**adc\_driver.c**

/\*

\* adc\_driver.c

\*

\* Created: 18-Oct-17 9:51:05 AM

\* Author: ScorpionIPX

\*/

#include <avr/io.h>

void ADC\_init(void)

{

DDRA = 0x00;

// AREF = AVcc

ADMUX = (1<<REFS0);

// ADC Enable and prescaler of 128

// 16000000/128 = 125000

ADCSRA = (1<<ADEN)|(1<<ADPS2)|(1<<ADPS1)|(1<<ADPS0);

}

// read ADC value

*uint16\_t* ADC\_get\_value(*uint8\_t* ch)

{

// select the corresponding channel 0~7

// ANDing with '7' will always keep the value

// of 'ch' between 0 and 7

ch &= 0b00000111; // AND operation with 7

ADMUX = (ADMUX & 0xF8)|ch; // clears the bottom 3 bits before ORing

// start single conversion

// write '1' to ADSC

ADCSRA |= (1<<ADSC);

// wait for conversion to complete

// ADSC becomes '0' again

// till then, run loop continuously

while(ADCSRA & (1<<ADSC));

return (ADC);

}

**adc\_driver.h**

/\*

\* adc\_driver.h

\*

\* Created: 18-Oct-17 9:58:32 AM

\* Author: uidq6025

\*/

#ifndef ADC\_DRIVER\_H\_

#define ADC\_DRIVER\_H\_

#include <avr/io.h>

#define ADC\_MAX 1023

#define ADC\_HALF 512

void ADC\_init(void);

*uint16\_t* ADC\_get\_value(*uint8\_t* ch);

#endif /\* ADC\_DRIVER\_H\_ \*/

**charge\_driver.c**

/\*

\* charge\_driver.c

\*

\* Created: 24-Mar-18 14:39:59

\* Author: ScorpionIPX

\*/

#include "charge\_driver.h"

#include "adc\_driver.h"

void init\_charge\_control(void)

{

RELAY\_DDR |= 1 << RELAY\_PIN; // Relay pin as output

RELAY\_PORT &= ~(1 << RELAY\_PIN); // Relay default state OFF

}

void turn\_on\_charging(void)

{

TURN\_ON\_RELAY;

}

void turn\_off\_charging(void)

{

TURN\_OFF\_RELAY;

}

unsigned int get\_battery\_voltage(void)

{

unsigned int battery\_voltage = ADC\_get\_value(BATTERY\_ADC\_CHANNEL);

battery\_voltage = battery\_voltage\*((long)BATTERY\_MAX\_MV) / ADC\_MAX;

battery\_voltage = battery\_voltage\*((long) V\_BAT\_GAIN) / 1000;

return battery\_voltage;

}

unsigned int get\_converter\_voltage(void)

{

unsigned int battery\_voltage = ADC\_get\_value(CONVERTER\_ADC\_CHANNEL);

battery\_voltage = battery\_voltage\*((long)CONVERTER\_MAX\_MV)/ADC\_MAX;

battery\_voltage = battery\_voltage\*((long) V\_CHR\_GAIL) / 1000;

return battery\_voltage;

}

**charge\_driver.h**

/\*

\* charge\_driver.h

\*

\* Created: 24-Mar-18 14:40:12

\* Author: ScorpionIPX

\*/

#include <avr/io.h>

#ifndef CHARGE\_DRIVER\_H\_

#define CHARGE\_DRIVER\_H\_

#define BATTERY\_ADC\_CHANNEL 6

#define CONVERTER\_ADC\_CHANNEL 7

#define BATTERY\_MAX\_MV 20000 // ADC channel uses a 1/4 voltage divider

#define CONVERTER\_MAX\_MV 20000 // ADC channel uses a 1/4 voltage divider

#define V\_BAT\_GAIN 975 // as unit of 1000

#define V\_CHR\_GAIL 975 // as unit of 1000

#define RELAY\_PIN PINB4

#define RELAY\_PORT PORTB

#define RELAY\_DDR DDRB

#define TURN\_ON\_RELAY (RELAY\_PORT |= 1 << RELAY\_PIN)

#define TURN\_OFF\_RELAY (RELAY\_PORT &= ~(1 << RELAY\_PIN))

unsigned int get\_battery\_voltage(void);

unsigned int get\_converter\_voltage(void);

void init\_charge\_control(void);

void turn\_on\_charging(void);

void turn\_off\_charging(void);

#endif /\* CHARGE\_DRIVER\_H\_ \*/

**global.h**

/\*

\* global.h

\*

\* Created: 06-Oct-17 11:42:12 AM

\* Author: ScorpionIPX

\*/

#ifndef GLOBAL\_H\_

#define GLOBAL\_H\_

#define F\_CPU 8000000UL

unsigned char STATE;

unsigned char OLD\_STATE;

unsigned char UNIPOLLAR\_01\_CURRENT\_STEP;

#define STATE\_INIT 0

#define STATE\_IDLE 1

#define STATE\_MANUAL 2

#define STATE\_TRACKING 3

#define STATE\_MONITORING 4

#endif /\* GLOBAL\_H\_ \*/

**graphics.c**

/\*

\* graphics.c

\*

\* Created: 28-Oct-17 6:25:15 PM

\* Author: ScorpionIPX

\*/

#include "hx1230.h"

#include "joystick\_driver.h"

#include "sg90\_driver.h"

#include <stdlib.h>

void display\_title(void)

{

hx\_set\_coordinates(0, 0);

hx\_write\_string("AutoTracking LDR");

hx\_set\_coordinates(0, 1);

hx\_write\_string(" ScorpionIPX");

}

void display\_data\_menu(void)

{

hx\_set\_coordinates(0, 3);

hx\_write\_string(" ||");

hx\_set\_coordinates(0, 4);

hx\_write\_string("================");

hx\_set\_coordinates(0, 5);

hx\_write\_string(" ||");

}

void display\_light\_sensor\_data(*uint8\_t* sensor, int data)

{

//hx\_set\_coordinates(60, 3 + sensor);

hx\_set\_coordinates(24 + 36 \* (sensor & 1), 3 + 2 \* (sensor >> 1));

// hx\_write\_char('0' + ((data / 100) % 10));

hx\_write\_char('0' + ((data / 10) % 10));

hx\_write\_char('0' + (data % 10));

hx\_set\_coordinates(18, 7);

hx\_write\_char('0' + OCR1A / 100);

hx\_write\_char('0' + (OCR1A / 10) % 10);

hx\_write\_char('0' + OCR1A % 10);

hx\_set\_coordinates(66, 7);

hx\_write\_char('0' + OCR1B / 100);

hx\_write\_char('0' + (OCR1B / 10) % 10);

hx\_write\_char('0' + OCR1B % 10);

}

void display\_idle\_state\_message(void)

{

hx\_set\_coordinates(0, 3);

hx\_write\_string("- system is in ");

hx\_set\_coordinates(0, 4);

hx\_write\_string("IDLE mode");

hx\_set\_coordinates(0, 6);

hx\_write\_string("going to sleep");

}

void display\_manual\_state\_message(void)

{

hx\_set\_coordinates(0, 3);

hx\_write\_string("- system is in ");

hx\_set\_coordinates(0, 4);

hx\_write\_string("MANUAL mode");

hx\_set\_coordinates(0, 6);

hx\_write\_string("JX:");

hx\_set\_coordinates(54, 6);

hx\_write\_string("B:");

hx\_set\_coordinates(0, 7);

hx\_write\_string("JY:");

hx\_set\_coordinates(54, 7);

hx\_write\_string("A:");

}

void display\_joystick\_data(unsigned int x, unsigned int y)

{

hx\_set\_coordinates(24, 6);

hx\_write\_char('0' + x / 1000);

hx\_write\_char('0' + (x / 100) % 10);

hx\_write\_char('0' + (x / 10) % 10);

hx\_write\_char('0' + x % 10);

hx\_set\_coordinates(72, 6);

hx\_write\_char('0' + SG90\_ROTATE\_DUTY\_CYCLE\_REGISTER / 1000);

hx\_write\_char('0' + (SG90\_ROTATE\_DUTY\_CYCLE\_REGISTER / 100) % 10);

hx\_write\_char('0' + (SG90\_ROTATE\_DUTY\_CYCLE\_REGISTER / 10) % 10);

hx\_write\_char('0' + SG90\_ROTATE\_DUTY\_CYCLE\_REGISTER % 10);

hx\_set\_coordinates(24, 7);

hx\_write\_char('0' + y / 1000);

hx\_write\_char('0' + (y / 100) % 10);

hx\_write\_char('0' + (y / 10) % 10);

hx\_write\_char('0' + y % 10);

hx\_set\_coordinates(72, 7);

hx\_write\_char('0' + SG90\_INCLINE\_DUTY\_CYCLE\_REGISTER / 1000);

hx\_write\_char('0' + (SG90\_INCLINE\_DUTY\_CYCLE\_REGISTER / 100) % 10);

hx\_write\_char('0' + (SG90\_INCLINE\_DUTY\_CYCLE\_REGISTER / 10) % 10);

hx\_write\_char('0' + SG90\_INCLINE\_DUTY\_CYCLE\_REGISTER % 10);

}

void display\_monitoring\_message(void)

{

hx\_set\_coordinates(0, 3);

hx\_write\_string("- system is in ");

hx\_set\_coordinates(0, 4);

hx\_write\_string("MONITORING mode");

hx\_set\_coordinates(0, 6);

hx\_write\_string("Vbat:");

hx\_set\_coordinates(80, 6);

hx\_write\_string("V");

hx\_set\_coordinates(0, 7);

hx\_write\_string("Vchr:");

hx\_set\_coordinates(80, 7);

hx\_write\_string("V");

}

void display\_monitoring\_data(unsigned int v\_bat, unsigned int v\_chr)

{

hx\_set\_coordinates(42, 6);

hx\_write\_char('0' + v\_bat / 10000);

hx\_write\_char('0' + (v\_bat / 1000) % 10);

hx\_write\_char('.');

hx\_write\_char('0' + (v\_bat / 100) % 10);

hx\_write\_char('0' + (v\_bat / 10) % 10);

hx\_write\_char('0' + v\_bat % 10);

hx\_set\_coordinates(42, 7);

hx\_write\_char('0' + v\_chr / 10000);

hx\_write\_char('0' + (v\_chr / 1000) % 10);

hx\_write\_char('.');

hx\_write\_char('0' + (v\_chr / 100) % 10);

hx\_write\_char('0' + (v\_chr / 10) % 10);

hx\_write\_char('0' + v\_chr % 10);

}

**graphics.h**

/\*

\* graphics.h

\*

\* Created: 28-Oct-17 6:25:27 PM

\* Author: ScorpionIPX

\*/

#ifndef GRAPHICS\_H\_

#define GRAPHICS\_H\_

void display\_title(void);

void display\_data\_menu(void);

void display\_light\_sensor\_data(unsigned char sensor, int data);

void display\_idle\_state\_message(void);

void display\_manual\_state\_message(void);

void display\_joystick\_data(unsigned int x, unsigned int y);

void display\_monitoring\_message(void);

void display\_monitoring\_data(unsigned int v\_bat, unsigned int v\_chr);

#endif /\* GRAPHICS\_H\_ \*/

**hx\_8x6characters.h**

/\*

\* hx\_8x6characters.h

\*

\* Created: 06-Oct-17 2:02:58 PM

\* Author: ScorpionIPX

\*/

#ifndef HX\_8X6CHARACTERS\_H\_

#define HX\_8X6CHARACTERS\_H\_

static const unsigned char HX\_character[][6] = {

{0x00,0x00,0x00,0x00,0x00,0x00}, // 0x 0 0

{0x00,0x64,0x18,0x04,0x64,0x18}, // \_ 0x 1 1

{0x00,0x3c,0x40,0x40,0x20,0x7c}, // \_ 0x 2 2

{0x00,0x0c,0x30,0x40,0x30,0x0c}, // \_ 0x 3 3

{0x00,0x3c,0x40,0x30,0x40,0x3c}, // \_ 0x 4 4

{0x00,0x00,0x3e,0x1c,0x08,0x00}, // \_ 0x 5 5

{0x00,0x04,0x1e,0x1f,0x1e,0x04}, // \_ 0x 6 6

{0x00,0x10,0x3c,0x7c,0x3c,0x10}, // \_ 0x 7 7

{0x00,0x20,0x40,0x3e,0x01,0x02}, // \_ 0x 8 8

{0x00,0x22,0x14,0x08,0x14,0x22}, // 0x 9 9

{0x00,0x00,0x38,0x28,0x38,0x00}, // 0x a 10

{0x00,0x00,0x10,0x38,0x10,0x00}, // \_ 0x b 11

{0x00,0x00,0x00,0x10,0x00,0x00}, // \_ 0x c 12

{0x00,0x08,0x78,0x08,0x00,0x00}, // 0x d 13

{0x00,0x00,0x15,0x15,0x0a,0x00}, // \_ 0x e 14

{0x00,0x7f,0x7f,0x09,0x09,0x01}, // \_ 0x f 15

{0x00,0x10,0x20,0x7f,0x01,0x01}, // \_ 0x10 16

{0x00,0x04,0x04,0x00,0x01,0x1f}, // \_ 0x11 17

{0x00,0x00,0x19,0x15,0x12,0x00}, // \_ 0x12 18

{0x00,0x40,0x60,0x50,0x48,0x44}, // \_ 0x13 19

{0x00,0x06,0x09,0x09,0x06,0x00}, // \_ 0x14 20

{0x00,0x0f,0x02,0x01,0x01,0x00}, // \_ 0x15 21

{0x00,0x00,0x01,0x1f,0x01,0x00}, // \_ 0x16 22

{0x00,0x44,0x44,0x4a,0x4a,0x51}, // \_ 0x17 23

{0x00,0x14,0x74,0x1c,0x17,0x14}, // \_ 0x18 24

{0x00,0x51,0x4a,0x4a,0x44,0x44}, // \_ 0x19 25

{0x00,0x00,0x00,0x04,0x04,0x04}, // 0x1a 26

{0x00,0x00,0x7c,0x54,0x54,0x44}, // \_ 0x1b 27

{0x00,0x08,0x08,0x2a,0x1c,0x08}, // \_ 0x1c 28

{0x00,0x7c,0x00,0x7c,0x44,0x7c}, // \_ 0x1d 29

{0x00,0x04,0x02,0x7f,0x02,0x04}, // \_ 0x1e 30

{0x00,0x10,0x20,0x7f,0x20,0x10}, // \_ 0x1f 31

{0x00,0x00,0x00,0x00,0x00,0x00}, // 0x20 32

{0x00,0x00,0x00,0x6f,0x00,0x00}, // ! 0x21 33

{0x00,0x00,0x07,0x00,0x07,0x00}, // " 0x22 34

{0x00,0x14,0x7f,0x14,0x7f,0x14}, // # 0x23 35

{0x00,0x00,0x07,0x04,0x1e,0x00}, // $ 0x24 36

{0x00,0x23,0x13,0x08,0x64,0x62}, // % 0x25 37

{0x00,0x36,0x49,0x56,0x20,0x50}, // & 0x26 38

{0x00,0x00,0x00,0x07,0x00,0x00}, // ' 0x27 39

{0x00,0x00,0x1c,0x22,0x41,0x00}, // ( 0x28 40

{0x00,0x00,0x41,0x22,0x1c,0x00}, // ) 0x29 41

{0x00,0x14,0x08,0x3e,0x08,0x14}, // \* 0x2a 42

{0x00,0x08,0x08,0x3e,0x08,0x08}, // + 0x2b 43

{0x00,0x00,0x50,0x30,0x00,0x00}, // , 0x2c 44

{0x00,0x08,0x08,0x08,0x08,0x08}, // - 0x2d 45

{0x00,0x00,0x60,0x60,0x00,0x00}, // . 0x2e 46

{0x00,0x20,0x10,0x08,0x04,0x02}, // / 0x2f 47

{0x00,0x3e,0x51,0x49,0x45,0x3e}, // 0 0x30 48

{0x00,0x00,0x42,0x7f,0x40,0x00}, // 1 0x31 49

{0x00,0x42,0x61,0x51,0x49,0x46}, // 2 0x32 50

{0x00,0x21,0x41,0x45,0x4b,0x31}, // 3 0x33 51

{0x00,0x18,0x14,0x12,0x7f,0x10}, // 4 0x34 52

{0x00,0x27,0x45,0x45,0x45,0x39}, // 5 0x35 53

{0x00,0x3c,0x4a,0x49,0x49,0x30}, // 6 0x36 54

{0x00,0x01,0x71,0x09,0x05,0x03}, // 7 0x37 55

{0x00,0x36,0x49,0x49,0x49,0x36}, // 8 0x38 56

{0x00,0x06,0x49,0x49,0x29,0x1e}, // 9 0x39 57

{0x00,0x00,0x36,0x36,0x00,0x00}, // : 0x3a 58

{0x00,0x00,0x56,0x36,0x00,0x00}, // ; 0x3b 59

{0x00,0x08,0x14,0x22,0x41,0x00}, // < 0x3c 60

{0x00,0x14,0x14,0x14,0x14,0x14}, // = 0x3d 61

{0x00,0x00,0x41,0x22,0x14,0x08}, // > 0x3e 62

{0x00,0x02,0x01,0x51,0x09,0x06}, // ? 0x3f 63

{0x00,0x3e,0x41,0x5d,0x49,0x4e}, // @ 0x40 64

{0x00,0x7e,0x09,0x09,0x09,0x7e}, // A 0x41 65

{0x00,0x7f,0x49,0x49,0x49,0x36}, // B 0x42 66

{0x00,0x3e,0x41,0x41,0x41,0x22}, // C 0x43 67

{0x00,0x7f,0x41,0x41,0x41,0x3e}, // D 0x44 68

{0x00,0x7f,0x49,0x49,0x49,0x41}, // E 0x45 69

{0x00,0x7f,0x09,0x09,0x09,0x01}, // F 0x46 70

{0x00,0x3e,0x41,0x49,0x49,0x7a}, // G 0x47 71

{0x00,0x7f,0x08,0x08,0x08,0x7f}, // H 0x48 72

{0x00,0x00,0x41,0x7f,0x41,0x00}, // I 0x49 73

{0x00,0x20,0x40,0x41,0x3f,0x01}, // J 0x4a 74

{0x00,0x7f,0x08,0x14,0x22,0x41}, // K 0x4b 75

{0x00,0x7f,0x40,0x40,0x40,0x40}, // L 0x4c 76

{0x00,0x7f,0x02,0x0c,0x02,0x7f}, // M 0x4d 77

{0x00,0x7f,0x04,0x08,0x10,0x7f}, // N 0x4e 78

{0x00,0x3e,0x41,0x41,0x41,0x3e}, // O 0x4f 79

{0x00,0x7f,0x09,0x09,0x09,0x06}, // P 0x50 80

{0x00,0x3e,0x41,0x51,0x21,0x5e}, // Q 0x51 81

{0x00,0x7f,0x09,0x19,0x29,0x46}, // R 0x52 82

{0x00,0x46,0x49,0x49,0x49,0x31}, // S 0x53 83

{0x00,0x01,0x01,0x7f,0x01,0x01}, // T 0x54 84

{0x00,0x3f,0x40,0x40,0x40,0x3f}, // U 0x55 85

{0x00,0x0f,0x30,0x40,0x30,0x0f}, // V 0x56 86

{0x00,0x3f,0x40,0x30,0x40,0x3f}, // W 0x57 87

{0x00,0x63,0x14,0x08,0x14,0x63}, // X 0x58 88

{0x00,0x07,0x08,0x70,0x08,0x07}, // Y 0x59 89

{0x00,0x61,0x51,0x49,0x45,0x43}, // Z 0x5a 90

{0x00,0x3c,0x4a,0x49,0x29,0x1e}, // [ 0x5b 91

{0x00,0x02,0x04,0x08,0x10,0x20}, // \ 0x5c 92

{0x00,0x00,0x41,0x7f,0x00,0x00}, // ] 0x5d 93

{0x00,0x04,0x02,0x01,0x02,0x04}, // ^ 0x5e 94

{0x00,0x40,0x40,0x40,0x40,0x40}, // \_ 0x5f 95

{0x00,0x00,0x00,0x03,0x04,0x00}, // ` 0x60 96

{0x00,0x20,0x54,0x54,0x54,0x78}, // a 0x61 97

{0x00,0x7f,0x48,0x44,0x44,0x38}, // b 0x62 98

{0x00,0x38,0x44,0x44,0x44,0x20}, // c 0x63 99

{0x00,0x38,0x44,0x44,0x48,0x7f}, // d 0x64 100

{0x00,0x38,0x54,0x54,0x54,0x18}, // e 0x65 101

{0x00,0x08,0x7e,0x09,0x01,0x02}, // f 0x66 102

{0x00,0x0c,0x52,0x52,0x52,0x3e}, // g 0x67 103

{0x00,0x7f,0x08,0x04,0x04,0x78}, // h 0x68 104

{0x00,0x00,0x44,0x7d,0x40,0x00}, // i 0x69 105

{0x00,0x20,0x40,0x44,0x3d,0x00}, // j 0x6a 106

{0x00,0x00,0x7f,0x10,0x28,0x44}, // k 0x6b 107

{0x00,0x00,0x41,0x7f,0x40,0x00}, // l 0x6c 108

{0x00,0x7c,0x04,0x18,0x04,0x78}, // m 0x6d 109

{0x00,0x7c,0x08,0x04,0x04,0x78}, // n 0x6e 110

{0x00,0x38,0x44,0x44,0x44,0x38}, // o 0x6f 111

{0x00,0x7c,0x14,0x14,0x14,0x08}, // p 0x70 112

{0x00,0x08,0x14,0x14,0x18,0x7c}, // q 0x71 113

{0x00,0x7c,0x08,0x04,0x04,0x08}, // r 0x72 114

{0x00,0x48,0x54,0x54,0x54,0x20}, // s 0x73 115

{0x00,0x04,0x3f,0x44,0x40,0x20}, // t 0x74 116

{0x00,0x3c,0x40,0x40,0x20,0x7c}, // u 0x75 117

{0x00,0x1c,0x20,0x40,0x20,0x1c}, // v 0x76 118

{0x00,0x3c,0x40,0x30,0x40,0x3c}, // w 0x77 119

{0x00,0x44,0x28,0x10,0x28,0x44}, // x 0x78 120

{0x00,0x0c,0x50,0x50,0x50,0x3c}, // y 0x79 121

{0x00,0x44,0x64,0x54,0x4c,0x44}, // z 0x7a 122

{0x00,0x00,0x08,0x36,0x41,0x41}, // { 0x7b 123

{0x00,0x00,0x00,0x7f,0x00,0x00}, // | 0x7c 124

{0x00,0x41,0x41,0x36,0x08,0x00}, // } 0x7d 125

{0x00,0x04,0x02,0x04,0x08,0x04}, // ~ 0x7e 126

{0x00,0x7f,0x6b,0x6b,0x6b,0x7f}, // 0x7f 127

{0x00,0x00,0x7c,0x44,0x7c,0x00}, // ¬ 0x80 128

{0x00,0x00,0x08,0x7c,0x00,0x00}, //  0x81 129

{0x00,0x00,0x64,0x54,0x48,0x00}, // ‚ 0x82 130

{0x00,0x00,0x44,0x54,0x28,0x00}, // ƒ 0x83 131

{0x00,0x00,0x1c,0x10,0x78,0x00}, // „ 0x84 132

{0x00,0x00,0x5c,0x54,0x24,0x00}, // … 0x85 133

{0x00,0x00,0x78,0x54,0x74,0x00}, // † 0x86 134

{0x00,0x00,0x64,0x14,0x0c,0x00}, // ‡ 0x87 135

{0x00,0x00,0x7c,0x54,0x7c,0x00}, // ˆ 0x88 136

{0x00,0x00,0x5c,0x54,0x3c,0x00}, // ‰ 0x89 137

{0x00,0x78,0x24,0x26,0x25,0x78}, // Š 0x8a 138

{0x00,0x78,0x25,0x26,0x24,0x78}, // ‹ 0x8b 139

{0x00,0x70,0x2a,0x29,0x2a,0x70}, // Œ 0x8c 140

{0x00,0x78,0x25,0x24,0x25,0x78}, //  0x8d 141

{0x00,0x20,0x54,0x56,0x55,0x78}, // } 0x8e 142

{0x00,0x20,0x55,0x56,0x54,0x78}, //  0x8f 143

{0x00,0x20,0x56,0x55,0x56,0x78}, //  0x90 144

{0x00,0x20,0x55,0x54,0x55,0x78}, // ‘ 0x91 145

{0x00,0x7c,0x54,0x56,0x55,0x44}, // ’ 0x92 146

{0x00,0x7c,0x55,0x56,0x54,0x44}, // “ 0x93 147

{0x00,0x7c,0x56,0x55,0x56,0x44}, // ” 0x94 148

{0x00,0x7c,0x55,0x54,0x55,0x44}, // • 0x95 149

{0x00,0x38,0x54,0x56,0x55,0x18}, // – 0x96 150

{0x00,0x38,0x55,0x56,0x54,0x18}, // — 0x97 151

{0x00,0x38,0x56,0x55,0x56,0x18}, // ˜ 0x98 152

{0x00,0x38,0x55,0x54,0x55,0x18}, // ™ 0x99 153

{0x00,0x00,0x44,0x7e,0x45,0x00}, // š 0x9a 154

{0x00,0x00,0x45,0x7e,0x44,0x00}, // › 0x9b 155

{0x00,0x00,0x46,0x7d,0x46,0x00}, // œ 0x9c 156

{0x00,0x00,0x45,0x7c,0x45,0x00}, //  0x9d 157

{0x00,0x00,0x48,0x7a,0x41,0x00}, // ~ 0x9e 158

{0x00,0x00,0x49,0x7a,0x40,0x00}, // Ÿ 0x9f 159

{0x00,0x00,0x4a,0x79,0x42,0x00}, // 0xa0 160

{0x00,0x00,0x49,0x78,0x41,0x00}, // ¡ 0xa1 161

{0x00,0x38,0x44,0x46,0x45,0x38}, // ¢ 0xa2 162

{0x00,0x38,0x45,0x46,0x44,0x38}, // £ 0xa3 163

{0x00,0x38,0x46,0x45,0x46,0x38}, // ¤ 0xa4 164

{0x00,0x38,0x45,0x44,0x45,0x38}, // ¥ 0xa5 165

{0x00,0x30,0x48,0x4a,0x49,0x30}, // ¦ 0xa6 166

{0x00,0x30,0x49,0x4a,0x48,0x30}, // § 0xa7 167

{0x00,0x30,0x4a,0x49,0x4a,0x30}, // ¨ 0xa8 168

{0x00,0x30,0x49,0x48,0x49,0x30}, // © 0xa9 169

{0x00,0x3c,0x40,0x42,0x41,0x3c}, // ª 0xaa 170

{0x00,0x3c,0x41,0x42,0x40,0x3c}, // « 0xab 171

{0x00,0x3c,0x42,0x41,0x42,0x3c}, // ¬ 0xac 172

{0x00,0x3c,0x41,0x40,0x41,0x3c}, // ­ 0xad 173

{0x00,0x3c,0x40,0x42,0x21,0x7c}, // ® 0xae 174

{0x00,0x3c,0x41,0x42,0x20,0x7c}, // ¯ 0xaf 175

{0x00,0x38,0x42,0x41,0x22,0x78}, // ° 0xb0 176

{0x00,0x3c,0x41,0x40,0x21,0x7c}, // ± 0xb1 177

{0x00,0x4e,0x51,0x71,0x11,0x0a}, // ² 0xb2 178

{0x00,0x58,0x64,0x64,0x24,0x10}, // ³ 0xb3 179

{0x00,0x7c,0x0a,0x11,0x22,0x7d}, // ´ 0xb4 180

{0x00,0x78,0x12,0x09,0x0a,0x71}, // µ 0xb5 181

{0x00,0x00,0x00,0x04,0x02,0x01}, // ¶ 0xb6 182

{0x00,0x01,0x02,0x04,0x00,0x00}, // · 0xb7 183

{0x00,0x00,0x02,0x00,0x02,0x00}, // ¸ 0xb8 184

{0x00,0x30,0x48,0x45,0x40,0x20}, // ¹ 0xb9 185

{0x00,0x00,0x00,0x7b,0x00,0x00}, // º 0xba 186

{0x00,0x38,0x44,0x44,0x38,0x44}, // » 0xbb 187

{0x00,0x40,0x3e,0x49,0x49,0x36}, // ¼ 0xbc 188

{0x00,0x08,0x04,0x08,0x70,0x0c}, // ½ 0xbd 189

{0x00,0x60,0x50,0x48,0x50,0x60}, // ¾ 0xbe 190

{0x00,0x20,0x52,0x55,0x59,0x30}, // ¿ 0xbf 191

{0x00,0x38,0x54,0x54,0x54,0x00}, // À 0xc0 192

{0x00,0x00,0x00,0x7f,0x41,0x00}, // Á 0xc1 193

{0x00,0x40,0x22,0x14,0x18,0x60}, // Â 0xc2 194

{0x00,0x7c,0x20,0x20,0x1c,0x20}, // Ã 0xc3 195

{0x00,0x44,0x3c,0x04,0x7c,0x44}, // Ä 0xc4 196

{0x00,0x40,0x3c,0x12,0x12,0x0c}, // Å 0xc5 197

{0x00,0x41,0x63,0x55,0x49,0x41}, // Æ 0xc6 198

{0x00,0x38,0x44,0x44,0x3c,0x04}, // Ç 0xc7 199

{0x00,0x08,0x04,0x3c,0x44,0x24}, // È 0xc8 200

{0x00,0x08,0x14,0x7f,0x14,0x08}, // É 0xc9 201

{0x00,0x4e,0x71,0x01,0x71,0x4e}, // Ê 0xca 202

{0x00,0x45,0x29,0x11,0x29,0x45}, // Ë 0xcb 203

{0x00,0x0d,0x51,0x51,0x51,0x3d}, // Ì 0xcc 204

{0x00,0x00,0x00,0x05,0x02,0x05}, // Í 0xcd 205

{0x00,0x40,0x00,0x40,0x00,0x40}, // Î 0xce 206

{0x00,0x00,0x08,0x1c,0x3e,0x00}, // Ï 0xcf 207

{0x00,0x1c,0x1c,0x1c,0x00,0x00}, // Ð 0xd0 208

{0x00,0x00,0x70,0x08,0x07,0x00}, // Ñ 0xd1 209

{0x00,0x00,0x08,0x08,0x08,0x00}, // Ò 0xd2 210

{0x00,0x00,0x1d,0x15,0x17,0x00}, // Ó 0xd3 211

{0x00,0x00,0x07,0x05,0x07,0x00}, // Ô 0xd4 212

{0x00,0x00,0x11,0x15,0x0a,0x00}, // Õ 0xd5 213

{0x00,0x00,0x00,0x00,0x00,0x00}, // Ö 0xd6 214

{0x00,0x04,0x3c,0x41,0x20,0x00}, // × 0xd7 215

{0x00,0x7c,0x16,0x15,0x16,0x08}, // Ø 0xd8 216

{0x00,0x21,0x16,0x08,0x34,0x42}, // Ù 0xd9 217

{0x00,0x7f,0x09,0x1d,0x01,0x03}, // Ú 0xda 218

{0x00,0x38,0x54,0x54,0x14,0x08}, // Û 0xdb 219

{0x00,0x00,0x00,0x7c,0x40,0x40}, // Ü 0xdc 220

{0x00,0x7f,0x0e,0x1c,0x38,0x7f}, // Ý 0xdd 221

{0x00,0x41,0x22,0x5d,0x22,0x1c}, // Þ 0xde 222

{0x00,0x1c,0x3e,0x1c,0x08,0x00}, // ß 0xdf 223

{0x00,0x7f,0x7f,0x7f,0x7f,0x7f}, // à 0xe0 224

{0x00,0x77,0x7b,0x01,0x7b,0x77}, // á 0xe1 225

{0x00,0x7f,0x43,0x75,0x43,0x7f}, // â 0xe2 226

{0x00,0x7f,0x6f,0x55,0x43,0x7f}, // ã 0xe3 227

{0x00,0x40,0x40,0x40,0x40,0x40}, // ä 0xe4 228

{0x00,0x44,0x42,0x5f,0x42,0x44}, // å 0xe5 229

{0x00,0x40,0x5e,0x45,0x5e,0x40}, // æ 0xe6 230

{0x00,0x40,0x48,0x55,0x5e,0x40}, // ç 0xe7 231

{0x00,0x00,0x04,0x08,0x10,0x20}, // è 0xe8 232

{0x00,0x03,0x07,0x0e,0x1c,0x38}, // é 0xe9 233

{0x00,0x01,0x03,0x07,0x0f,0x1f}, // ê 0xea 234

{0x00,0x7c,0x78,0x70,0x60,0x40}, // ë 0xeb 235

{0x00,0x08,0x08,0x1c,0x22,0x1c}, // ì 0xec 236

{0x00,0x00,0x1c,0x22,0x1c,0x00}, // í 0xed 237

{0x00,0x02,0x00,0x08,0x00,0x20}, // î 0xee 238

{0x00,0x04,0x3e,0x3f,0x3e,0x04}, // ï 0xef 239

{0x00,0x10,0x3e,0x7e,0x3e,0x10}, // ð 0xf0 240

{0x00,0x55,0x2a,0x55,0x2a,0x55}, // ñ 0xf1 241

{0x00,0x24,0x2a,0x7f,0x2a,0x12}, // ò 0xf2 242

{0x00,0x04,0x1e,0x1f,0x1e,0x04}, // ó 0xf3 243

{0x00,0x00,0x00,0x00,0x00,0x00}, // ô 0xf4 244

{0x00,0x00,0x00,0x00,0x00,0x00}, // õ 0xf5 245

{0x00,0x00,0x00,0x00,0x00,0x00}, // ö 0xf6 246

{0x00,0x00,0x00,0x00,0x00,0x00}, // ÷ 0xf7 247

{0x00,0x00,0x00,0x00,0x00,0x00}, // ø 0xf8 248

{0x00,0x00,0x00,0x00,0x00,0x00}, // ù 0xf9 249

{0x00,0x00,0x00,0x00,0x00,0x00}, // ú 0xfa 250

{0x00,0x00,0x00,0x00,0x00,0x00}, // û 0xfb 251

{0x00,0x00,0x00,0x00,0x00,0x00}, // ü 0xfc 252

{0x00,0x00,0x00,0x00,0x00,0x00}, // ý 0xfd 253

{0x00,0x00,0x00,0x00,0x00,0x00}, // þ 0xfe 254

{0x00,0x00,0x00,0x00,0x00,0x00} // ÿ 0xff 255

};

#endif /\* HX\_8X6CHARACTERS\_H\_ \*/

**hx1230.c**

/\*

\* hx1230.c

\*

\* Created: 06-Oct-17 12:35:49 AM

\* Author: ScorpionIPX

\*/

#include "global.h"

#include "hx1230.h"

#include <util/delay.h>

#include "hx\_8x6\_characters.h"

#include <string.h>

void init\_hx1230\_control(void)

{

// set required pins as output

HX1230\_DDR |= ((1 << HX\_RST) | (1 << HX\_CE) | (1 << HX\_DIN) | (1 << HX\_CLK));

// set idle state

CLEAR\_HX\_CLK;

#ifdef HX\_DELAY\_ENABLED

*\_delay\_us*(HX\_DELAY\_US);

#endif

CLEAR\_HX\_RST;

#ifdef HX\_DELAY\_ENABLED

*\_delay\_us*(HX\_DELAY\_US);

#endif

*\_delay\_ms*(50);

SET\_HX\_RST;

#ifdef HX\_DELAY\_ENABLED

*\_delay\_us*(HX\_DELAY\_US);

#endif

CLEAR\_HX\_CE;

#ifdef HX\_DELAY\_ENABLED

*\_delay\_us*(HX\_DELAY\_US);

#endif

*\_delay\_ms*(1);

SET\_HX\_CE;

#ifdef HX\_DELAY\_ENABLED

*\_delay\_us*(HX\_DELAY\_US);

#endif

*\_delay\_ms*(1);

// commands needed to initialize hx1230 display

// found within a chinese data sheet

hx\_send\_command(0x2f);

hx\_send\_command(0x90);

hx\_send\_command(0xa6);

hx\_send\_command(0xa4);

hx\_send\_command(0xaf);

hx\_send\_command(0x40);

hx\_send\_command(0xb0);

hx\_send\_command(0x10);

hx\_send\_command(0x00);

}

void hx\_send\_data(unsigned char \_data)

{

// activate hx1230

CLEAR\_HX\_CE;

#ifdef HX\_DELAY\_ENABLED

*\_delay\_us*(HX\_DELAY\_US);

#endif

// configure communication for data transfer

SET\_HX\_DIN;

#ifdef HX\_DELAY\_ENABLED

*\_delay\_us*(HX\_DELAY\_US);

#endif

// toggle clock

SET\_HX\_CLK;

#ifdef HX\_DELAY\_ENABLED

*\_delay\_us*(HX\_DELAY\_US);

#endif

CLEAR\_HX\_CLK;

#ifdef HX\_DELAY\_ENABLED

*\_delay\_us*(HX\_DELAY\_US);

#endif

// send the actual data, MSB fiHX\_RST

for(int bit\_position = 7; bit\_position >= 0; bit\_position--)

{

// calculate bit to be send

if(((\_data >> bit\_position) & 1) == 1)

{

SET\_HX\_DIN;

#ifdef HX\_DELAY\_ENABLED

*\_delay\_us*(HX\_DELAY\_US);

#endif

}

else

{

CLEAR\_HX\_DIN;

#ifdef HX\_DELAY\_ENABLED

*\_delay\_us*(HX\_DELAY\_US);

#endif

}

// toggle clock

SET\_HX\_CLK;

#ifdef HX\_DELAY\_ENABLED

*\_delay\_us*(HX\_DELAY\_US);

#endif

CLEAR\_HX\_CLK;

#ifdef HX\_DELAY\_ENABLED

*\_delay\_us*(HX\_DELAY\_US);

#endif

}

// deactivate hx1230

SET\_HX\_CE;

}

void hx\_send\_command(unsigned char \_command)

{

// activate hx1230

CLEAR\_HX\_CE;

#ifdef HX\_DELAY\_ENABLED

*\_delay\_us*(HX\_DELAY\_US);

#endif

// configure communication for command transfer

CLEAR\_HX\_DIN;

// toggle clock

SET\_HX\_CLK;

#ifdef HX\_DELAY\_ENABLED

*\_delay\_us*(HX\_DELAY\_US);

#endif

CLEAR\_HX\_CLK;

#ifdef HX\_DELAY\_ENABLED

*\_delay\_us*(HX\_DELAY\_US);

#endif

// send the actual command, MSB fiHX\_RST

for(int bit\_position = 7; bit\_position >= 0; bit\_position--)

{

// calculate bit to be send

if(((\_command >> bit\_position) & 1) == 1)

{

SET\_HX\_DIN;

#ifdef HX\_DELAY\_ENABLED

*\_delay\_us*(HX\_DELAY\_US);

#endif

}

else

{

CLEAR\_HX\_DIN;

#ifdef HX\_DELAY\_ENABLED

*\_delay\_us*(HX\_DELAY\_US);

#endif

}

// toggle clock

SET\_HX\_CLK;

#ifdef HX\_DELAY\_ENABLED

*\_delay\_us*(HX\_DELAY\_US);

#endif

CLEAR\_HX\_CLK;

#ifdef HX\_DELAY\_ENABLED

*\_delay\_us*(HX\_DELAY\_US);

#endif

}

// deactivate hx1230

SET\_HX\_CE;

}

void hx\_set\_coordinates(unsigned char \_x, unsigned char \_y)

{

// 0, 0 is the upper left corner

hx\_send\_command(0xB0 + \_y);

hx\_send\_command(0x10 | ((\_x & 0x7F) >> 4));

hx\_send\_command(0x0F & \_x);

}

void hx\_clear\_screen(void)

{

unsigned char col, row;

hx\_set\_coordinates(0, 0);

for(row = 0; row <= HX\_MAX\_ROW\_ROOT; row ++)

{

for(col = 0; col <= HX\_MAX\_COL; col ++)

{

hx\_send\_data(0x00);

}

}

}

void hx\_fill\_screen(void)

{

unsigned char col, row;

hx\_set\_coordinates(0, 0);

for(row = 0; row < 9; row ++)

{

for(col = 0; col < 96; col ++)

{

hx\_send\_data(0xFF);

}

}

}

void hx\_write\_char(const unsigned char \_character)

{

for(int row\_index = 0; row\_index < 6; row\_index ++)

{

hx\_send\_data(HX\_character[\_character][row\_index]);

}

}

void hx\_write\_string(const char \*\_characters\_array)

{

int string\_length = *strlen*(\_characters\_array);

for(int char\_index = 0; char\_index < string\_length; char\_index++)

{

hx\_write\_char((const unsigned char)(\_characters\_array[char\_index]));

}

}

**hx1230.h**

/\*

\* hx1230.h

\*

\* Created: 06-Oct-17 12:36:42 AM

\* Author: ScorpionIPX

\*/

#include "global.h"

#include <avr/io.h>

#include "hx\_8x6\_characters.h"

#ifndef HX1230\_H\_

#define HX1230\_H\_

//#define HX\_DELAY\_ENABLED // if uC is too fast, HX1230 won't be able to read commands

#ifdef HX\_DELAY\_ENABLED

#define HX\_DELAY\_US 1

#endif

#define HX1230\_PORT PORTC //port used to control hx1230

#define HX1230\_DDR DDRC //data direction register used for hx1230

#define HX\_RST PC0 //external reset input

#define HX\_CE PC1 //chip enable

#define HX\_DIN PC6 //serial data input

#define HX\_CLK PC7 //serial clock input

#define SET\_HX\_RST (HX1230\_PORT |= (1 << HX\_RST))

#define SET\_HX\_CE (HX1230\_PORT |= (1 << HX\_CE))

#define SET\_HX\_DIN (HX1230\_PORT |= (1 << HX\_DIN))

#define SET\_HX\_CLK (HX1230\_PORT |= (1 << HX\_CLK))

#define CLEAR\_HX\_RST (HX1230\_PORT &= ~(1 << HX\_RST))

#define CLEAR\_HX\_CE (HX1230\_PORT &= ~(1 << HX\_CE))

#define CLEAR\_HX\_DIN (HX1230\_PORT &= ~(1 << HX\_DIN))

#define CLEAR\_HX\_CLK (HX1230\_PORT &= ~(1 << HX\_CLK))

#define HX\_MAX\_ROW 64

#define HX\_MAX\_ROW\_ROOT 8

#define HX\_MAX\_COL 96

void init\_hx1230\_control(void);

void hx\_send\_data(unsigned char \_data);

void hx\_send\_command(unsigned char \_command);

void hx\_set\_coordinates(unsigned char \_x, unsigned char \_y);

void hx\_clear\_screen(void);

void hx\_fill\_screen(void);

void hx\_write\_char(const unsigned char \_character);

void hx\_write\_string(const char \*\_characters\_array);

#endif /\* HX1230\_H\_ \*/

**joystick\_driver.c**

/\*

\* joystick\_driver.c

\*

\* Created: 3/19/2018 11:34:43 PM

\* Author: uidq6025

\*/

#include "global.h"

#include <util/delay.h>

#include "user\_interface.h"

#include "adc\_driver.h"

#include "graphics.h"

#include "joystick\_driver.h"

#include "sg90\_driver.h"

#include "unipolar\_driver.h"

#include "l293d.h"

void manual\_control(void)

{

unsigned int x = ADC\_get\_value(ADC\_CHANNEL\_X\_AXIS);

unsigned int y = ADC\_get\_value(ADC\_CHANNEL\_Y\_AXIS);

display\_joystick\_data(x, y);

if(y > (JOYSTICK\_IDLE\_VALUE + JOYSTICK\_DEAD\_ZONE))

{

unipolar\_01\_step\_backward(UNIPOLLAR\_01\_CURRENT\_STEP);

}

else if(y < (JOYSTICK\_IDLE\_VALUE - JOYSTICK\_DEAD\_ZONE))

{

unipolar\_01\_step\_forward(UNIPOLLAR\_01\_CURRENT\_STEP);

}

else

{

unipolar\_01\_clear\_steps();

}

if(x > (JOYSTICK\_IDLE\_VALUE + JOYSTICK\_DEAD\_ZONE))

{

l293d\_hb2\_rotate\_right();

}

else if(x < (JOYSTICK\_IDLE\_VALUE - JOYSTICK\_DEAD\_ZONE))

{

l293d\_hb2\_rotate\_left();

}

else

{

l293d\_hb2\_stop();

}

*\_delay\_ms*(25);

}

signed int format\_axis(unsigned int axis)

{

signed int formated\_axis;

if(axis >= ADC\_HALF)

{

formated\_axis = (axis - ADC\_HALF) / JOYSTICK\_RESOLUTION;

}

else

{

formated\_axis = axis / JOYSTICK\_RESOLUTION;

formated\_axis = formated\_axis \* (-1);

{

}

}

return formated\_axis;

}

**joystick\_driver.h**

/\*

\* joystick\_driver.h

\*

\* Created: 3/19/2018 11:34:57 PM

\* Author: uidq6025

\*/

#ifndef JOYSTICK\_DRIVER\_H\_

#define JOYSTICK\_DRIVER\_H\_

#define JOYSTICK\_RESOLUTION 10

#define JOYSTICK\_IDLE\_VALUE 512

#define JOYSTICK\_DEAD\_ZONE 100

void manual\_control(void);

signed int format\_axis(unsigned int axis);

#endif /\* JOYSTICK\_DRIVER\_H\_ \*/

**main.c**

#include "global.h"

#include <avr/io.h>

#include <util/delay.h>

#include <avr/interrupt.h>

#include "adc\_driver.h"

#include "hx1230.h"

#include "hx\_8x6\_characters.h"

#include "graphics.h"

#include "light.h"

#include "pwm\_driver.h"

#include "sg90\_driver.h"

#include "tracking.h"

#include "user\_interface.h"

#include "state\_handler.h"

#include "joystick\_driver.h"

#include "monitoring.h"

#include "unipolar\_driver.h"

#include "l293d.h"

void uC\_init(void);

int main(void)

{

STATE = STATE\_INIT;

OLD\_STATE = STATE\_INIT;

uC\_init();

STATE = STATE\_IDLE;

while (1)

{

if(STATE\_CHANGED)

{

OLD\_STATE = STATE; // update state

go\_to\_state(STATE);

*\_delay\_ms*(250);

sei(); // enable interrupts

}

switch(OLD\_STATE)

{

case STATE\_TRACKING:

{

track();

break;

}

case STATE\_MANUAL:

{

manual\_control();

break;

}

case STATE\_MONITORING:

{

monitor();

break;

}

default:

{

break;

}

}

}

}

void uC\_init(void)

{

// Wait for system to get fully powered up

*\_delay\_ms*(100);

// initialize required modules

ADC\_init();

*\_delay\_ms*(50);

init\_user\_interface();

*\_delay\_ms*(50);

init\_unipolar\_control();

*\_delay\_ms*(50);

init\_l293d\_control();

*\_delay\_ms*(50);

init\_hx1230\_control();

*\_delay\_ms*(50);

hx\_fill\_screen();

*\_delay\_ms*(500);

hx\_clear\_screen();

*\_delay\_ms*(50);

display\_title();

display\_idle\_state\_message();

sei(); // enable global interrupts

}

**monitoring.c**

/\*

\* monitoring.c

\*

\* Created: 24-Mar-18 15:06:41

\* Author: ScorpionIPX

\*/

#include "global.h"

#include <util/delay.h>

#include "charge\_driver.h"

#include "graphics.h"

void monitor(void)

{

unsigned int battery\_voltage = get\_battery\_voltage();

unsigned int converter\_voltage = get\_converter\_voltage();

display\_monitoring\_data(battery\_voltage, converter\_voltage);

*\_delay\_ms*(100);

}

**monitoring.h**

/\*

\* monitoring.h

\*

\* Created: 24-Mar-18 15:06:54

\* Author: ScorpionIPX

\*/

#ifndef MONITORING\_H\_

#define MONITORING\_H\_

void monitor(void);

#endif /\* MONITORING\_H\_ \*/

**state\_handler.c**

/\*

\* state\_handler.c

\*

\* Created: 3/19/2018 8:08:05 PM

\* Author: uidq6025

\*/

#include "global.h"

#include <util/delay.h>

#include "graphics.h"

#include "state\_handler.h"

#include "hx1230.h"

#include "unipolar\_driver.h"

#include "l293d.h"

void go\_to\_state(unsigned char state)

{

l293d\_hb2\_stop(); /\* make sure motor control is turned off when changing states \*/

unipolar\_01\_clear\_steps(); /\* make sure motor control is turned off when changing states \*/

hx\_clear\_screen();

switch(state)

{

case STATE\_IDLE:

{

STATE = STATE\_IDLE; // update global state

idle\_state\_setup();

break;

}

case STATE\_MANUAL:

{

STATE = STATE\_MANUAL; // update global state

manual\_state\_setup();

break;

}

case STATE\_TRACKING:

{

STATE = STATE\_TRACKING; // update global state

tracking\_state\_setup();

break;

}

case STATE\_MONITORING:

{

STATE = STATE\_MONITORING; // update global state

monitoring\_state\_setup();

break;

}

}

}

void idle\_state\_setup(void)

{

*\_delay\_ms*(200);

hx\_clear\_screen();

display\_title();

display\_idle\_state\_message();

}

void manual\_state\_setup(void)

{

*\_delay\_ms*(200);

hx\_clear\_screen();

display\_title();

display\_manual\_state\_message();

}

void tracking\_state\_setup(void)

{

*\_delay\_ms*(200);

hx\_clear\_screen();

display\_title();

display\_data\_menu();

*\_delay\_ms*(500);

}

void monitoring\_state\_setup(void)

{

*\_delay\_ms*(200);

hx\_clear\_screen();

display\_title();

display\_monitoring\_message();

}

**state\_handler.h**

/\*

\* state\_handler.h

\*

\* Created: 3/19/2018 8:10:15 PM

\* Author: uidq6025

\*/

#include "global.h"

#ifndef STATE\_HANDLER\_H\_

#define STATE\_HANDLER\_H\_

#define STATE\_CHANGED (!(OLD\_STATE == STATE))

void go\_to\_state(unsigned char state);

void idle\_state\_setup(void);

void manual\_state\_setup(void);

void tracking\_state\_setup(void);

void monitoring\_state\_setup(void);

#endif /\* STATE\_HANDLER\_H\_ \*/

**tracking.h**

/\*

\* tracking.h

\*

\* Created: 29-Oct-17 5:25:08 PM

\* Author: ScorpionIPX

\*/

#ifndef TRACKING\_H\_

#define TRACKING\_H\_

#define INCLINE\_TRACKING\_TOLERANCE 2

#define ROTATE\_TRACKING\_TOLERANCE 2

void track(void);

#endif /\* TRACKING\_H\_ \*/

**user\_interface.c**

/\*

\* user\_interface.c

\*

\* Created: 3/19/2018 8:38:39 PM

\* Author: uidq6025

\*/

#include "global.h"

#include <util/delay.h>

#include <avr/interrupt.h>

#include "user\_interface.h"

#include "state\_handler.h"

#include "adc\_driver.h"

void init\_user\_interface(void)

{

init\_next\_state\_button();

}

void init\_next\_state\_button(void)

{

BUTTON\_1\_DRR &= ~(1 << BUTTON\_1\_PIN); // PD2 is input

BUTTON\_1\_PORT |= (1 << BUTTON\_1\_PIN); // turn on the pull-up resistor

// PD2 is now an input with pull-up enabled

MCUCR &= ~(1 << ISC00 | 1 << ISC01); // low level of INT0 generates an interrupt request: when BUTTON\_1 is pressed

GICR |= (1 << INT0); // turns on INT0

}

ISR (INT0\_vect)

{

cli(); // temporarily disable interrupts

switch(STATE)

{

case STATE\_IDLE:

{

STATE = STATE\_MANUAL;

break;

}

case STATE\_MANUAL:

{

STATE = STATE\_TRACKING;

break;

}

case STATE\_TRACKING:

{

STATE = STATE\_MONITORING;

break;

}

case STATE\_MONITORING:

{

STATE = STATE\_IDLE;

break;

}

default:

{

STATE = STATE\_IDLE;

break;

}

}

}

**user\_interface.h**

/\*

\* user\_interface.h

\*

\* Created: 3/19/2018 8:38:53 PM

\* Author: uidq6025

\*/

#include <avr/io.h>

#ifndef USER\_INTERFACE\_H\_

#define USER\_INTERFACE\_H\_

#define BUTTON\_1\_DRR DDRD

#define BUTTON\_1\_PORT PORTD

#define BUTTON\_1\_PIN 2 // PD2

#define ADC\_CHANNEL\_X\_AXIS 4

#define ADC\_CHANNEL\_Y\_AXIS 5

void init\_next\_state\_button(void);

void init\_user\_interface(void);

#endif /\* USER\_INTERFACE\_H\_ \*/