void \_\_attribute\_\_((\_\_interrupt\_\_, \_\_no\_auto\_psv\_\_)) \_T2Interrupt(void) {

if (MotorStatus.Motorstatus.tracking == true) { // go on forever until cancelled

if (Motordrive.period != Motor\_Speeds.tracking) {

Motordrive.period = Motor\_Speeds.tracking;

PR2 = Motordrive.period; // change the motor speed (pulses per second) to the new value

}

OC1CONbits.OCM = 0b100; // reset Output Compare

} else {

if (MotorStatus.Motorstatus.homeing == true) {

OC1CONbits.OCM = 0b100; // continue by resetting output compare

} else { // slewing

if (Movement\_QED != Target\_QED) { // current != target

if (Motordrive.dir == CW) { // rotation clockwise (Current decrementing)

if (Movement\_QED < Target\_QED) { // current position < target, gone too far so stop

OC1CONbits.OCM = 0; // by clearing Output Compare

Movement\_Complete = true; // and flagging complete

} else { // current position >= target position so continue

OC1CONbits.OCM = 0b100; // resetting output compare

}

} else { // rotation counter clockwise (Current incrementing))

if (Movement\_QED > Target\_QED) { // current position > target, not gone far enough, so continue

OC1CONbits.OCM = 0b100; // by resetting Output Compare

} else { // current position <= target so stop

OC1CONbits.OCM = 0; // by clearing output compare

Movement\_Complete = true; // and flagging complete

}

}

} else { // current = target

OC1CONbits.OCM = 0; // so clear output compare

Movement\_Complete = true; // and flagging complete

}

}

}

IFS0bits.T2IF = 0; // clear interrupt flag

}

void \_\_attribute\_\_((\_\_interrupt\_\_, \_\_no\_auto\_psv\_\_)) \_OC1Interrupt(void) {

IFS0bits.OC1IF = 0; // clear the interrupt flag

}

void \_\_attribute\_\_((\_\_interrupt\_\_, \_\_no\_auto\_psv\_\_)) \_T1Interrupt(void) { // millisecond timer

if (millisecondunits < 9) {

millisecondunits++;

millisecond\_timer++;

} else {

millisecondunits = 0;

if (millisecondtens < 9) {

millisecondtens++;

} else {

millisecondtens = 0;

if (decimalseconds < 9) {

decimalseconds++;

} else {

decimalseconds = 0;

if (seconds < 59) // is cumulative seconds < 59?

{

seconds++; // yes, so increment seconds

} else { // else seconds => 59

seconds = 0x00; // reset seconds

if (minutes < 59) // is cumulative minutes < 59?

{

minutes++; // yes, so updates minutes

} else { // else minutes => 59

minutes = 0x00; // reset minutes

if (hours < 23) // is cumulative hours < 23

{

hours++; // yes, so update hours

} else {

hours = 0x00; // reset time

}

}

}

}

}

IFS0bits.T1IF = 0;

}

}

void \_\_attribute\_\_((\_\_interrupt\_\_, \_\_no\_auto\_psv\_\_)) \_CNInterrupt(void) {

static signed short Motor\_PreviousRotaryState;

signed short Motor\_CurrentRotaryState;

LED\_Toggle(LED\_D8);

Motor\_CurrentRotaryState = ReadRotaryState();

if (Motor\_PreviousRotaryState != Motor\_CurrentRotaryState) {

switch (Motor\_CurrentRotaryState) {

case 0b00: //previous state = 00

if (Motor\_PreviousRotaryState == 0b10) {

Current\_QED++;

Current\_Bearing += Bearing\_Increment;

}

if (Motor\_PreviousRotaryState == 0b01) {

Current\_QED--;

Current\_Bearing -= Bearing\_Increment;

}

break;

case 0b01: // previous state RG7 low, RG6 high

if (Motor\_PreviousRotaryState == 0b00) {

Current\_QED++;

Current\_Bearing += Bearing\_Increment;

}

if (Motor\_PreviousRotaryState == 0b11) {

Current\_QED--;

Current\_Bearing -= Bearing\_Increment;

}

break;

case 0b11:

if (Motor\_PreviousRotaryState == 0b01) {

Current\_QED++;

Current\_Bearing += Bearing\_Increment;

}

if (Motor\_PreviousRotaryState == 0b10) {

Current\_QED--;

Current\_Bearing -= Bearing\_Increment;

}

break;

case 0b10:

if (Motor\_PreviousRotaryState == 0b11) {

Current\_QED++;

Current\_Bearing += Bearing\_Increment;

}

if (Motor\_PreviousRotaryState == 0b00) {

Current\_QED--;

Current\_Bearing -= Bearing\_Increment;

}

break;

}

Motor\_PreviousRotaryState = Motor\_CurrentRotaryState;

MotorStatus.Motorstatus.change = true;

Movement\_QED++;

}

IFS1bits.CNIF = 0; // clear Input Change Notification Interrupt flag

}

void \_\_attribute\_\_((\_\_interrupt\_\_, \_\_no\_auto\_psv\_\_)) \_U1RXInterrupt(void) {

unsigned char i = 0;

if (U1STAbits.FERR == 1);

if (U1STAbits.OERR == 1);

U1character = U1RXREG;

U1Buffer[U1BufferInputPointer++] = U1character;

if (U1BufferInputPointer > displaymessagelength) U1BufferInputPointer = 0;

if ((U1character == ETX)&&(U1BufferInputPointer == (displaymessagelength))&&(U1Buffer[0] == STX)) {

Display\_Packet\_Received = true;

for (i = 0; i <= 20; i++) Display\_Packet[i] = U1Buffer[i];

U1BufferInputPointer = 0;

} else Display\_Packet\_Received = false;

\_U1RXIF = 0;

}

void \_\_attribute\_\_((\_\_interrupt\_\_, \_\_no\_auto\_psv\_\_)) \_U2RXInterrupt(void) {

unsigned char i = 0;

if (U2STAbits.FERR == 1);

if (U2STAbits.OERR == 1);

U2character = U2RXREG;

U2Buffer[U2BufferInputPointer++] = U2character;

if (U2BufferInputPointer > (drivermessagelength + 2)) U2BufferInputPointer = 0;

if ((U2character == ETX)&&(U2BufferInputPointer == (drivermessagelength + 2))&&(U2Buffer[0] == STX)) {

Command\_Packet\_Received = true;

for (i = 1; i <= drivermessagelength; i++) Command\_Packet[i - 1] = U2Buffer[i];

U2BufferInputPointer = 0;

} else Command\_Packet\_Received = false;

IFS1bits.U2RXIF = 0;

}

void \_\_attribute\_\_((\_\_interrupt\_\_, \_\_no\_auto\_psv\_\_)) \_U1TXInterrupt(void) {

\_U1TXIF = 0;

}

void \_\_attribute\_\_((\_\_interrupt\_\_, \_\_no\_auto\_psv\_\_)) \_U2TXInterrupt(void) {

IFS1bits.U2TXIF = 0;

}

// End of Interrupt Service Routines ---------------------------------------------------------------------------------------------------------------------

// Main --------------------------------------------------------------------------------------------------------------------------------------------------

int main(void) {

SYS\_Initialize(); // System initialisation

printf("\r"); // clear LCD Screen

ClearLEDS(); // turn all the LEDs off

InitApp(); // hardware Initialisation

InitStatus(); // initialise all status to false

MotorStatus.Motorstatus.atpark = true;

Bearing\_Increment = (double) 1/DEGREE2QED;

Current\_Bearing = (double) HOME\_BEARING;

Current\_QED = (double) 0.0;

while (1) {

// -- Show some Lights ----------------------------------------------------------------------------------------------

LED\_Toggle(LED\_D3); // Toggle D3 to show programme alive

if (HomeSensor == HOME) { // Display the state of the Home Sensor

LED\_On(LED\_D4); // At home (on)

} else {

LED\_Off(LED\_D4); // Away (off)

}

if ((HomeButton\_CW == PRESSED) || (HomeButton\_ACW == PRESSED) || (AZ\_HomeButton\_CW == PRESSED) || (AZ\_HomeButton\_ACW == PRESSED)) LED\_On(LED\_D5);

if ((HomeButton\_CW != PRESSED) && (HomeButton\_ACW != PRESSED) && (AZ\_HomeButton\_CW != PRESSED) && (AZ\_HomeButton\_ACW != PRESSED)) LED\_Off(LED\_D5);

//-- Home Processing ------------------------------------------------------------------------------------------------

if (MotorStatus.Motorstatus.homeing == true) {

if (HomeSensor == HOME) { // reached home (optical sensor on axle))

Turn\_Motor\_Off(); // turn the motor off

MotorStatus.Motorstatus.homeing = false; // and let the world know it is complete

MotorStatus.Motorstatus.change = true;

MotorStatus.Motorstatus.athome = true;

Home\_CW\_requested = false;

Home\_ACW\_requested = false;

Current\_Bearing = HOME\_BEARING;

Current\_QED = 0;

}

}

Check\_Home\_Buttons(); // has operator requested Home by pressing one of the Home Buttons?

//-- Communications from the Hub ---------------------------------------------------------------------------------------------

Check\_Command\_Received(); // anything received from the hub?

//--Slew Processing -------------------------------------------------------------------------------------------------------------------------------

if (MotorStatus.Motorstatus.slewing == true) {

if (Movement\_Complete == true) { // check if slew finished

Turn\_Motor\_Off();

MotorStatus.Motorstatus.slewing = false;

MotorStatus.Motorstatus.change = true;

} // end of QED\_Complete

} // end of is slewing true

//-- Update HUB and 7 Segment Display --------------------------------------------------------------------------------

if (millisecond\_timer >= UPDATE\_TIME) {// update timer is incremented every 500 milliseconds

millisecond\_timer = 0;

Send\_Update\_to\_HUB(0.0, 0.0); // send dynamic information to the hub

Send\_Update\_to\_Display(); // send dynamic information to the 7 Segment display

} // end of update hub

} // end of while

} //end of main

// End of Main -------------------------------------------------------------------------------------------------------------------------------------------

void InitApp(void) {

InitIO();

InitPLLIntOsc(); // Initialisation of PLL

InitTimer1(); // Initialise the clock timer

InitCN(); // Initialisation of Rotary Encoder with CN module

InitU1();

InitU2(); // Initialisation of UART 1

}

void InitIO(void) {

BUTTON\_Enable(BUTTON\_1);

BUTTON\_Enable(BUTTON\_2);

BUTTON\_Enable(BUTTON\_3);

BUTTON\_Enable(BUTTON\_4);

LED\_Enable(LED\_D3);

LED\_Enable(LED\_D4);

LED\_Enable(LED\_D5);

LED\_Enable(LED\_D6);

LED\_Enable(LED\_D7);

LED\_Enable(LED\_D8);

LED\_Enable(LED\_D9);

LED\_Enable(LED\_D10);

TurnOnMotorDirection(); // initialise the motor direction

TurnOnHome(); // initialise pins associated with the HOME function

TurnOnRotary(); // initialise the QEI (Rotary) Encoder)

TurnOnDisplay(); // initialise the 7 Segment Display

}

void InitStatus(void) {

MotorStatus.Motorstatus.athome = false;

MotorStatus.Motorstatus.atpark = false;

MotorStatus.Motorstatus.homeing = false;

MotorStatus.Motorstatus.running = false;

MotorStatus.Motorstatus.slewing = false;

MotorStatus.Motorstatus.tracking = false;

MotorStatus.Motorstatus.change = false;

}

void InitCN(void) {

ADPCFG = 0xffff; //set all pins to digital

\_TRISG6 = 1; // make RG6 pin an input

\_TRISG7 = 1; // make RG7 pin an input

\_CN8IE = 1; // Enable Input Change Notification Interrupt on RG6 pin

\_CN9IE = 1; // Enable Input Change Notification Interrupt on RG7 pin

\_CNIF = 0; // Clear Input Change Notification Flag

\_CNIP = 3; // Set priority of Input Change Notification Interrupt

\_CNIE = 1; // Enable Input Change Notification Interrupt

}

void Turn\_Motor\_On(double speed, char direction) {

Motordrive.dir = direction;

Set\_Motor\_Direction(); // implement the motor direction

Motordrive.period = speed;

Motordrive.pulse\_rising = Motordrive.period >> 1; // OCR - signal goes high

Motordrive.pulse\_falling = Motordrive.period; // OCRS - signal goes low

InitTimer2();

Movement\_Complete = false;

Movement\_QED = 0;

OC1CONbits.OCM = 0b100; // pin starts low, driven high on OC1P/OC1PS

T2CONbits.TON = 0x01; // Start Timer2

MotorStatus.Motorstatus.running = true;

MotorStatus.Motorstatus.change = true;

}

void Turn\_Motor\_Off(void) {

OC1CONbits.OCM = 0; // stop the output compare

T2CONbits.TON = 0; // stop Timer2

Movement\_QED = 0;

MotorStatus.Motorstatus.running = false;

MotorStatus.Motorstatus.change = true;

}

void Check\_Home\_Buttons(void) {

// Hardware instruction to Turn Motor CW

if ((HomeButton\_CW == PRESSED) && (HomeButton\_ACW == PRESSED) && (AZ\_HomeButton\_CW == PRESSED)) {

Turn\_Motor\_On(Motor\_Speeds.homeing, CW);

MotorStatus.Motorstatus.homeing = true;

MotorStatus.Motorstatus.running = true;

MotorStatus.Motorstatus.change = true;

}// Hardware instruction to Turn Motor ACW

if ((HomeButton\_CW == PRESSED) && (HomeButton\_ACW == PRESSED) && (AZ\_HomeButton\_ACW == PRESSED)) {

Turn\_Motor\_On(Motor\_Speeds.homeing,ACW);

MotorStatus.Motorstatus.homeing = true;

MotorStatus.Motorstatus.running = true;

MotorStatus.Motorstatus.change = true;

}

if ((HomeButton\_CW == PRESSED) && (HomeButton\_ACW == NOTPRESSED) && (AZ\_HomeButton\_CW == NOTPRESSED) && (AZ\_HomeButton\_ACW == NOTPRESSED)) {

Home\_CW\_requested = true;

Home\_ACW\_requested = false;

}

if ((HomeButton\_CW == NOTPRESSED) && (HomeButton\_ACW == PRESSED) && (AZ\_HomeButton\_CW == NOTPRESSED) && (AZ\_HomeButton\_ACW == NOTPRESSED)) {

Home\_ACW\_requested = true;

Home\_CW\_requested = false;

}

if ((HomeButton\_CW == NOTPRESSED) && (HomeButton\_ACW == NOTPRESSED) && (AZ\_HomeButton\_CW == PRESSED) && (AZ\_HomeButton\_ACW == NOTPRESSED)) {

Send\_Reply\_to\_HUB(AZ\_Home\_command, CW, 0);

}

if ((HomeButton\_CW == NOTPRESSED) && (HomeButton\_ACW == NOTPRESSED) && (AZ\_HomeButton\_CW == NOTPRESSED) && (AZ\_HomeButton\_ACW == PRESSED)) {

Send\_Reply\_to\_HUB(AZ\_Home\_command, ACW, 0);

}

if ((Home\_CW\_requested == true) || (Home\_ACW\_requested == true)) { // a new home request made

if ((MotorStatus.Motorstatus.running == false) && (HomeSensor == AWAY)) { // check we are not running and not at home already

if ((Home\_CW\_requested == true)) {

Turn\_Motor\_On(Motor\_Speeds.homeing, CW);

}

if ((Home\_ACW\_requested == true)) {

Turn\_Motor\_On(Motor\_Speeds.homeing, ACW);

}

MotorStatus.Motorstatus.homeing = true;

MotorStatus.Motorstatus.athome = false;

MotorStatus.Motorstatus.change = true;

}

}

}

void Check\_Command\_Received(void) {

unsigned char i;

if (Command\_Packet\_Received == true) {

LED\_Toggle(LED\_D7);

for (i = 1; i <= LENGTHMESSAGE; i++) {

CommandMessage.sg\_chars[i] = Command\_Packet[i];

}

Command\_Packet\_Received = false;

Command\_Packet\_reply = false;

switch (Command\_Packet[0]) { // first byte of packet is the command byte

case 0:

break;

// <editor-fold defaultstate="collapsed" desc="AZ\_home\_command">

#ifdef AZIMUTH\_MOTOR

case AZ\_home\_command:

Send\_Reply\_to\_HUB(AZ\_home\_command, good, 0);

GoHome();

break;

#endif

// </editor-fold>

// <editor-fold defaultstate="collapsed" desc="c2 TargetAzimuth\_command">

#ifdef AZIMUTH\_MOTOR

case TargetAzimuth\_command:

TargetBearing\_value = CommandMessage.Drivermessage.field1;

Send\_Reply\_to\_HUB(TargetAzimuth\_command, good, TargetBearing\_value);

break;

#endif

// </editor-fold>

// <editor-fold defaultstate="collapsed" desc="c3 TargetAltitude\_command">

#ifdef ALTITUDE\_MOTOR

case TargetAltitude\_command:

TargetBearing\_value = CommandMessage.Drivermessage.field1;

Send\_Reply\_to\_HUB(TargetAltitude\_command, good, TargetBearing\_value);

break;

#endif

// </editor-fold>

// <editor-fold defaultstate="collapsed" desc="c4 Tracking\_command">

case Tracking\_command: // Turn tracking on or off

// some work necessary here to determine speed and direction of track

if (CommandMessage.Drivermessage.field1 > (double) 0.0) { // Start

if (Tracking\_Rates\_Set != true) {

Motor\_Speeds.tracking = Motor\_Speeds.sidereal; // default speed = sidereal

}

Turn\_Motor\_On(Motor\_Speeds.tracking,ACW);

} else { // Stop

MotorStatus.Motorstatus.tracking = false;

Turn\_Motor\_Off();

}

Send\_Reply\_to\_HUB(Tracking\_command, good, CommandMessage.Drivermessage.field1);

break;

// </editor-fold>

// <editor-fold defaultstate="collapsed" desc="c5 TrackingRates\_command">

case TrackingRates\_command: // variable will contain the speed in arcseconds/second to implement when tracking engaged

#ifdef AZIMUTH\_MOTOR

Motor\_Speeds.tracking = CommandMessage.Drivermessage.field1; // Azimuth Rate

#else

Motor\_Speeds.tracking = CommandMessage.Drivermessage.field2; // Altitude Rate

#endif

Tracking\_Rates\_Set = true;

Send\_Reply\_to\_HUB(Tracking\_command, good, Motor\_Speeds.tracking);

break;

// </editor-fold>

// <editor-fold defaultstate="collapsed" desc="c6 AbortSlew\_command">

case AbortSlew\_command:

Turn\_Motor\_Off();

MotorStatus.Motorstatus.slewing = false;

Send\_Reply\_to\_HUB(AbortSlew\_command, good, 0);

break;

// </editor-fold>

// <editor-fold defaultstate="collapsed" desc="c7 FindHome\_command">

case FindHome\_command:

Command\_Packet\_Reply\_value = CommandMessage.Drivermessage.field1;

Command\_Packet\_Reply\_answer = good;

Command\_Packet\_Reply\_command = FindHome\_command;

Command\_Packet\_reply = true;

GoHome();

break;

// </editor-fold>

// <editor-fold defaultstate="collapsed" desc="c8 Park\_command">

case Park\_command:

Send\_Reply\_to\_HUB(Park\_command, good, CommandMessage.Drivermessage.field1);

GoHome();

MotorStatus.Motorstatus.atpark = true;

break;

// </editor-fold>

// <editor-fold defaultstate="collapsed" desc="c9 SlewToAltAz\_command">

case SlewToAltAz\_command:

Turn\_Motor\_Off(); // stop any current movement

double Requested\_Bearing = CommandMessage.Drivermessage.field1; // get the required bearing

Target\_QED = Calc\_QEDpulses2target(Requested\_Bearing); // calculate how far and in which direction to slew

Turn\_Motor\_On(Motor\_Speeds.slew,Motordrive.dir); // start the motor turning

MotorStatus.Motorstatus.slewing = true; // set the status to slewing

Send\_Reply\_to\_HUB(SlewToAltAz\_command, good, Requested\_Bearing); // reply to command

break;

// </editor-fold>

// <editor-fold defaultstate="collapsed" desc="c10 SlewToTarget\_command">

case SlewToTarget\_command:

Turn\_Motor\_Off();

Target\_QED = Calc\_QEDpulses2target(TargetBearing\_value); // calculate how far and in which direction to slew

Turn\_Motor\_On(Motor\_Speeds.slew,Motordrive.dir); // start the motor running

MotorStatus.Motorstatus.slewing = true; // set the status to slewing

Send\_Reply\_to\_HUB(SlewToTarget\_command, good, TargetBearing\_value);

break;

// </editor-fold>

// <editor-fold defaultstate="collapsed" desc="c11 SyncToAltAz\_command">

case SyncToAltAz\_command:

Current\_Bearing = (double) CommandMessage.Drivermessage.field1;

Send\_Reply\_to\_HUB(SyncToAltAz\_command, good, Current\_Bearing);

break;

// </editor-fold>

// <editor-fold defaultstate="collapsed" desc="c12 UnPark\_command">

case UnPark\_command:

MotorStatus.Motorstatus.atpark = false;

Send\_Reply\_to\_HUB(UnPark\_command, good, 0);

break;

// </editor-fold >

// <editor-fold defaultstate="collapsed" desc="default\_command">

default:

break;

// </editor-fold>

}

}

}

void Send\_Reply\_to\_HUB(char command, char reply, double value) {

int i = 0;

LED\_Toggle(LED\_D6);

CommandMessage.Drivermessage.command\_number = command; // [0]

CommandMessage.Drivermessage.status\_byte = reply; // [1]

CommandMessage.Drivermessage.field1 = value; // [2,3,4,5]

CommandMessage.Drivermessage.field2 = 0.0; // [6,7,8,9]

CommandMessage.Drivermessage.field3 = 0.0; // [10,11,12,13]

CommandMessage.Drivermessage.field4 = 0.0; // [14,15,16,17]

while (U2STAbits.UTXBF);

U2TXREG = (char) STX;

for (i = 0; i < 18; i++) {

while (U2STAbits.UTXBF);

U2TXREG = CommandMessage.sg\_chars[i];

}

while (U2STAbits.UTXBF);

U2TXREG = (char) ETX;

}

void Send\_Update\_to\_HUB(double field3, double field4) {

int i = 0;

LED\_Toggle(LED\_D6);

CommandMessage.Drivermessage.command\_number = 0x05; // [0]

CommandMessage.Drivermessage.status\_byte = MotorStatus.Status; // [1]

CommandMessage.Drivermessage.field1 = Current\_Bearing; // [2,3,4,5]

CommandMessage.Drivermessage.field2 = Current\_QED; // [6,7,8,9]

CommandMessage.Drivermessage.field3 = field3; // [10,11,12,13]

CommandMessage.Drivermessage.field4 = field4; // [14,15,16,17]

while (U2STAbits.UTXBF);

U2TXREG = (char) STX;

for (i = 0; i < 18; i++) {

while (U2STAbits.UTXBF);

U2TXREG = CommandMessage.sg\_chars[i];

}

while (U2STAbits.UTXBF);

U2TXREG = (char) ETX;

MotorStatus.Motorstatus.change = false;

}

void Send\_Update\_to\_Display(void) {

DisplayMessage.Displaymessage.altitude = (double) Current\_Bearing; // [1,2,3,4]

while (U1STAbits.UTXBF);

U1TXREG = (char) STX;

for (i = 0; i < 4; i++) {

while (U1STAbits.UTXBF);

U1TXREG = DisplayMessage.sg\_chars[i];

}

while (U1STAbits.UTXBF);

U1TXREG = (char) ETX;

}

long Calc\_QEDpulses2target(double Target\_Bearing) { // Calculate the required number of QED pulses to travel and the direction of travel

double BearingDelta = (Target\_Bearing - Current\_Bearing);

while (BearingDelta <-(double) 180.0) BearingDelta += (double) 360.0;

while (BearingDelta > (double) 180.0) BearingDelta -= (double) 360.0;

if (BearingDelta < (double) 0) {

BearingDelta = (BearingDelta \* (double) -1.0); // BearingDelta should always be positive

Motordrive.dir = ACW;

} else Motordrive.dir = CW;

return (long) (BearingDelta \* (double) DEGREE2QED); // calculate and return QED to move

}

void Set\_Motor\_Direction(void) {

if (Motordrive.dir == CW) {

Motor\_DirectionCW(); // set direction pin to clockwise

LED\_Off(LED\_D9);

LED\_On(LED\_D10);

} else {

Motor\_DirectionACW(); // set direction pin to anticlockwise

LED\_Off(LED\_D10);

LED\_On(LED\_D9);

}

}

void GoHome(void) { // go home

MotorStatus.Motorstatus.atpark = false;

if (HomeSensor == HOME) { // are we already at HOME

MotorStatus.Motorstatus.athome = true;

Current\_Bearing = (double) HOME\_BEARING;

Current\_QED = 0;

} else { // NOT at HOME, so start, or keep motor turning

if (Current\_Bearing > (double) HOME\_BEARING) {

Motordrive.dir = CW;

} else {

Motordrive.dir = ACW;

}

MotorStatus.Motorstatus.athome = false;

Turn\_Motor\_On(Motor\_Speeds.homeing,Motordrive.dir); // start the motor running

MotorStatus.Motorstatus.homeing = true;

MotorStatus.Motorstatus.athome = false;

MotorStatus.Motorstatus.change = true;

}

}