Embedded Application Doxygen Report

Generated by Doxygen 1.9.3

Chapter 1

Embedded Application Coursework Project

Author

Manuel Dogbatse (gwc20rpu@uea.ac.uk) and Joshua Crafton (avr19xvu@uea.ac.uk)

Typical motorcycles aren't very informative and are more likely to be involved in fatal vehicle accidents than cars. Therefore, we designed an RTOS system that uses sensor data to alert the motorcyclist of potential risks whilst driving. The device shows the distance between the motorcycle and potential hazards from the side using ultrasonic sensors, it measures the temperature of the motorcycle's engine using a digital temperature sensor, and the lean angle of the motorcycle is measured and displayed using a 3-axis accelerometer. There is also a buzzer and an LED which activate when the motorcyclist leans too much in a certain direction. This sensor information is then displayed on an STM32F746G-Discovery board. The motorcyclist can also use change the way this information is displayed using the touchscreen, which toggles between the main screen and the settings screen. This settings screen contains buttons that change the units of measurement for the temperature and the distance, as well as the colour palette for the main screen.

For the GLCD display, there are several functions created for the purpose of drawing specific shapes and making the display much simpler to design in the main function. Such a function is the 'drawCircle' function, which uses Bresenham's circle algorithm to calculate the placement of each pixel based off the center coordinates and the radius of the circle. This algorithm works by drawing a foundation of 8 pixels, of which each coordinate is a variation of (centerX +/- distanceX, centerY +/- distanceY) and (centerX +/- distanceY, centerY +/- distanceX). Then the program enters a while loop and draws 8 pixels for every foundation pixel. The decision parameter 'dp' determines the best position for the next 8 pixels to be drawn using the formula dp = dp + (4 $*$ (x - y)) + 10 if dp $>$ 0, or dp = dp + (4 $*$ x) + 6 if dp $<=$ 0. This formula makes the circle look more rounded.

The 3-axis accelerometer, named the MPU6050, uses both accelerometer and gyroscope readings to determine the lean angle of the motorcyclist. In order for the accelerometer to function, it needs to be initialised, and this is done by reading and writing to several registers using the I2C protocol. Then the X,Y and Z values from the 'ACCEL_XOUT_H' and 'GYRO_XOUT_H' registers are read and converted to g and dps respectively. The pitch, roll and yaw of the accelerometer are then calculated using a formula. We require the roll (y-axis) value, so it would be calculated by: 180 $*$ arctan(accelY / sqrt(accelX^2 + accelZ^2)) / pi. The roll value's polarity has to be flipped as the MPU is facing the opposite way. In our case, we are using only the roll value, so the roll value is converted to radians using: radians = angle $*$ (pi / 180), where angle = roll value. The radians value then has 3 $*$ pi / 2 added so that the lean angle line starts in the middle of the display. Then the calculated radians are used to determine the lean angle line's ending point using trigonometric functions: x = radius $*$ cos(radians) + x0, y = radius $*$ sin(radians) + y0, where x0 and y0 are the coordinates for the lean angle line's starting point. This allows the lean angle to be displayed to the motorcyclist.

The rotary encoder works with a 5 pin map, with the first 2 being GND and VCC for the power supply. Then there are 3 output pins: SW, DT, and CLK. The SW is the active low push button, so when the button on the erotary encoder is pressed, a low voltage output is transmitted to the board. The DT output determines the direction of the rotation. This direction can be calculated because the rotary encoder has a common ground pin C, as well as two

contact pins A and B, of which A and B are shifted 90 degrees out of phase with each other. This way, each pin will come into contact with pin C before one another, and through this the direction of rotation can be determined. The CLK output will go through a cycle of going high and then low every time the knob is rotated by one detent. We use this rotary encoder as a replacement for the ultrasonic sensors to show the functionality of the ultrasonic sensor chevron display.

There were a few issues that we ran into while working on our project. Firstly, initialising the timers and configuring them for the ultrasonic and digital temperature sensors was problematic, specifically getting the TIM1_CH1 channel to become active and allow the sensor data to be read. Therefore the ultrasonic sensor display is instead triggered by rotary encoders placed on each side of the board and the temperature display is incremented by the clock to show the functionality of the 'getDigits' function, which separates each digit from a given multiple-digit number.

References: Bresenham's Circle Algorithm: https://www.geeksforgeeks.org/bresenhams-circle-drawing-a
Accelerometer Initialisation and Setup: https://controllerstech.com/how-to-interface-mpu6050-gy-521-
Accelerometer Pitch, Roll and Yaw Calculations: https://engineering.stackexchange.↩
com/questions/3348/calculating-pitch-yaw-and-roll-from-mag-acc-and-gyro-data
Rotary Encoder Functionality: https://lastminuteengineers.com/rotary-encoder-arduino-tutorial/

Chapter 2

File Index

2.1 File List

Here is a list of all documented files with brief descriptions:

Chapter 3

File Documentation

3.1 main.c

```
00001 /*
00002
00003  File               : main.c
00004
00005  Primary Author : Joshua Crafton
00006
00007  Description        : The main c file that configures the timers, the pins, and the GPIO
00008                                  outputs. Then uses all the functions to display a screen showing
00009                                  sensor information for the motorcyle mount system.
00010 */
00011
00012 // Header file for external files
00013 #include "main.h"
00014
00015 #define wait_delay HAL_Delay
00016
00017 //--------MPU Registers--------------------
00018 //
00019 #define MPU6050_ADDR (0x68 « 1) // 0xD0
00020
00021
00022 #define SMPLRT_DIV_REG 0x19
00023 #define GYRO_CONFIG_REG 0x1B
00024 #define ACCEL_CONFIG_REG 0x1C
00025 #define ACCEL_XOUT_H_REG 0x3B
00026 #define TEMP_OUT_H_REG 0x41
00027 #define GYRO_XOUT_H_REG 0x43
00028 #define PWR_MGMT_1_REG 0x6B
00029 #define WHO_AM_I_REG 0x75
00030
00031 //----------------------------------------
00032
00033 #ifdef __RTX
00034 extern uint32_t os_time;
00035 uint32_t HAL_GetTick(void) {
00036     return os_time;
00037 }
00038 #endif
00039
00043 TIM_HandleTypeDef htim2;
00044 I2C_HandleTypeDef hi2c1;
00045
00046 uint16_t colourScheme, temperature, tempUnit, distUnit; // Variables to change UI related
       units/colours
00047 int currentDistLeft = 0; // For remembering how many chevrons are currently appearing
00048 int currentDistRight = 0;
00049 uint16_t distLeft = 0; //Actual distance mesurement
00050 uint16_t distRight = 0;
00051
00052
00053 // Pi to 21 significant figures
00054 const float M_PI = 3.14159265358979323846;
00055
00056 // Accelleromerter Raw Values
00057 int16_t Accel_X_RAW = 0;
00058 int16_t Accel_Y_RAW = 0;
00059 int16_t Accel_Z_RAW = 0;
00060 //Gyroscope Raw Values
```

```
00061 int16_t Gyro_X_RAW = 0;
00062 int16_t Gyro_Y_RAW = 0;
00063 int16_t Gyro_Z_RAW = 0;
00064 //Values after raw to real converstion
00065 float Ax, Ay, Az, Gx, Gy, Gz;
00066 float pitch = 0;
00067 float roll = 0;
00068 float yaw = 0;
00069 // Init position of lean pointer head
00070 int circX = 240;
00071 int circY = 142;
00072
00073 uint32_t colour1;//Background usually
00074 uint32_t colour2;//Foreground usually
00075 uint32_t colour3;//Spare
00076
00077
00082 void SystemClock_Config(void);
00083 static void TIM2_Init(void);
00084 static void GPIO_Init(void);
00085 void Error_Handler(void);
00086 static void I2C1_Init(void);
00087
00088 // Buzzer/LED control
00089 void turnOnBuzzer(){
00090     HAL_GPIO_WritePin(GPIOI, GPIO_PIN_3, GPIO_PIN_SET);
00091 }
00092
00093 void turnOffBuzzer(){
00094     HAL_GPIO_WritePin(GPIOI, GPIO_PIN_3, GPIO_PIN_RESET);
00095 }
00096
00097
00098 //-----------------------START MPU CODE-----------------------------------
00099 // Reference: https://controllerstech.com/how-to-interface-mpu6050-gy-521-with-stm32/
00100 void MPU6050_Init (void)
00101 {
00102     uint8_t check;
00103     uint8_t Data;
00104
00105     // check device ID WHO_AM_I
00106
00107     HAL_I2C_Mem_Read (&hi2c1, MPU6050_ADDR,WHO_AM_I_REG,1, &check, 1, 1000);
00108
00109     if (check == 104)  // 0x68 will be returned by the sensor if everything goes well
00110     {
00111         // power management register 0X6B we should write all 0's to wake the sensor up
00112         Data = 0;
00113         HAL_I2C_Mem_Write(&hi2c1, MPU6050_ADDR, PWR_MGMT_1_REG, 1,&Data, 1, 1000);
00114
00115         // Set DATA RATE of 1KHz by writing SMPLRT_DIV register
00116         Data = 0x07;
00117         HAL_I2C_Mem_Write(&hi2c1, MPU6050_ADDR, SMPLRT_DIV_REG, 1, &Data, 1, 1000);
00118
00119         // Set accelerometer configuration in ACCEL_CONFIG Register
00120         // XA_ST=0,YA_ST=0,ZA_ST=0, FS_SEL=0 ->  2g
00121         Data = 0x00;
00122         HAL_I2C_Mem_Write(&hi2c1, MPU6050_ADDR, ACCEL_CONFIG_REG, 1, &Data, 1, 1000);
00123
00124         // Set Gyroscopic configuration in GYRO_CONFIG Register
00125         // XG_ST=0,YG_ST=0,ZG_ST=0, FS_SEL=0 ->  250 /s
00126         Data = 0x00;
00127         HAL_I2C_Mem_Write(&hi2c1, MPU6050_ADDR, GYRO_CONFIG_REG, 1, &Data, 1, 1000);
00128     }
00129 }
00130
00131 void MPU6050_Read_Accel (void)
00132 {
00133     uint8_t Rec_Data[6];
00134
00135     // Read 6 BYTES of data starting from ACCEL_XOUT_H register
00136
00137     HAL_I2C_Mem_Read (&hi2c1, MPU6050_ADDR, ACCEL_XOUT_H_REG, 1, Rec_Data, 6, 1000);
00138
00139     Accel_X_RAW = (int16_t)(Rec_Data[0] « 8 | Rec_Data [1]);
00140     Accel_Y_RAW = (int16_t)(Rec_Data[2] « 8 | Rec_Data [3]);
00141     Accel_Z_RAW = (int16_t)(Rec_Data[4] « 8 | Rec_Data [5]);
00142
00143     /*** convert the RAW values into acceleration in 'g'
00144          we have to divide according to the Full scale value set in FS_SEL
00145          I have configured FS_SEL = 0. So I am dividing by 16384.0
00146          for more details check ACCEL_CONFIG Register              ****/
00147
00148     Ax = Accel_X_RAW/16384.0;
00149     Ay = Accel_Y_RAW/16384.0;
00150     Az = Accel_Z_RAW/16384.0;
00151 }
```

```
00152
00153
00154 void MPU6050_Read_Gyro (void)
00155 {
00156     uint8_t Rec_Data[6];
00157
00158     // Read 6 BYTES of data starting from GYRO_XOUT_H register
00159
00160     HAL_I2C_Mem_Read (&hi2c1, MPU6050_ADDR, GYRO_XOUT_H_REG, 1, Rec_Data, 6, 1000);
00161
00162     Gyro_X_RAW = (int16_t)(Rec_Data[0] « 8 | Rec_Data [1]);
00163     Gyro_Y_RAW = (int16_t)(Rec_Data[2] « 8 | Rec_Data [3]);
00164     Gyro_Z_RAW = (int16_t)(Rec_Data[4] « 8 | Rec_Data [5]);
00165
00166     /*** convert the RAW values into dps (/s)
00167          we have to divide according to the Full scale value set in FS_SEL
00168          I have configured FS_SEL = 0. So I am dividing by 131.0
00169          for more details check GYRO_CONFIG Register              ****/
00170
00171     Gx = Gyro_X_RAW/131.0;
00172     Gy = Gyro_Y_RAW/131.0;
00173     Gz = Gyro_Z_RAW/131.0;
00174 }
00175
00176 // Calculating the pitch, roll, and yaw values using the accelerometer input data
00177 // Reference:
      https://engineering.stackexchange.com/questions/3348/calculating-pitch-yaw-and-roll-from-mag-acc-and-gyro-data
00178 void convertAcc (void){
00179
00180     pitch = 180 * atan (Ax/sqrt(Ay*Ay + Az*Az))/M_PI;
00181     roll = 180 * atan (Ay/sqrt(Ax*Ax + Az*Az))/M_PI;
00182     yaw = 180 * atan (Az/sqrt(Ax*Ax + Az*Az))/M_PI;
00183     roll *= -1; // The MPU is facing the opposite way to the screen and so the value is required to be
      flipped
00184 }
00185
00186 //Returns the radians value of a degree angle
00187 float toRadians(float angle){
00188     return angle * ( M_PI / 180.0 );
00189 }
00190
00191 // saves the Circumference X and Y, When the MPU is held up in the same orientation as the screen, the
      angle will be the roll value.
00192 void getCircumferenceXY (int x0, int y0, int r, float angle){
00193     float xPos = 0;
00194     float yPos = 0;
00195     // The values lie between -90 and 90 with 0 at the top/bottom middle. Radians start at 0 on the
      right.
00196     //Therefore can be translated by 3PI/2 to rotate 0 to be directly downwards.
00197     float radAngle = toRadians(angle) + (3*M_PI)/2;
00198     // Find the x and y positions with trigonemetry functions for the circumference of a circle
00199     xPos = r * (float)cos(radAngle) + x0;
00200   yPos = r * (float)sin(radAngle) + y0;
00201
00202   circX = (int)xPos;
00203     circY = (int)yPos;
00204
00205     if(angle >= 60 || angle <= -60){
00206         turnOnBuzzer();
00207     }else{
00208         turnOffBuzzer();
00209     }
00210 }
00211 //------------------------END MPU CODE---------------------------------------
00212
00213 // All no moving/changing UI elements are called in the function.
00214 void mainScreen(){
00215
00216     GLCD_ClearScreen();
00217     //Used to ensure the correct colour scheme is setup.
00218     if (colourScheme == 0){
00219         colour1 = GLCD_COLOR_BLACK;
00220         colour2 = GLCD_COLOR_WHITE;
00221     }
00222     else if (colourScheme == 1){
00223         colour1 = GLCD_COLOR_BLACK;
00224         // Cyan
00225         colour2 = 0x07F9;
00226     }
00227     else if (colourScheme == 2){
00228         colour1 = GLCD_COLOR_BLACK;
00229         colour2 = GLCD_COLOR_MAGENTA;
00230     }
00231     else if (colourScheme == 3){
00232         colour1 = GLCD_COLOR_BLACK;
00233         // Red
00234         colour2 = 0xFA20;
```

```
00235     }
00236     //Fill the background colour
00237     fillBackground(colour1);
00238     // Settings Button
00239     drawRectangle(320, 5, 60, 30, colour2);
00240     // Settings Annotation
00241     drawString(327, 11, "SET", colour2, colour1);
00242 
00243     // Temperature Display
00244     drawCircle(240, 71, 71, colour2);
00245 
00246     // Temperature Reading
00247     if(tempUnit == 0){
00248         // F Annotation
00249         drawCircle(233, 88, 4, colour2);
00250         drawString(240, 85, "F", colour2, colour1);
00251     }else{
00252         // C Annotation
00253         drawCircle(233, 88, 4, colour2);
00254         drawString(240, 85, "C", colour2, colour1);
00255     }
00256 
00257     // Gyrometer Display
00258     drawCircle(240, 272, 130, colour2);
00259 
00260     // Left Ultrasonic Display
00261     displayDisChevronsLeft(0, colour1, colour2);
00262 
00263     if(distUnit == 0){
00264         // Left Ultrasonic Reading
00265         drawString(118, 125, ".", colour2, colour1);
00266         drawString(164, 125, "yd", colour2, colour1);
00267     }else{
00268         // Left Ultrasonic Reading
00269         drawString(118, 125, ".", colour2, colour1);
00270         drawString(164, 125, "m", colour2, colour1);
00271     }
00272 
00273     // Right Ultrasonic Display
00274     displayDisChevronsRight(0, colour1, colour2);
00275 
00276     if(distUnit == 0){
00277         // Right Ultrasonic Reading
00278     drawString(325, 125, ".", colour2, colour1);
00279     drawString(371, 125, "yd", colour2, colour1);
00280     }else{
00281         // Right Ultrasonic Reading
00282     drawString(325, 125, ".", colour2, colour1);
00283     drawString(371, 125, "m", colour2, colour1);
00284     }
00285 }
00286 
00287 void settingsScreen(){
00288 
00289     int touchValue;
00290 
00291     TOUCH_STATE tsc_state;
00292 
00293     GLCD_ClearScreen();
00294 
00295     // Setting up colour schemes
00296     if (colourScheme == 0){
00297         colour1 = GLCD_COLOR_BLACK;
00298         colour2 = GLCD_COLOR_WHITE;
00299     }
00300     else if (colourScheme == 1){
00301         colour1 = GLCD_COLOR_BLACK;
00302         colour2 = 0x07F9;
00303     }
00304     else if (colourScheme == 2){
00305         colour1 = GLCD_COLOR_MAGENTA;
00306         colour2 = GLCD_COLOR_WHITE;
00307     }
00308     else if (colourScheme == 3){
00309         colour1 = GLCD_COLOR_BLACK;
00310         colour2 = 0xFA20;
00311     }
00312 
00313     // Settings Title
00314     drawRectangle(160, 0, 160, 35, GLCD_COLOR_BLACK);
00315     drawString(177, 8, "SETTINGS", GLCD_COLOR_BLACK, GLCD_COLOR_WHITE);
00316 
00317     // Back Button
00318     drawRectangle(403, 5, 70, 30, GLCD_COLOR_BLACK);
00319     // Back Annotation
00320     drawString(406, 11, "BACK", GLCD_COLOR_BLACK, GLCD_COLOR_WHITE);
00321 
```

```
00322     // Unit Measurement Select Display
00323     drawRectangle(5, 61, 213, 204, GLCD_COLOR_BLACK);
00324     drawString(72, 65, "Units", GLCD_COLOR_BLACK, GLCD_COLOR_WHITE);
00325
00326     // Temperature Measurement Select
00327     drawString(24, 101, "Temperature", GLCD_COLOR_BLACK, GLCD_COLOR_WHITE);
00328     // C Button
00329     drawRectangle(25, 135, 70, 30, GLCD_COLOR_BLACK);
00330     drawCircle(50, 143, 4, GLCD_COLOR_BLACK);
00331     drawString(57, 140, "C", GLCD_COLOR_BLACK, GLCD_COLOR_WHITE);
00332     // F Button
00333     drawRectangle(131, 135, 70, 30, GLCD_COLOR_BLACK);
00334     drawCircle(156, 143, 4, GLCD_COLOR_BLACK);
00335     drawString(163, 140, "F", GLCD_COLOR_BLACK, GLCD_COLOR_WHITE);
00336
00337     // Distance Measurement Select
00338     drawString(52, 180, "Distance", GLCD_COLOR_BLACK, GLCD_COLOR_WHITE);
00339     // m button
00340     drawRectangle(25, 215, 70, 30, GLCD_COLOR_BLACK);
00341     drawString(53, 220, "m", GLCD_COLOR_BLACK, GLCD_COLOR_WHITE);
00342     // yd button
00343     drawRectangle(131, 215, 70, 30, GLCD_COLOR_BLACK);
00344     drawString(150, 218, "yd", GLCD_COLOR_BLACK, GLCD_COLOR_WHITE);
00345
00346     // Colour Palette Select
00347     drawRectangle(260, 61, 213, 204, GLCD_COLOR_BLACK);
00348     drawString(322, 65, "Colour", GLCD_COLOR_BLACK, GLCD_COLOR_WHITE);
00349
00350     // Palette Displays
00351     drawPalette(295, 120, 45);
00352     fillPalette(295, 120, 45, GLCD_COLOR_WHITE);
00353     drawString(293, 88, "Day", GLCD_COLOR_BLACK, GLCD_COLOR_WHITE);
00354     drawPalette(396, 120, 45);
00355     fillPalette(396, 120, 45, 0x07F9);
00356     drawString(381, 88, "Night", GLCD_COLOR_BLACK, GLCD_COLOR_WHITE);
00357     drawPalette(295, 208, 45);
00358     fillPalette(295, 208, 45, GLCD_COLOR_MAGENTA);
00359     drawString(280, 175, "Funky", GLCD_COLOR_BLACK, GLCD_COLOR_WHITE);
00360     drawPalette(396, 208, 45);
00361     fillPalette(396, 208, 45, 0xFA20);
00362     drawString(389, 175, "Evil", GLCD_COLOR_BLACK, GLCD_COLOR_WHITE);
00363
00364     // Highlight the bounds of the current setting box
00365     highlightTempUnit(tempUnit);
00366     highlightDistUnit(distUnit);
00367     highlightColour(colourScheme);
00368
00369     // Required to stay on this screen until the back button is pressed.
00370     for(;;)
00371     {
00372         Touch_GetState(&tsc_state);
00373         if (tsc_state.pressed)
00374         {
00375             touchValue = checkCoordsSettings(tsc_state.x, tsc_state.y);
00376             if (touchValue == -1)
00377             {
00378                 mainScreen();
00379                 break;
00380             }
00381         }
00382         else if (touchValue == 0)
00383             tempUnit = 0;
00384         else if (touchValue == 1)
00385                 tempUnit = 1;
00386         else if (touchValue == 10)
00387                 distUnit = 0;
00388         else if (touchValue == 11)
00389                 distUnit = 1;
00390         else if (touchValue == 20)
00391                 colourScheme = 0;
00392         else if (touchValue == 21)
00393                 colourScheme = 1;
00394         else if(touchValue == 22)
00395                 colourScheme = 2;
00396         else if(touchValue == 23)
00397                 colourScheme = 3;
00398     }
00399 }
00400
00401 int main(void){
00402     int touchValue;
00403     char buf[3];
00404     int* digits;
00405     char tempBuffer[3][128], lUltBuffer[4][128], rUltBuffer[4][128];
00406     int prev = 0;
00407     int loop = 0;
00408
```

```
00409     TOUCH_STATE tsc_state;
00410
00411     //-------------INIT START--------------------
00412     HAL_Init(); //Init Hardware Abstraction Layer
00413     SystemClock_Config(); //Config Clocks
00414     GPIO_Init();
00415     I2C1_Init();
00416     TIM2_Init();
00417     __HAL_RCC_TIM2_CLK_ENABLE();
00418
00419     Touch_Initialize();
00420     GLCD_Initialize(); //Init GLCD
00421     GLCD_ClearScreen();
00422     GLCD_SetFont(&GLCD_Font_16x24);
00423
00424     MPU6050_Init();
00425     //-------------INIT END----------------------
00426
00427     temperature = 0;
00428     // C = 1, F = 0
00429     tempUnit = 1;
00430     // m = 1, yd = 0
00431     distUnit = 1;
00432
00433     colourScheme = 0;
00434
00435     mainScreen();
00436     HAL_Delay(1000);
00437     for(;;)
00438     {
00439         //call MPU read functions
00440         MPU6050_Read_Accel();
00441         MPU6050_Read_Gyro();
00442
00443         //Check if the user want to go to the settings menu
00444         Touch_GetState(&tsc_state);
00445         if (tsc_state.pressed)
00446         {
00447             touchValue = checkCoordsMain(tsc_state.x, tsc_state.y);
00448             if (touchValue)
00449             {
00450                 settingsScreen();
00451             }
00452         }
00453
00454         //------------Start MPU Calculations---------
00455         convertAcc();
00456
00457         drawDiagonalLine(240, 272, circX, circY, colour1);
00458         getCircumferenceXY(240, 272, 128, roll);
00459         drawDiagonalLine(240, 272, circX, circY,colour2);
00460
00461         //-----------------END MPU Calcs--------------
00462
00463         //-------------Distance------------------
00464         // Determine how many chevrons to place on the left or right
00465         if(distLeft > 0 && distLeft <= 5 && currentDistLeft != 5){
00466             displayDisChevronsLeft(5, colour1, colour2);
00467             currentDistLeft = 5;
00468         }else if(distLeft > 5 && distLeft <= 10 && currentDistLeft != 4){
00469             displayDisChevronsLeft(4, colour1, colour2);
00470             currentDistLeft = 4;
00471         }else if(distLeft > 10 && distLeft <= 15 && currentDistLeft != 3){
00472             displayDisChevronsLeft(3, colour1, colour2);
00473             currentDistLeft = 3;
00474         }else if(distLeft > 15 && distLeft <= 20 && currentDistLeft != 2){
00475             displayDisChevronsLeft(2, colour1, colour2);
00476             currentDistLeft = 2;
00477         }else if(distLeft > 20 && distLeft <= 25 && currentDistLeft != 1){
00478             displayDisChevronsLeft(1, colour1, colour2);
00479             currentDistLeft = 1;
00480         }else if(distLeft > 30){
00481             displayDisChevronsLeft(0, colour1, colour2);
00482             distLeft = 0;
00483         }
00484
00485         if(distRight > 0 && distRight <= 5 && currentDistRight != 5){
00486             displayDisChevronsRight(5, colour1, colour2);
00487             currentDistRight = 5;
00488         }else if(distRight > 5 && distRight <= 10 && currentDistRight != 4){
00489             displayDisChevronsRight(4, colour1, colour2);
00490             currentDistRight = 4;
00491         }else if(distRight > 10 && distRight <= 15 && currentDistRight != 3){
00492             displayDisChevronsRight(3, colour1, colour2);
00493             currentDistRight = 3;
00494         }else if(distRight > 15 && distRight <= 20 && currentDistRight != 2){
00495             displayDisChevronsRight(2, colour1, colour2);
```

```
00496             currentDistRight = 2;
00497         }else if(distRight > 20 && distRight <= 25 && currentDistRight != 1){
00498             displayDisChevronsRight(1, colour1, colour2);
00499             currentDistRight = 1;
00500         }else if(distRight > 30){
00501             displayDisChevronsRight(0, colour1, colour2);
00502             distRight = 0;
00503         }
00504 
00505 
00506         // If the rotary encoders button is pressed down then allow for the rotating function to be
      checked
00507         // otherwise pass straight through
00508         for(;;) //right side
00509         {
00510             //Read the button pin
00511             if(HAL_GPIO_ReadPin(GPIOI, GPIO_PIN_0) == GPIO_PIN_RESET)
00512             {
00513                 //run check function and return 1 higher or lower dependent on direction spun
00514                 distLeft = checkEncoderLeft(distLeft);
00515                 sprintf(buf, "%2d", distLeft);
00516                 if(distUnit == 0)
00517                 {
00518                     // Left Ultrasonic Reading
00519                     drawString(118, 125, buf, colour2, colour1);
00520                     drawString(164, 125, "yd", colour2, colour1);
00521                 }
00522                 else
00523                 {
00524                     // Left Ultrasonic Reading
00525                     drawString(118, 125, buf, colour2, colour1);
00526                     drawString(164, 125, "m", colour2, colour1);
00527                 }
00528             }
00529             else
00530             {
00531                 break;
00532             }
00533         }
00534 
00535         for(;;){//left side
00536             if(HAL_GPIO_ReadPin(GPIOB, GPIO_PIN_4) == GPIO_PIN_RESET){
00537             distRight = checkEncoderRight(distRight);
00538 
00539             sprintf(buf, "%2d", distRight);
00540             if(distUnit == 0){
00541             // Right Ultrasonic Reading
00542             drawString(325, 125, buf, colour2, colour1);
00543             drawString(371, 125, "yd", colour2, colour1);
00544             }else{
00545                 // Right Ultrasonic Reading
00546             drawString(325, 125, buf, colour2, colour1);
00547             drawString(371, 125, "m", colour2, colour1);
00548             }
00549 
00550             }else{break;}
00551         }
00552         //-----------------end--------------------
00553 
00554         //-------------Temperature---------------
00555 
00556                 digits = getDigits(temperature);
00557                 sprintf(tempBuffer[0], "%d", digits[0]);
00558                 sprintf(tempBuffer[1], "%d", digits[1]);
00559                 sprintf(tempBuffer[2], "%d", digits[2]);
00560 
00561                 drawString(220, 50, tempBuffer[2], colour2, colour1);
00562                 drawString(235, 50, tempBuffer[1], colour2, colour1);
00563                 drawString(250, 50, tempBuffer[0], colour2, colour1);
00564 
00565         //-----------------End------------------
00566 
00567         temperature += 1;
00568 
00569         if (temperature == 1000)
00570             temperature = 0;
00571 
00572         HAL_Delay(200);
00573     }
00574 }
00575 
00576 void SystemClock_Config(void)
00577 {
00578             RCC_OscInitTypeDef RCC_OscInitStruct;
00579             RCC_ClkInitTypeDef RCC_ClkInitStruct;
00580             /* Enable Power Control clock */
00581             __HAL_RCC_PWR_CLK_ENABLE();
```

```
00582             /* The voltage scaling allows optimizing the power
00583             consumption when the device is clocked below the
00584             maximum system frequency. */
00585             __HAL_PWR_VOLTAGESCALING_CONFIG(PWR_REGULATOR_VOLTAGE_SCALE1);
00586             /* Enable HSE Oscillator and activate PLL
00587             with HSE as source */
00588             RCC_OscInitStruct.OscillatorType = RCC_OSCILLATORTYPE_HSE;
00589             RCC_OscInitStruct.HSEState = RCC_HSE_ON;
00590             RCC_OscInitStruct.PLL.PLLState = RCC_PLL_ON;
00591             RCC_OscInitStruct.PLL.PLLSource = RCC_PLLSOURCE_HSE;
00592             RCC_OscInitStruct.PLL.PLLM = 25;
00593             RCC_OscInitStruct.PLL.PLLN = 336;
00594             RCC_OscInitStruct.PLL.PLLP = RCC_PLLP_DIV2;
00595             RCC_OscInitStruct.PLL.PLLQ = 7;
00596             HAL_RCC_OscConfig(&RCC_OscInitStruct);
00597             /* Select PLL as system clock source and configure
00598             the HCLK, PCLK1 and PCLK2 clocks dividers */
00599             RCC_ClkInitStruct.ClockType = RCC_CLOCKTYPE_SYSCLK |
00600             RCC_CLOCKTYPE_PCLK1 | RCC_CLOCKTYPE_PCLK2;
00601             RCC_ClkInitStruct.SYSCLKSource = RCC_SYSCLKSOURCE_PLLCLK;
00602             RCC_ClkInitStruct.AHBCLKDivider = RCC_SYSCLK_DIV1;
00603             RCC_ClkInitStruct.APB1CLKDivider = RCC_HCLK_DIV4;
00604             RCC_ClkInitStruct.APB2CLKDivider = RCC_HCLK_DIV2;
00605             HAL_RCC_ClockConfig(&RCC_ClkInitStruct, FLASH_LATENCY_5);
00606 }
00607
00608 static void I2C1_Init(void)
00609 {
00610   /* USER CODE BEGIN I2C1_Init 0 */
00611     __HAL_RCC_I2C1_CLK_ENABLE();
00612
00613   __HAL_RCC_I2C1_FORCE_RESET();
00614   HAL_Delay(2);
00615   __HAL_RCC_I2C1_RELEASE_RESET();
00616   /* USER CODE END I2C1_Init 0 */
00617
00618   /* USER CODE BEGIN I2C1_Init 1 */
00619
00620   /* USER CODE END I2C1_Init 1 */
00621   hi2c1.Instance = I2C1;
00622   hi2c1.Init.Timing = 0x00808CD2;
00623   hi2c1.Init.OwnAddress1 = 0;
00624   hi2c1.Init.AddressingMode = I2C_ADDRESSINGMODE_7BIT;
00625   hi2c1.Init.DualAddressMode = I2C_DUALADDRESS_DISABLE;
00626   hi2c1.Init.OwnAddress2 = 0;
00627   hi2c1.Init.OwnAddress2Masks = I2C_OA2_NOMASK;
00628   hi2c1.Init.GeneralCallMode = I2C_GENERALCALL_DISABLE;
00629   hi2c1.Init.NoStretchMode = I2C_NOSTRETCH_DISABLE;
00630   if (HAL_I2C_Init(&hi2c1) != HAL_OK)
00631   {
00632     Error_Handler();
00633   }
00634
00637   if (HAL_I2CEx_ConfigAnalogFilter(&hi2c1, I2C_ANALOGFILTER_ENABLE) != HAL_OK)
00638   {
00639     Error_Handler();
00640   }
00641
00644   if (HAL_I2CEx_ConfigDigitalFilter(&hi2c1, 0) != HAL_OK)
00645   {
00646     Error_Handler();
00647   }
00648   /* USER CODE BEGIN I2C1_Init 2 */
00649
00650
00651   /* USER CODE END I2C1_Init 2 */
00652
00653
00654 }
00655
00656 static void TIM2_Init(void)
00657 {
00658
00659   /* USER CODE BEGIN TIM2_Init 0 */
00660
00661   /* USER CODE END TIM2_Init 0 */
00662
00663   TIM_MasterConfigTypeDef sMasterConfig = {0};
00664
00665   /* USER CODE BEGIN TIM2_Init 1 */
00666
00667   /* USER CODE END TIM2_Init 1 */
00668   htim2.Instance = TIM2;
00669   htim2.Init.Prescaler = 32000;
00670   htim2.Init.CounterMode = TIM_COUNTERMODE_UP;
00671   htim2.Init.Period = 1;
00672   htim2.Init.AutoReloadPreload = TIM_AUTORELOAD_PRELOAD_DISABLE;
```

```
00673   if (HAL_TIM_Base_Init(&htim2) != HAL_OK)
00674   {
00675     Error_Handler();
00676   }
00677   sMasterConfig.MasterOutputTrigger = TIM_TRGO_RESET;
00678   sMasterConfig.MasterSlaveMode = TIM_MASTERSLAVEMODE_DISABLE;
00679   if (HAL_TIMEx_MasterConfigSynchronization(&htim2, &sMasterConfig) != HAL_OK)
00680   {
00681     Error_Handler();
00682   }
00683   /* USER CODE BEGIN TIM2_Init 2 */
00684
00685   /* USER CODE END TIM2_Init 2 */
00686
00687 }
00688
00689 static void GPIO_Init(void)
00690 {
00691     GPIO_InitTypeDef GPIO_InitStruct;
00692
00693   /* GPIO Ports Clock Enable */
00694   __HAL_RCC_GPIOB_CLK_ENABLE();
00695   __HAL_RCC_GPIOA_CLK_ENABLE();
00696     __HAL_RCC_GPIOC_CLK_ENABLE();
00697   __HAL_RCC_GPIOH_CLK_ENABLE();
00698     __HAL_RCC_GPIOG_CLK_ENABLE();
00699     __HAL_RCC_GPIOI_CLK_ENABLE();
00700
00701
00702     // MPU GPIO
00703     GPIO_InitStruct.Pin = GPIO_PIN_8 | GPIO_PIN_9;
00704     GPIO_InitStruct.Mode = GPIO_MODE_AF_OD; // alternate function - open drain
00705     GPIO_InitStruct.Pull = GPIO_PULLUP;
00706     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_VERY_HIGH;
00707     GPIO_InitStruct.Alternate = GPIO_AF4_I2C1;
00708
00709     HAL_GPIO_Init(GPIOB, &GPIO_InitStruct);
00710
00711   // LED_Buzzer GPIO
00712     GPIO_InitStruct.Pin = GPIO_PIN_3;
00713     GPIO_InitStruct.Mode = GPIO_MODE_OUTPUT_PP;
00714     GPIO_InitStruct.Pull = GPIO_NOPULL;
00715     GPIO_InitStruct.Speed = GPIO_SPEED_FREQ_LOW;
00716     GPIO_InitStruct.Alternate = NULL;
00717
00718     HAL_GPIO_Init(GPIOI, &GPIO_InitStruct);
00719
00720     HAL_GPIO_WritePin(GPIOB, GPIO_PIN_4, GPIO_PIN_RESET);
00721
00722
00723   //Encoder GPIO
00724     GPIO_InitStruct.Pin = GPIO_PIN_6 | GPIO_PIN_7;
00725     GPIO_InitStruct.Mode = GPIO_MODE_INPUT;
00726     GPIO_InitStruct.Pull = GPIO_NOPULL;
00727     GPIO_InitStruct.Speed = NULL;
00728     GPIO_InitStruct.Alternate = NULL;
00729
00730     HAL_GPIO_Init(GPIOC, &GPIO_InitStruct);
00731     HAL_GPIO_Init(GPIOG, &GPIO_InitStruct);
00732 }
00733
00734 void Error_Handler(void)
00735 {
00736   /* USER CODE BEGIN Error_Handler_Debug */
00737   /* User can add his own implementation to report the HAL error return state */
00738
00739   /* USER CODE END Error_Handler_Debug */
00740 }
00741
00742 #ifdef  USE_FULL_ASSERT
00750 void assert_failed(uint8_t *file, uint32_t line)
00751 {
00752   /* USER CODE BEGIN 6 */
00753   /* User can add his own implementation to report the file name and line number,
00754      ex: printf("Wrong parameters value: file %s on line %d\r\n", file, line) */
00755   /* USER CODE END 6 */
00756 }
00757 #endif /* USE_FULL_ASSERT */
00758
00759
00760
```

3.2 main.h

```
00001 /*
00002
00003  File               : main.h
00004
00005  Primary Author : Joshua Crafton
00006
00007  Description        : The header file that links all the separate source and header
00008                                  files in this project.
00009
00010 */
00011
00012 #ifndef __MAIN_H
00013 #define __MAIN_H
00014
00015 #include <stdio.h>
00016 #include "stm32f7xx_hal.h"
00017 #include "GLCD_Config.h"
00018 #include "Board_GLCD.h"
00019 #include "Board_Touch.h"
00020 #include <string.h>
00021 #include <stdlib.h>
00022 #include <math.h>
00023
00024 #include "rotary_encoder.h"
00025 #include "sensor_ui.h"
00026
00027 extern GLCD_FONT GLCD_Font_6x8;
00028 extern GLCD_FONT GLCD_Font_16x24;
00029
00030 #endif /* __MAIN_H */
```

3.3 rotary_encoder.h

```
00001 /*
00002
00003  File               : rotary_encoder.h
00004
00005  Primary Author : Joshua Crafton
00006
00007  Description        : The header file with functions that read the rotary encoders.
00008
00009 */
00010
00011 #ifndef __ROTARY_ENCODER_H
00012 #define __ROTARY_ENCODER_H
00013
00014 #include "main.h"
00015
00016
00017
00018 #define GPIO_PORT_RIGHT GPIOC
00019 #define GPIO_PORT_LEFT GPIOG
00020 #define OUTA_PIN_RIGHT GPIO_PIN_6
00021 #define OUTB_PIN_RIGHT GPIO_PIN_7
00022 #define OUTA_PIN_LEFT GPIO_PIN_7
00023 #define OUTB_PIN_LEFT GPIO_PIN_6
00024
00025 uint16_t checkEncoderLeft(uint16_t);
00026 uint16_t checkEncoderRight(uint16_t);
00027
00028
00029
00030
00031 uint16_t checkEncoderLeft(uint16_t counter){
00032
00033     if (HAL_GPIO_ReadPin(GPIO_PORT_LEFT, OUTA_PIN_LEFT) == GPIO_PIN_RESET)  // If the OUTA is RESET
00034         {
00035             if (HAL_GPIO_ReadPin(GPIO_PORT_LEFT, OUTB_PIN_LEFT) == GPIO_PIN_RESET)  // If OUTB is also
      reset... CCK
00036             {
00037                 while (HAL_GPIO_ReadPin(GPIO_PORT_LEFT, OUTB_PIN_LEFT) == GPIO_PIN_RESET);  // wait
      for the OUTB to go high
00038                 counter--;
00039                 while (HAL_GPIO_ReadPin(GPIO_PORT_LEFT, OUTA_PIN_LEFT) == GPIO_PIN_RESET);  // wait
      for the OUTA to go high
00040                 HAL_Delay (10);  // wait for some more time
00041             }
00042
00043             else if (HAL_GPIO_ReadPin(GPIO_PORT_LEFT, OUTB_PIN_LEFT) == GPIO_PIN_SET)  // If OUTB is
      also set
00044             {
```

```
00045                 while (HAL_GPIO_ReadPin(GPIO_PORT_LEFT, OUTB_PIN_LEFT) == GPIO_PIN_SET);  // wait for
      the OUTB to go LOW.. CK
00046                 counter++;
00047                 while (HAL_GPIO_ReadPin(GPIO_PORT_LEFT, OUTA_PIN_LEFT) == GPIO_PIN_RESET);  // wait
      for the OUTA to go high
00048                 while (HAL_GPIO_ReadPin(GPIO_PORT_LEFT, OUTB_PIN_LEFT) == GPIO_PIN_RESET);  // wait
      for the OUTB to go high
00049                 HAL_Delay (10);  // wait for some more time
00050             }
00051
00052             if (counter<0) counter = 0;
00053             if (counter>30) counter = 30;
00054
00055
00056
00057     }
00058
00059     return counter;
00060 }
00061
00062 uint16_t checkEncoderRight(uint16_t counter){
00063     if (HAL_GPIO_ReadPin(GPIO_PORT_RIGHT, OUTA_PIN_RIGHT) == GPIO_PIN_RESET)  // If the OUTA is RESET
00064         {
00065             if (HAL_GPIO_ReadPin(GPIO_PORT_RIGHT, OUTB_PIN_RIGHT) == GPIO_PIN_RESET)  // If OUTB is
      also reset... CCK
00066             {
00067                 while (HAL_GPIO_ReadPin(GPIO_PORT_RIGHT, OUTB_PIN_RIGHT) == GPIO_PIN_RESET);  // wait
      for the OUTB to go high
00068                 counter--;
00069                 while (HAL_GPIO_ReadPin(GPIO_PORT_RIGHT, OUTA_PIN_RIGHT) == GPIO_PIN_RESET);  // wait
      for the OUTA to go high
00070                 HAL_Delay (10);  // wait for some more time
00071             }
00072
00073             else if (HAL_GPIO_ReadPin(GPIO_PORT_RIGHT, OUTB_PIN_RIGHT) == GPIO_PIN_SET)  // If OUTB is
      also set
00074             {
00075                 while (HAL_GPIO_ReadPin(GPIO_PORT_RIGHT, OUTB_PIN_RIGHT) == GPIO_PIN_SET);  // wait
      for the OUTB to go LOW.. CK
00076                 counter++;
00077                 while (HAL_GPIO_ReadPin(GPIO_PORT_RIGHT, OUTA_PIN_RIGHT) == GPIO_PIN_RESET);  // wait
      for the OUTA to go high
00078                 while (HAL_GPIO_ReadPin(GPIO_PORT_RIGHT, OUTB_PIN_RIGHT) == GPIO_PIN_RESET);  // wait
      for the OUTB to go high
00079                 HAL_Delay (10);  // wait for some more time
00080             }
00081
00082
00083
00084             if (counter<0) counter = 0;
00085             if (counter>30) counter = 30;
00086
00087
00088         }
00089
00090         return counter;
00091 }
00092
00093
00094
00095 #endif
```

3.4 sensor_ui.h

```
00001 /*
00002
00003  File               : sensor_ui.h
00004
00005  Primary Author : Manuel Dogbatse
00006
00007  Description        : The header file that defines all the GLCD display and
00008                                     touchscreen functionalities.
00009
00010 */
00011
00012 #ifndef __SENSOR_UI_H
00013 #define __SENSOR_UI_H
00014
00015 #include "main.h"
00016
00017 void wait(int delay)
00018 {
00019     unsigned int i;
```

```
00020
00021     // wait for specified time
00022     for(i = delay; i > 0; i--);
00023 }
00024
00025 // Gets a 4-digit number and returns an array of each digit
00026 int* getDigits(int num)
00027 {
00028     static int digits[4];
00029     int i;
00030     for (i = 0; i < 4; i++)
00031     {
00032         // Mod operator insures the exact digit is recorded
00033         digits[i] = num % 10;
00034         num = num / 10;
00035     }
00036     return digits;
00037 }
00038
00039 // Function to remove the need to change colours in separate command
00040 void drawRectangle(int x, int y, int dx, int dy, uint32_t colour)
00041 {
00042     GLCD_SetForegroundColor(colour);
00043     GLCD_DrawRectangle(x, y, dx, dy);
00044     GLCD_DrawPixel(x+dx, y+dy);
00045     GLCD_SetForegroundColor(GLCD_COLOR_BLACK);
00046 }
00047
00048 // Function to remove the need to change foreground and background colours in separate command
00049 void drawString(int x, int y, char buffer[128], uint32_t foreColour, uint32_t backColour)
00050 {
00051     GLCD_SetForegroundColor(foreColour);
00052     GLCD_SetBackgroundColor(backColour);
00053     GLCD_DrawString(x, y, buffer);
00054     GLCD_SetForegroundColor(GLCD_COLOR_BLACK);
00055     GLCD_SetBackgroundColor(GLCD_COLOR_WHITE);
00056 }
00057
00058 // Draws the foundation of a circle using the center value and radius values
00059 void drawCircleFoundation(int centerX, int centerY, int distX, int distY)
00060 {
00061     GLCD_DrawPixel(centerX - distX, centerY - distY);
00062     GLCD_DrawPixel(centerX + distX, centerY - distY);
00063     GLCD_DrawPixel(centerX - distX, centerY + distY);
00064     GLCD_DrawPixel(centerX + distX, centerY + distY);
00065     GLCD_DrawPixel(centerX - distY, centerY - distX);
00066     GLCD_DrawPixel(centerX + distY, centerY - distX);
00067     GLCD_DrawPixel(centerX - distY, centerY + distX);
00068     GLCD_DrawPixel(centerX + distY, centerY + distX);
00069 }
00070
00071 // Drawing circles for the temperature and gyrometer displays
00072 void drawCircle(int centerX, int centerY, int radius, uint32_t colour)
00073 {
00074     // Drawing circle using Bresenham's circle algorithm
00075 // Reference = https://www.geeksforgeeks.org/bresenhams-circle-drawing-algorithm/
00076     int x = 0, y = radius, dp = 3 - (2 * radius);
00077     GLCD_SetForegroundColor(colour);
00078     drawCircleFoundation(centerX, centerY, x, y);
00079     while (y >= x)
00080     {
00081         // For every pixel drawn in 'drawCircleFoundation', 8 pixels will be drawn
00082         x++;
00083
00084         // Checks decision parameter and correspondingly updates d, x and y
00085         if (dp > 0)
00086         {
00087             y--;
00088             dp = dp + (4 * (x - y)) + 10;
00089         }
00090         else
00091         {
00092             dp = dp + (4 * x) + 6;
00093         }
00094             drawCircleFoundation(centerX, centerY, x, y);
00095     }
00096     GLCD_SetForegroundColor(GLCD_COLOR_BLACK);
00097 }
00098
00099 // Drawing a diagonal line for the chevrons and gyrometer arrow
00100 void drawDiagonalLineLow(int x0, int y0, int x1, int y1)
00101 {
00102     // Drawing a line using the Bresenham's line algorithm
00103     // Reference: https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm
00104     int dx, dy, yi, D, x, y;
00105     dx = x1 - x0;
00106     dy = y1 - y0;
```

```
00107     yi = 1;
00108     if (dy < 0)
00109     {
00110         yi = -1;
00111         dy = -dy;
00112     }
00113
00114     D = (2 * dy) - dx;
00115     y = y0;
00116
00117     for (x = x0; x < x1; x++)
00118     {
00119         GLCD_DrawPixel(x, y);
00120         if (D > 0)
00121         {
00122             y = y + yi;
00123             D = D + (2 * (dy - dx));
00124         }
00125         else
00126         {
00127             D = D + 2*dy;
00128         }
00129     }
00130 }
00131
00132 void drawDiagonalLineHigh(int x0, int y0, int x1, int y1)
00133 {
00134     // Drawing a line using the Bresenham's line algorithm
00135     // Reference: https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm
00136     int dx, dy, xi, D, x, y;
00137     dx = x1 - x0;
00138     dy = y1 - y0;
00139     xi = 1;
00140     if (dx < 0)
00141     {
00142         xi = -1;
00143         dx = -dx;
00144     }
00145     D = (2 * dx) - dy;
00146     x = x0;
00147
00148     for (y = y0; y < y1; y++)
00149     {
00150         GLCD_DrawPixel(x, y);
00151         if (D > 0)
00152         {
00153             x = x + xi;
00154             D = D + (2 * (dx - dy));
00155         }
00156         else
00157         {
00158             D = D + 2*dx;
00159         }
00160     }
00161 }
00162
00163 void drawDiagonalLine(int x0, int y0, int x1, int y1, uint32_t colour)
00164 {
00165     GLCD_SetForegroundColor(colour);
00166     // These statements insure that the correct variation of the Bresenham's
00167     // line algorithm is used for given starting and ending points
00168     if (abs(y1 - y0) < abs(x1 - x0))
00169     {
00170         if (x0 > x1)
00171         {
00172             drawDiagonalLineLow(x1, y1, x0, y0);
00173         }
00174         else
00175         {
00176             drawDiagonalLineLow(x0, y0, x1, y1);
00177         }
00178     }
00179     else
00180     {
00181         if (y0 > y1)
00182         {
00183             drawDiagonalLineHigh(x1, y1, x0, y0);
00184         }
00185         else
00186         {
00187             drawDiagonalLineHigh(x0, y0, x1, y1);
00188         }
00189     }
00190     GLCD_SetForegroundColor(GLCD_COLOR_BLACK);
00191 }
00192
00193 // Draws a chevron relative to the starting x value.
```

```
00194 // This is used for the ultrasound sensors
00195 void drawChevron(int x, bool isReverse, uint32_t colour)
00196 {
00197     GLCD_SetForegroundColor(colour);
00198     // Ensures that the chevrons are drawn in the correct direction
00199     if (isReverse)
00200     {
00201         drawDiagonalLine(478-x, 136, 471-x, 0, colour);
00202         drawDiagonalLine(478-x, 136, 471-x, 272, colour);
00203         GLCD_DrawHLine(461-x, 0, 10);
00204         GLCD_DrawHLine(461-x, 270, 10);
00205         drawDiagonalLine(468-x, 136, 461-x, 0, colour);
00206         drawDiagonalLine(468-x, 136, 461-x, 272, colour);
00207     }
00208     else
00209     {
00210         drawDiagonalLine(x, 136, x+7, 0, colour);
00211         drawDiagonalLine(x, 136, x+7, 272, colour);
00212         GLCD_DrawHLine(x+7, 1, 10);
00213         GLCD_DrawHLine(x+7, 271, 10);
00214         drawDiagonalLine(x+10, 136, x+17, 0, colour);
00215         drawDiagonalLine(x+10, 136, x+17, 272, colour);
00216     }
00217     GLCD_SetForegroundColor(GLCD_COLOR_BLACK);
00218 }
00219
00220 // Fill the background a given colour
00221 void fillBackground(uint32_t colour)
00222 {
00223     int i, j;
00224     GLCD_SetForegroundColor(colour);
00225     for (i = 0; i < 479; i++)
00226     {
00227         for (j = 0; j < 271; j++)
00228         {
00229             GLCD_DrawPixel(i, j);
00230         }
00231     }
00232     GLCD_SetForegroundColor(GLCD_COLOR_BLACK);
00233 }
00234
00235 // Fill a rectangle a given colour
00236 void fillRectangle(int x, int y, int dx, int dy, uint32_t colour)
00237 {
00238     int i, j;
00239     GLCD_SetForegroundColor(colour);
00240     for (i = x + 1; i < x + dx; i++)
00241     {
00242         for (j = y + 1; j < y + dy; j++)
00243         {
00244             GLCD_DrawPixel(i, j);
00245         }
00246     }
00247     GLCD_SetForegroundColor(GLCD_COLOR_BLACK);
00248 }
00249
00250 // Fill a chevron with a given colour
00251 void fillChevron(int x, bool isReverse, uint32_t colour)
00252 {
00253     int i;
00254     GLCD_SetForegroundColor(colour);
00255     if (isReverse)
00256     {
00257         for(i = 1; i < 6; i++)
00258         {
00259             drawDiagonalLine(478-x-i, 136, 471-x-i, 0, colour);
00260             drawDiagonalLine(478-x-i, 136, 471-x-i, 272, colour);
00261             drawDiagonalLine(468-x+i, 136, 461-x+i, 0, colour);
00262             drawDiagonalLine(468-x+i, 136, 461-x+i, 272, colour);
00263         }
00264     }
00265     else
00266     {
00267         for(i = 1; i < 6; i++)
00268         {
00269             drawDiagonalLine(x+i, 136, x+7+i, 0, colour);
00270             drawDiagonalLine(x+i, 136, x+7+i, 272, colour);
00271             drawDiagonalLine(x+10-i, 136, x+17-i, 0, colour);
00272             drawDiagonalLine(x+10-i, 136, x+17-i, 272, colour);
00273         }
00274     }
00275     GLCD_SetForegroundColor(GLCD_COLOR_BLACK);
00276 }
00277
00278 // Function for drawing the display for the colour palettes
00279 void drawPalette(int x, int y, int d)
00280 {
```

```
00281     int f = (d+1)/2;
00282
00283     drawRectangle(x, y, d, d, GLCD_COLOR_BLACK);
00284     GLCD_DrawHLine(x, y+f, d);
00285     GLCD_DrawVLine(x+f, y, d);
00286 }
00287
00288 // Filling the colour palettes with their respective colours
00289 void fillPalette(int x, int y, int d, uint32_t colourPalette)
00290 {
00291     int f = (d+1)/2;
00292
00293     fillRectangle(x, y, f, f, colourPalette);
00294     fillRectangle(x+f, y, f-1, f, GLCD_COLOR_BLACK);
00295     fillRectangle(x, y+f, f, f-1, GLCD_COLOR_BLACK);
00296     fillRectangle(x+f, y+f, f-1, f-1, colourPalette);
00297 }
00298
00299 // Highlights the chosen buttons in the settings menu
00300 void highlightButton(int x, int y, int dx, int dy, uint32_t colour)
00301 {
00302     int i;
00303
00304     for (i = 0; i < 4; i++)
00305     {
00306         drawRectangle(x-5-i, y-5-i, dx+10+(2*i), dy+10+(2*i), colour);
00307     }
00308 }
00309
00310 // Function for highlighting temperature and distance units in the settings screen
00311 // without having to repeat the if statement
00312 void highlightTempUnit(int tempUnit)
00313 {
00314     if (tempUnit)
00315     {
00316         highlightButton(25, 135, 70, 30, 0x033F);
00317         highlightButton(131, 135, 70, 30, GLCD_COLOR_WHITE);
00318     }
00319     else
00320     {
00321         highlightButton(131, 135, 70, 30, 0x033F);
00322         highlightButton(25, 135, 70, 30, GLCD_COLOR_WHITE);
00323     }
00324 }
00325
00326 void highlightDistUnit(int distUnit)
00327 {
00328     if (distUnit)
00329     {
00330         highlightButton(25, 215, 70, 30, 0x033F);
00331         highlightButton(131, 215, 70, 30, GLCD_COLOR_WHITE);
00332     }
00333     else
00334     {
00335         highlightButton(131, 215, 70, 30, 0x033F);
00336         highlightButton(25, 215, 70, 30, GLCD_COLOR_WHITE);
00337     }
00338 }
00339
00340 void highlightColour(int colourScheme)
00341 {
00342     //Dependent on colour1 the values will be removed then replace with the required box
00343     if (colourScheme == 0)
00344     {
00345         highlightButton(295, 120, 44, 44, 0x033F); //top left
00346         highlightButton(295, 208, 44, 44, GLCD_COLOR_WHITE); //bot left
00347         highlightButton(396, 120, 44, 44, GLCD_COLOR_WHITE); //top right
00348         highlightButton(396, 208, 44, 44, GLCD_COLOR_WHITE); //bot right
00349
00350     }
00351     else if(colourScheme == 1)
00352     {
00353         highlightButton(295, 120, 44, 44, GLCD_COLOR_WHITE); //top left
00354         highlightButton(295, 208, 44, 44, GLCD_COLOR_WHITE); //bot left
00355         highlightButton(396, 120, 44, 44, 0x033F); //top right
00356         highlightButton(396, 208, 44, 44, GLCD_COLOR_WHITE); //bot right
00357     }
00358     else if(colourScheme == 2)
00359     {
00360         highlightButton(295, 120, 44, 44, GLCD_COLOR_WHITE); //top left
00361         highlightButton(295, 208, 44, 44, 0x033F); //bot left
00362         highlightButton(396, 120, 44, 44, GLCD_COLOR_WHITE); //top right
00363         highlightButton(396, 208, 44, 44, GLCD_COLOR_WHITE); //bot right
00364     }
00365     else if(colourScheme == 3)
00366     {
00367         highlightButton(295, 120, 44, 44, GLCD_COLOR_WHITE); //top left
```

```
00368         highlightButton(295, 208, 44, 44, GLCD_COLOR_WHITE); //bot left
00369         highlightButton(396, 120, 44, 44, GLCD_COLOR_WHITE); //top right
00370         highlightButton(396, 208, 44, 44, 0x033F); //bot right
00371     }
00372 }
00373
00374 int checkCoordsMain(int x, int y){
00375         // Location of the setting button
00376         if (x > 320 && x < 380 && y > 5 && y < 35)
00377         {
00378             // Returns one to change menu screens
00379             return 1;
00380         }
00381         return 0;
00382 }
00383
00384 int checkCoordsSettings(int x, int y)
00385 {
00386         // Location of the back button
00387         if (x > 388 && x < 458 && y > 20 && y < 50)
00388         {
00389             // Returns one to change menu screens
00390             return -1;
00391         }
00392
00393         // temp and distance unit settings
00394         if (x >= 25 && x <= 135)
00395         {
00396             if (y >= 135 && y <= 165)
00397             {
00398                 highlightTempUnit(1);
00399                 return 1;
00400             }
00401             if (y >= 195 && y <= 225)
00402             {
00403                 highlightDistUnit(1);
00404                 return 11;
00405             }
00406         }
00407         else if(x >= 131 && x <= 200)
00408         {
00409             if (y >= 135 && y <= 165)
00410             {
00411                 highlightTempUnit(0);
00412                 return 0;
00413             }
00414             if (y >= 215 && y <= 245)
00415             {
00416                 highlightDistUnit(0);
00417                 return 10;
00418             }
00419         }
00420
00421         //-----Colours setting boxes
00422
00423         else if(x >= 295 && x <= 340)
00424         {
00425             if (y >= 120 && y <= 165)
00426             {
00427                 highlightColour(0);
00428                 return 20;
00429             }
00430             if (y >= 208 && y <= 253)
00431             {
00432                 highlightColour(2);
00433                 return 22;
00434             }
00435         }
00436         else if(x >= 396 && x <= 441)
00437         {
00438             if (y >= 120 && y <= 165)
00439             {
00440                 highlightColour(1);
00441                 return 21;
00442             }
00443             if (y >= 208 && y <= 253)
00444             {
00445                 highlightColour(3);
00446                 return 23;
00447             }
00448         }
00449     // '0' is used to represent 'null'
00450     return 0;
00451 }
00452
00453 void displayDisChevronsLeft(int numChev, uint32_t colour1, uint32_t colour2)
00454 {
```

```
00455     //Remove all current chevrons
00456     drawChevron(68, false, colour1);
00457     fillChevron(68, false, colour1);
00458     drawChevron(51, false, colour1);
00459     fillChevron(51, false, colour1);
00460     drawChevron(34, false, colour1);
00461     fillChevron(34, false, colour1);
00462     drawChevron(17, false, colour1);
00463     fillChevron(17, false, colour1);
00464     drawChevron(0, false, colour1);
00465     fillChevron(0, false, colour1);
00466 
00467     // Is used to place cheverons.
00468     // No breaks are needed because if 1 is need then it goes straight to the bottom
00469     // and if 5 are needed then it will droip through each switch case and add another chevron.
00470     switch(numChev)
00471     {
00472 
00473         case 5:
00474             drawChevron(68, false, colour2);
00475             fillChevron(68, false, colour2);
00476 
00477         case 4:
00478                 drawChevron(51, false, colour2);
00479                 fillChevron(51, false, colour2);
00480 
00481         case 3:
00482                 drawChevron(34, false, colour2);
00483                 fillChevron(34, false, colour2);
00484 
00485         case 2:
00486                 drawChevron(17, false, colour2);
00487                 fillChevron(17, false, colour2);
00488 
00489         case 1:
00490                 drawChevron(0, false, colour2);
00491                 fillChevron(0, false, colour2);
00492 
00493         default:
00494             break;
00495     }
00496 }
00497 
00498 void displayDisChevronsRight(int numChev, uint32_t colour1, uint32_t colour2)
00499 {
00500     //Remove all current chevrons
00501     drawChevron(68, true, colour1);
00502     fillChevron(68, true, colour1);
00503     drawChevron(51, true, colour1);
00504     fillChevron(51, true, colour1);
00505     drawChevron(34, true, colour1);
00506     fillChevron(34, true, colour1);
00507     drawChevron(17, true, colour1);
00508     fillChevron(17, true, colour1);
00509     drawChevron(0, true, colour1);
00510     fillChevron(0, true, colour1);
00511     // Is used to place cheverons.
00512     // No breaks are needed because if 1 is need then it goes straight to the bottom
00513     // and if 5 are needed then it will droip through each switch case and add another chevron.
00514         switch(numChev)
00515     {
00516         case 5:
00517             drawChevron(68, true, colour2);
00518             fillChevron(68, true, colour2);
00519 
00520         case 4:
00521                 drawChevron(51, true, colour2);
00522                 fillChevron(51, true, colour2);
00523 
00524         case 3:
00525                 drawChevron(34, true, colour2);
00526                 fillChevron(34, true, colour2);
00527 
00528         case 2:
00529                 drawChevron(17, true, colour2);
00530                 fillChevron(17, true, colour2);
00531 
00532         case 1:
00533                 drawChevron(0, true, colour2);
00534                 fillChevron(0, true, colour2);
00535 
00536         default:
00537             break;
00538     }
00539 }
00540 
00541 #endif
```

Index