     //drives backward until wall is hit, then it stops
0259
0260     while(bl.Value() == 1 && br.Value() == 1 /*&&
     trayBump.Value() == 0*/) {
0261         leftMotor.SetPercent(percent);
0262         rightMotor.SetPercent(percent);
0263     }
0264
0265     leftMotor.SetPercent(0);
0266     rightMotor.SetPercent(0);
0267
0268     Sleep(1);
0269 }
0270
0271 void driveBackwardStopBackBumpBoth()
     {                                    //drive until both
     back bump sensors are hit
0272     //drives backward until wall is hit, then it stops
0273
0274     while(bl.Value() == 1 || br.Value() == 1) {
0275         leftMotor.SetPercent(-25);
0276         rightMotor.SetPercent(-25);
0277     }
0278
0279     leftMotor.SetPercent(0);
0280     rightMotor.SetPercent(0);
0281
0282     Sleep(1);
0283 }
0284
0285 /*
0286  * Methods for rotation
0287  */
0288 void rotateClockwiseBumpSwitch()
```

```
{                               //using bump switches
    while(bl.Value() == 1 || br.Value() == 1) {
        if (br.Value() == 1) {
             rightMotor.SetPercent(-25);
        } else {
            leftMotor.SetPercent(25);
            rightMotor.SetPercent(-25);
        }

    }

    leftMotor.SetPercent(0);
    rightMotor.SetPercent(0);

    Sleep(1);
}

void rotateCounterClockBumpSwitch()
{                              //using bump switches
    while(bl.Value() == 1 || br.Value() == 1) {
        if (bl.Value() == 1) {
             leftMotor.SetPercent(-25);
        } else {
            leftMotor.SetPercent(-25);
            rightMotor.SetPercent(25);
        }

    }

    leftMotor.SetPercent(0);
    rightMotor.SetPercent(0);

    Sleep(1);
}

void jukeboxLightReading() {

    while (cdsCell.Value() > CdS_BLUE_MAX && fr.Value()
== 1 && fl.Value() == 1) {
        //drive forward
        rightMotor.SetPercent(25);
        leftMotor.SetPercent(25);
    }

    rightMotor.SetPercent(0);
    leftMotor.SetPercent(0);

    Sleep(.1);

    //test for light
    if (cdsCell.Value() < CdS_BLUE_MIN) {
        redLight = true;
    } else {
        blueLight = true;   //fingers crossed it's
Annie's favorite color
    }

    //print color of light ot screen
    if (redLight) {
        LCD.WriteLine("Red light.");
    } else if (blueLight) {
```

```
0346         LCD.WriteLine("Blue light.");
0347     } else {
0348         LCD.WriteLine("No light detected. This is
     bad.");
0349     }
0350 }
0351
0352 void driveShaft(int percent, int counts) //using
     encoders
0353 {
0354     //Reset encoder counts
0355     rightShaftEncoder.ResetCounts();
0356     leftShaftEncoder.ResetCounts();
0357
0358     //Set both motors to desired percent
0359     rightMotor.SetPercent(percent);
0360     leftMotor.SetPercent(percent);
0361
0362     //While the average of the left and right encoder is
     less than counts,
0363     //keep running motors
0364     while((leftShaftEncoder.Counts() +
     rightShaftEncoder.Counts()) / 2. < counts);
0365
0366     //Turn off motors
0367     rightMotor.Stop();
0368     leftMotor.Stop();
0369 }
0370
0371 void driveShaftSeparate(int leftPercent, intrightPercent, int counts) //using
     encoders
0372 {
0373     //Reset encoder counts
0374     rightShaftEncoder.ResetCounts();
0375     leftShaftEncoder.ResetCounts();
0376
0377     //Set both motors to desired percent
0378     rightMotor.SetPercent(rightPercent);
0379     leftMotor.SetPercent(leftPercent);
0380
0381     //While the average of the left and right encoder is
     less than counts,
0382     //keep running motors
0383     while((leftShaftEncoder.Counts() +
     rightShaftEncoder.Counts()) / 2. < counts);
0384
0385     //Turn off motors
0386     rightMotor.Stop();
0387     leftMotor.Stop();
0388 }
0389
0390
0391 void clockwiseTurnShaft(int percent, int counts) //
     using encoders
0392 {
0393     //Resent encoder counts
0394     rightShaftEncoder.ResetCounts();
0395     leftShaftEncoder.ResetCounts();
0396
0397     //Set both motors to desired percent
0398     rightMotor.SetPercent(percent * -1.0);
```

```
0399     leftMotor.SetPercent(percent);
0400 
0401     //While average of left and right encoder is less
     than counts, keep running motors
0402     while ((leftShaftEncoder.Counts() +
     rightShaftEncoder.Counts()) / 2.0 < counts) {
0403         Sleep(100);
0404     }
0405 
0406     //Turn off motors
0407     rightMotor.Stop();
0408     leftMotor.Stop();
0409 }
0410 
0411 void clockwiseTurnShaftTicket(int percent, int counts)//
     using encoders
0412 {
0413     //Resent encoder counts
0414     rightShaftEncoder.ResetCounts();
0415     leftShaftEncoder.ResetCounts();
0416 
0417     //Set both motors to desired percent
0418     rightMotor.SetPercent(percent * -1.0);
0419     leftMotor.SetPercent(percent);
0420 
0421     //While average of left and right encoder is less
     than counts, keep running motors
0422     float timeCurrent = TimeNow();
0423     while ((leftShaftEncoder.Counts() +
     rightShaftEncoder.Counts()) / 2.0 < counts && TimeNow()
     - timeCurrent < 2) {
0424         Sleep(100);
0425     }
0426 
0427     //Turn off motors
0428     rightMotor.Stop();
0429     leftMotor.Stop();
0430 }
0431 
0432 void counterClockWiseTurnShaft(int percent, int counts)//
     using encoders
0433 {
0434     //Resent encoder counts
0435     rightShaftEncoder.ResetCounts();
0436     leftShaftEncoder.ResetCounts();
0437 
0438     //Set both motors to desired percent
0439     leftMotor.SetPercent(percent * -1.0);
0440     rightMotor.SetPercent(percent);
0441 
0442     //While average of left and right encoder is less
     than counts, keep running motors
0443     while ((leftShaftEncoder.Counts() +
     rightShaftEncoder.Counts()) / 2.0 < counts) {
0444         Sleep(100);
0445     }
0446 
0447     //Turn off motors
0448     rightMotor.Stop();
0449     leftMotor.Stop();
0450 }
```

```
0451
0452 void turnForTime(int leftPercent, int rightPercent,float timeSeconds) //
     using encoders
0453 {
0454
0455
0456     //Set both motors to desired percent
0457     leftMotor.SetPercent(leftPercent);
0458     rightMotor.SetPercent(rightPercent);
0459
0460     Sleep(timeSeconds);
0461
0462     //Turn off motors
0463     rightMotor.Stop();
0464     leftMotor.Stop();
0465 }
0466
0467 void counterClockWiseTurnRightOnly(int percent, intcounts) //
     using encoders
0468 {
0469     //Resent encoder counts
0470     rightShaftEncoder.ResetCounts();
0471
0472
0473     //Set both motors to desired percent
0474
0475     rightMotor.SetPercent(percent);
0476
0477     //While average of left and right encoder is less
     than counts, keep running motors
0478     while ((rightShaftEncoder.Counts()) < counts) {
0479         Sleep(100);
0480     }
0481
0482     //Turn off motors
0483     rightMotor.Stop();
0484 }
0485
0486 void jukeboxButton() {
0487
0488     if (blueLight) {
0489       counterClockWiseTurnShaft(25, 50);
0490       driveShaft(25, 115);
0491       clockwiseTurnShaft(25, 50);
0492       rightMotor.SetPercent(0);
0493       leftMotor.SetPercent(0);
0494       driveForwardStopFrontBumpOne();
0495
0496       rightMotor.SetPercent(0);
0497       leftMotor.SetPercent(25);
0498       Sleep(.5);
0499
0500       rightMotor.SetPercent(0);
0501       leftMotor.SetPercent(0);
0502
0503     } else {
0504
0505         counterClockWiseTurnShaft(25, 7);
0506         driveShaft(25, 10);
0507         clockwiseTurnShaft(25, 7);
0508         driveForwardStopFrontBumpOne();
```

```
0509
0510        rightMotor.SetPercent(0);
0511        leftMotor.SetPercent(25);
0512        Sleep(.5);
0513
0514        rightMotor.SetPercent(0);
0515        leftMotor.SetPercent(0);
0516    }
0517 }
0518
0519 void lightStart() {
0520    while (cdsCell.Value() > CdS_BLUE_MAX) {
0521        Sleep(.1);
0522    }
0523 }
0524
0525 void findLine() {
0526    driveShaft(25, 10);
0527 }
0528
0529 void ticketSlide() {
0530    driveForwardStopFrontBumpBoth();
0531
0532    ticketSlideServo.SetMin(TICKET_SERVO_MIN);
0533    ticketSlideServo.SetMax(TICKET_SERVO_MAX);
0534
0535    ticketSlideServo.SetDegree(TICKET_SLIDE_DEGREE);
0536    Sleep(.5);
0537
0538    //Reset encoder counts
0539    rightShaftEncoder.ResetCounts();
0540    leftShaftEncoder.ResetCounts();
0541
0542    float currentTime = TimeNow();
0543    while (TimeNow() - currentTime < 2) {
0544        rightMotor.SetPercent(-25);
0545        leftMotor.SetPercent(-20);
0546    }
0547
0548    //Set both motors to stop, then sleep
0549    rightMotor.SetPercent(0);
0550    leftMotor.SetPercent(0);
0551    Sleep(.1);
0552
0553    //release hook from ticket slide
0554    driveShaft(25, 25);
0555    Sleep(.1);
0556    ticketSlideServo.SetDegree(-1.0*TICKET_SLIDE_DEGREE);
0557    Sleep(.1);
0558
0559 }
0560
0561 void batteryVoltage() {
0562    LCD.WriteLine("Battery: ");
0563    LCD.Write(Battery.Voltage());
0564 }
0565
0566 void driveUntilLine(int percent) {
0567    rightShaftEncoder.ResetCounts();
0568    leftShaftEncoder.ResetCounts();
0569    while (analogGlobal[0] != 1 && analogGlobal[1] != 1
```

```
    && analogGlobal[2] != 1) {
        updateEncoder();
        rightMotor.SetPercent(percent);
        leftMotor.SetPercent(percent);
    }

    rightMotor.Stop();
    leftMotor.Stop();
    Sleep(.1);
}

void driveOnRPSX(float coord, float ycoord) {
    //until bump
    float RPSx = RPS.X();
    float RPSy = RPS.Y();

    while (RPSx<coord) {
        leftMotor.SetPercent(20);
        rightMotor.SetPercent(25);
        Sleep(100);
        LCD.WriteLine(RPSx);
        leftMotor.SetPercent(0);
        rightMotor.SetPercent(0);
        Sleep(100);
        RPSx = RPS.X();
    }

    float heading = RPS.Heading();
    while (heading<178 || heading>185) {

        if (heading < 180 && heading > 0) {
            leftMotor.SetPercent(-25);
            rightMotor.SetPercent(25);
        }
        if (heading > 180) {
            leftMotor.SetPercent(-25);
            rightMotor.SetPercent(25);
        }
        Sleep(50);
        LCD.WriteLine(heading);
        leftMotor.SetPercent(0);
        rightMotor.SetPercent(0);
        Sleep(100);

        heading = RPS.Heading();
    }


    while (fr.Value() == 1 && fl.Value() == 1 &&
RPSy<ycoord) {
        if (RPSx>0){
        leftMotor.SetPercent(25+4*(coord-RPSx));
        rightMotor.SetPercent(25+4*(RPSx-coord));
        }
        Sleep(100);
        LCD.Write(RPSx);
        LCD.Write(" ");
        LCD.Write(RPSy);
        leftMotor.SetPercent(0);
        rightMotor.SetPercent(0);
        Sleep(100);
```

```
0629          RPSx = RPS.X();
0630          RPSy = RPS.Y();
0631
0632      }
0633  }
0634
0635  void alignWithHotPlate() {
0636
0637      driveUntilLine(30); //drive forward until line in
      front of hotplate is reached
0638      counterClockWiseTurnShaft(25, 40); //rotate to align
0639      driveShaft(15, 70);//Drive forward
0640      Sleep(1000);
0641      LCD.WriteLine("Forward 1");
0642      clockwiseTurnShaft(25, 33); //rotate to align
0643      LCD.WriteLine("Turn 1");
0644      Sleep(1000);
0645      driveForwardStopFrontBumpOne();//Drive forward
0646      LCD.WriteLine("Forward 2");
0647      Sleep(1000);
0648      counterClockWiseTurnRightOnly(25, -10);
0649      counterClockWiseTurnShaft(25, 17); //rotate to align
0650      LCD.WriteLine("turn 2");
0651      Sleep(1000);
0652      driveForwardStopFrontBumpBoth();//Drive forward
0653      LCD.WriteLine("Done");
0654
0655
0656      /*
0657       * Inserting mechanism into hot plate.
0658       * If right bump switch is pressed, back up slightly
      and alter angle.
0659       */
0660
0661  //    while (fr.Value() == 1 & fl.Value() == 1) {
0662  //        leftMotor.SetPercent(25);
0663  //        rightMotor.SetPercent(20);
0664  // /*
0665  //        if (fr.Value() == 0) {
0666  //            driveShaft(-25, 20);
0667  //            counterClockWiseTurnShaft(25, 5);
0668  //            leftMotor.SetPercent(25);
0669  //            rightMotor.SetPercent(20);
0670  //            Sleep(.5);
0671  //        }
0672  //        */
0673  //    }
0674  //    while (fl.Value() == 1) {
0675  //    leftMotor.SetPercent(25);
0676  //    rightMotor.SetPercent(0);
0677  //    }
0678
0679  }
0680
0681  /* return to start and push button*/
0682  // call: check_x_minus(the target x coordinate);
0683  void check_x_minus(float x_coordinate) //using RPS while
      robot is in the -x direction
0684  {
0685      //check if receiving proper RPS coordinates and
      whether the robot is within an acceptable range
```

```
0686     while(RPS.X()>0  && (RPS.X() < x_coordinate - 2 ||
     RPS.X() > x_coordinate + 2))
0687     {
0688         if(RPS.X() > x_coordinate)
0689         {
0690                 //pulse the motors for a short duration
     in the correct direction
0691                 rightMotor.SetPercent(20);
0692                 leftMotor.SetPercent(20);
0693                 Sleep(0.1);
0694         }
0695         if(RPS.X() < x_coordinate)
0696         {
0697                 //pulse the motors for a short duration
     in the correct direction
0698                 rightMotor.SetPercent(-20);
0699                 leftMotor.SetPercent(-20);
0700                 Sleep(0.1);
0701         }
0702         //stop motors
0703         rightMotor.SetPercent(0);
0704         leftMotor.SetPercent(0);
0705     }
0706 }
0707
0708 void check_x_plus(float x_coordinate) //using RPS while
     robot is in the +x direction
0709 {
0710     //check if receiving proper RPS coordinates and
     whether the robot is within an acceptable range
0711     while(RPS.X()>0  && (RPS.X() < x_coordinate - 2 ||
     RPS.X() > x_coordinate + 2))
0712     {
0713         if(RPS.X() > x_coordinate)
0714         {
0715                 //pulse the motors for a short duration
     in the correct direction
0716                 rightMotor.SetPercent(-20);
0717                 leftMotor.SetPercent(-20);
0718                 Sleep(0.1);
0719         }
0720         if(RPS.X() < x_coordinate)
0721         {
0722                 //pulse the motors for a short duration
     in the correct direction
0723                 rightMotor.SetPercent(20);
0724                 leftMotor.SetPercent(20);
0725                 Sleep(0.1);
0726         }
0727         //stop motors
0728         rightMotor.SetPercent(0);
0729         leftMotor.SetPercent(0);
0730     }
0731 }
0732
0733 void check_y_minus(float y_coordinate) //using RPS while
     robot is in the -y direction
0734 {
0735     //check if receiving proper RPS coordinates and
     whether the robot is within an acceptable range
0736     while(RPS.Y()>0  && (RPS.Y() < y_coordinate - 2 ||
```

```
    RPS.Y() > y_coordinate + 2))
0737    {
0738        if(RPS.Y() > y_coordinate)
0739        {
0740                //pulse the motors for a short duration
    in the correct direction
0741                rightMotor.SetPercent(20);
0742                leftMotor.SetPercent(20);
0743                Sleep(0.1);
0744        }
0745        if(RPS.Y() < y_coordinate)
0746        {
0747                //pulse the motors for a short duration
    in the correct direction
0748                rightMotor.SetPercent(-20);
0749                leftMotor.SetPercent(-20);
0750                Sleep(0.1);
0751        }
0752        //stop motors
0753        rightMotor.SetPercent(0);
0754        leftMotor.SetPercent(0);
0755    }
0756 }
0757
0758 void check_y_plus(float y_coordinate) //using RPS while
    robot is in the +y direction
0759 {
0760    //check if receiving proper RPS coordinates and
    whether the robot is within an acceptable range
0761    while(RPS.Y()>0  && (RPS.Y() < y_coordinate - 2 ||
    RPS.Y() > y_coordinate + 2))
0762    {
0763        if(RPS.Y() > y_coordinate)
0764        {
0765                //pulse the motors for a short duration
    in the correct direction
0766                rightMotor.SetPercent(-20);
0767                leftMotor.SetPercent(-20);
0768                Sleep(0.1);
0769        }
0770        if(RPS.Y() < y_coordinate)
0771        {
0772                //pulse the motors for a short duration
    in the correct direction
0773                rightMotor.SetPercent(20);
0774                leftMotor.SetPercent(20);
0775                Sleep(0.1);
0776        }
0777        //stop motors
0778        rightMotor.SetPercent(0);
0779        leftMotor.SetPercent(0);
0780    }
0781 }
0782
0783 void driveToLever(int percent) {
0784        followLineToJukeBox();
0785        driveForwardStopFrontBumpBoth();
0786        iceCreamServo.SetDegree(180.);
0787        Sleep(7);
0788        iceCreamServo.SetDegree(0.);
0789 }
```

```
0790
0791 // call: check_heading(the degree of the QR code's
     target orientation);
0792 void check_heading(float heading) //using RPS
0793 {
0794     if (heading != 0){
0795         //check if receiving proper RPS coordinates and
     whether the robot is within an acceptable range
0796         while (RPS.Heading() >= 0 && RPS.Heading() <
     heading -1 || RPS.Heading() > heading +1)
0797         {
0798             if ((RPS.Heading()<heading-1 ||
     RPS.Heading()==359) && RPS.Heading() != -1)
0799             //will potentially be used to turn to
     45degrees from 359, don't need to worry about turning to
     1,0,359 because that's included in separate loop
0800             {
0801                 //pulse the motors for a short duration
     in the correct direction
0802                 rightMotor.SetPercent(25);
0803                 leftMotor.SetPercent(-25);
0804                 Sleep(.1);
0805
0806                 //stop motors
0807                 rightMotor.SetPercent(0);
0808                 leftMotor.SetPercent(0);
0809
0810             }
0811             if(RPS.Heading()>heading+1 && RPS.Heading()
     != -1)
0812             {
0813                 //pulse the motors for a short duration
     in the correct direction
0814                 rightMotor.SetPercent(-25);
0815                 leftMotor.SetPercent(25);
0816                 Sleep(.1);
0817
0818                 //stop motors
0819                 rightMotor.SetPercent(0);
0820                 leftMotor.SetPercent(0);
0821             }
0822         }
0823     } else { //if target heading is 0
0824         //check if receiving proper RPS coordinates and
     whether the robot is within an acceptable range
0825         while (RPS.Heading() > 1 && RPS.Heading() < 359)
0826         {
0827             if (RPS.Heading() > 1 && RPS.Heading() <
     180)
0828             {
0829                 //pulse the motors for a short duration
     in the correct direction
0830                 rightMotor.SetPercent(-25);
0831                 leftMotor.SetPercent(25);
0832                 Sleep(.1);
0833
0834                 //stop motors
0835                 rightMotor.SetPercent(0);
0836                 leftMotor.SetPercent(0);
0837             }
0838             if (RPS.Heading() < 359 && RPS.Heading() >
```

```
    180)
0839            {
0840                //pulse the motors for a short duration
    in the correct direction
0841                rightMotor.SetPercent(25);
0842                leftMotor.SetPercent(-25);
0843                Sleep(.1);
0844
0845                //stop motors
0846                rightMotor.SetPercent(0);
0847                leftMotor.SetPercent(0);
0848            }
0849        }
0850    }
0851 }
0852
0853 void robotStart() {
0854     LCD.Clear( FEHLCD::Black );
0855     LCD.SetFontColor( FEHLCD::White );
0856     RPS.InitializeTouchMenu();
0857
0858     initializeRobot();
0859     /*
0860      * Start with light
0861      */
0862     touchScreen();
0863     lightStart();
0864 }
0865
0866 void trayDump() {
0867     /*
0868      * Drive forward, turn, then drive forward until
    front bump switches pressed
0869      */
0870     driveShaft(25, 300);
0871     counterClockWiseTurnShaft(25, 50);
0872     driveForwardStopFrontBumpBoth();
0873
0874     /*
0875      * 90 degree turn counterclockwise - using shaft
    encoders
0876      */
0877     leftMotor.Stop();
0878     rightMotor.Stop();
0879     Sleep(1.0);
0880     driveShaft(-25, 25);
0881      counterClockWiseTurnShaft(25, 145);
0882
0883      /*
0884       * Drive backward until back bump switches pressed.
0885       */
0886      leftMotor.Stop();
0887      rightMotor.Stop();
0888      Sleep(1.0);
0889      driveBackwardStopBackBumpOne(-75);
0890 }
0891
0892 void jukeboxComplete() {
0893     /*
0894      * Follow line until there isn't a line anymore.
0895
```

```
0896      leftMotor.Stop();
0897      rightMotor.Stop();
0898      Sleep(1.0);
0899      updateEncoder();
0900     findLine();
0901     followLineToJukeBox();
0902     leftMotor.Stop();
0903     rightMotor.Stop();
0904     clockwiseTurnShaft(25, 5);
0905
0906     /*
0907      * Read CdS cell
0908      */
0909     leftMotor.Stop();
0910     rightMotor.Stop();
0911     Sleep(1.0);
0912     jukeboxLightReading();
0913
0914     /*
0915      * Press button
0916
0917     leftMotor.Stop();
0918     rightMotor.Stop();
0919     Sleep(1.0);
0920
0921     if (fl.Value() == 1 && fr.Value() == 1) {
0922         jukeboxButton();
0923     }
0924   */
0925
0926     driveForwardStopFrontBumpOne();
0927
0928 }
0929
0930 void driveUpRamp() {
0931     /*
0932      * Align with ramp
0933      */
0934     driveShaft(-25, 40);
0935     counterClockWiseTurnShaft(25, 125);
0936     Sleep(.2);
0937     driveBackwardStopBackBumpBeforeRamp(-25);
0938     Sleep(.2);
0939     driveShaft(25, 340);
0940     counterClockWiseTurnShaft(25, 120);
0941
0942
0943     batteryVoltage();
0944
0945     /*
0946      * Drive up ramp
0947      */
0948     driveShaft(50, 540);
0949 }
0950
0951 void ticketSlideFinal() {
0952     /*
0953      * Ticket slide
0954      */
0955     clockwiseTurnShaftTicket(25, 120); //90 degree turn
0956     driveShaft(50, 20);
```

```
0957     clockwiseTurnShaftTicket(25, 35); //90 degree turn
0958     ticketSlide();
0959 }
0960
0961 void hotPlateFinal() {
0962     /*
0963      * Align with hot plate
0964      */
0965     counterClockWiseTurnShaft(25, 50);
0966     burgerFlipServo.SetDegree(0);
0967     driveForwardStopFrontBumpOne();
0968
0969     /*
0970      * Turn hot plate
0971      */
0972     burgerFlipServo.SetDegree(95);
0973     Sleep (.5);
0974     burgerFlipServo.SetDegree(0);
0975 }
0976
0977 int main(void)
0978 {
0979
0980     /*
0981      * Final code
0982      */
0983     robotStart();
0984     trayDump();
0985     jukeboxComplete();
0986     driveUpRamp();
0987     ticketSlideFinal();
0988     hotPlateFinal();
0989
0990     /*
0991      * Code still in development
0992      */
0993
0994     //drive to lever quick fix - shaft encoding
0995     driveUntilLine(-30);
0996     burgerFlipServo.SetDegree(BURGER_SLIDE_DEGREE);
0997     counterClockWiseTurnShaft(25, 140);
0998     Sleep(.1);
0999     driveShaft(25, 75);
1000     Sleep(.1);
1001     clockwiseTurnShaft(25, 50);
1002     Sleep(.1);
1003     turnForTime(25, 25, 7);
1004
1005     iceCreamServo.SetDegree(0);
1006     Sleep(8.);
1007     iceCreamServo.SetDegree(180);
1008     Sleep(1.);
1009     iceCreamServo.SetDegree(145);
1010
1011     driveShaft(-25, 350);
1012
1013     /*
1014      * Return to start button
1015      */
1016
1017     //degrees for QR code orientations (plug into
```

```
    check_heading())
1018    float y_plus_heading, y_minus_heading,
    x_plus_heading, x_minus_heading, final_angle;
1019    y_plus_heading = 180;
1020    y_minus_heading = 0;
1021    x_plus_heading = 90;
1022    x_minus_heading = 270;
1023    final_angle = 45;
1024
1025    //face negative x direction
1026    rightMotor.SetPercent(25);
1027    leftMotor.SetPercent(-25);
1028    check_heading(x_minus_heading);
1029
1030    //move to x~18
1031    rightMotor.SetPercent(25);
1032    leftMotor.SetPercent(25);
1033    check_x_minus(20.5);
1034
1035    //face negative y direction
1036    rightMotor.SetPercent(25);
1037    leftMotor.SetPercent(-25);
1038    check_heading(y_minus_heading);
1039
1040    //move to y=18
1041    rightMotor.SetPercent(25);
1042    leftMotor.SetPercent(25);
1043    check_y_minus(18);
1044
1045    //face button
1046    rightMotor.SetPercent(25);
1047    leftMotor.SetPercent(-25);
1048    check_heading(final_angle);
1049
1050    //hit button
1051    rightMotor.SetPercent(25);
1052    leftMotor.SetPercent(25);
1053
1054    batteryVoltage();
1055
1056    return 0;
1057 }
```