```
// Robot Software
// Created By Garrett Webster and Arash Ahmadi
// with suplimentry code segmants from C.P. Diduch (UNB ECE4333 Prof.)
//
// Last Edited April 6th 2013
//
// Robot has two modes of operation: Manual Control (wasd)
//                                   Automated Magnetic Path Following (GMR)
//
// Instructions will print out on the bluetooth screen for operational selection and control
// A debugger/monitor is included for displaying various values and outputs (see below)
//
/* ================================================================================== */
// Included Libraries
#include "rtos.h"
#include "mbed.h"
#include "math.h"
#include "stdlib.h"
/* ================================================================================== */
// Definitions
#define Dummy 0
#define pi 3.14159

/* GMR Control Definitions */
#define GMReMax 10          //max error in cm
#define GMReMin -10         //min error in cm

/* Motor PI Control Definitions */
#define eMax 20             //max error in cm/s
#define eMin -20            //min error in cm/s
#define xMax 50             //max eular approximation in cm/s
#define xMin -50            //min eular approximation in cm/s
#define uMax 50             //max control signal in cm/s
#define uMin -50            //min control signal in cm/s

#define VmaxR 32            //absolute max capable speed of right motor  and gear train in cm/s
#define VmaxL 32            //absolute max capable speed of left motor and gear train in cm/s

/* ================================================================================== */
/* Function prototypes */
/* GMR Prototypes */
void GMRControllerISR(void);
void GMRControlThread(void const *argument);

/* Exterior Collision Prototypes */
void ExtCollisionISR(void);
void ExtCollisionThread(void const *argument);

/* Motor Control Prototypes */
void PiControllerISR(void);
void PiControlThread(void const *argument);
void MotorRight(float u);
void MotorLeft(float u);
```

```
/* Timer Prototypes */
void WdtFaultISR(void);
void Watchdog(void const *n);

/* ============================================================================================ */
/* Global Variables */
/* GMR PI Variables */
float Beta1=0, Beta2=0, Alpha=0.0243;   //GMR Aplha and Beta Profile Values
float Off1=3.3, Off2=3.3;
float L;                                //Seperation between GMR sensors
float GMRxState;                        //GMR integration euler approximations
float gKp=2, gKi=2;                     //GMR PI control gains
float gKg = 16;                         //GMR conversion (from cm to cm/s)
                                        //   (+ if sensor in front of axle, - if behind)

/* Motor PI Variables */
int dPositionR=0, dPositionL=0;         //encoder position read variables
int dTimeR=0, dTimeL=0;                 //encoder time read variables
float eR=0, eL=0;                       //wheel velocity errors from setpoints
float xStateR=0, xStateL=0;             //motor integration euler approximations
float uR=0, uL=0, OlduR=0, OlduL=0;     //motor control signals
float Kp=0.75, Ki=0.9;                  //motor PI control gains  p=0.5 i=0.74

/* Setpoints */
float SetpointR=0, SetpointL=0;         //control signals for wheel velocities in cm/s
float OldSetpointR=0, OldSetpointL=0;   //storage variables for speed tracking
float Set=0, Steer=0;                   //desired setpoint for robot velocity and steering in
cm/s

/* ============================================================================================ */
/* Debugger */
/* MBED must be connected to a PC via the USB cable to use the debugger                         */
/* Debug = 0 --> Print Out Nothing                                                              */

/* Debug = 1 --> Print Out GMR Calibration Readings                                             */
/* Debug = 2 --> Print Out PI Control GMR Readings                                              */
/* Debug = 3 --> Print Out Tracking Results (Roadway)                                           */
/* Debug = 4 --> Print Out GMR e, x, u Calculation Results                                      */
/* Debug = 5 --> Print Out GMR Motor Outputs                                                    */
/* Debug = 6 --> Print Out Encoder Readings from DE0                                            */
/* Debug = 7 --> Print Out Calculated Wheel Velocities                                          */
/* Debug = 8 --> Print Out dVel, e, x, u, setpoint for each wheel (Right then Left)             */
int Debug = 8;
/* ============================================================================================ */

/* ============================================================================================ */
/* Controll Signal */
/* Control = 1 --> Normal Operation, PI Thread and QEI Encoders Enabled                         */
/* Control = 0 --> Basic Operation, PI Thread and QEI Encoders Disabled                         */
int Control = 1;
/* ============================================================================================ */
```

```
/* Processes and threads */
/* GMR Threads */
int32_t SignalGMR;
osThreadId GMRControl;
osThreadDef(GMRControlThread, osPriorityNormal, DEFAULT_STACK_SIZE);

/* Motor Control Threads */
int32_t SignalPi;
osThreadId PiControl;
osThreadDef(PiControlThread, osPriorityNormal, DEFAULT_STACK_SIZE);

/* Exterior Collision Threads */
int32_t SignalExtCollision;
osThreadId ExtCollision;
osThreadDef(ExtCollisionThread, osPriorityNormal, DEFAULT_STACK_SIZE);

/* Wdt Threads */
int32_t SignalWdt;
osThreadId WdtFault;
osTimerDef(Wdtimer, Watchdog);
Ticker PeriodicInt;

/* ================================================================================== */
// IO Port Configuration
/* GMR Pins */
AnalogIn GMR1(p19);               // Left GMR input
AnalogIn GMR2(p20);               // Right GMR input

/* Motor Pins */
DigitalOut MotorRDir(p26);        // Dirction Control Right Motor
PwmOut MotorRpwm(p25);            // Speed Control Right Motor
DigitalOut MotorLDir(p24);        // Direction Control Left Motor
PwmOut MotorLpwm(p23);            // Speed Control Left Motor

/* Collision Bumper Interupt Pins */
InterruptIn Bumper(p8);           // External interrupt pin

/* LED Pins */
DigitalOut led1(LED1);
DigitalOut led2(LED2);
DigitalOut led3(LED3);
DigitalOut led4(LED4);

/* Serial Communication Pins */
Serial pc(USBTX, USBRX);            // (tx, rx) for Parani/Promi Bluetooth serial channel
Serial BluetoothSerial(p28, p27);   // (tx, rx) for PC serial channel
SPI DE0(p5, p6, p7);                // (mosi, miso, sclk) DE0 is the SPI channel with the DE0
FPGA
DigitalOut SpiReset(p12);           // DE0 Reset (pulse high then low to trigger reset)
DigitalOut SpiStart(p11);           // DE0 Start Read (pulse high then low to trigger read)

/* ================================================================================== */
// ******** Main Thread ********
```

```
int main() {

    /* Initialize and start Collision Control Thread */
    ExtCollision = osThreadCreate(osThread(ExtCollisionThread), NULL);
    osTimerId OneShot = osTimerCreate(osTimer(Wdtimer), osTimerOnce, (void *)0);
    Bumper.rise(&ExtCollisionISR);  // Attach the address of the interrupt handler to the
                                    //      rising edge of Bumper

    Reset:          //marker to reset control option
    int x = 0;
    char Mode='k';

    /* Select mode of operation */
    BluetoothSerial.printf("\n\rTo perform Manual Control of your robot press 'm'.\n\r");
    BluetoothSerial.printf("To perform Automated GMR path following press 'g'.\n\r");

    do{
        Mode = BluetoothSerial.getc();
        BluetoothSerial.printf("You entered: %c \n\n\r", Mode);  // Echo entry
        if( (Mode!='m') && (Mode!='g')  ){
            BluetoothSerial.printf("Invalid Selection\n\r");
            BluetoothSerial.printf("Enter 'm' for manual control or 'g' for GMR guidpath
            following.\n\r");
            x=0;
        }
        if( (Mode=='m') || (Mode=='g')  )
            x = 1;
    }while(x==0);

    if (Control==1){
            /* Initialize and start PI Motor Control Threads */
            PiControl = osThreadCreate(osThread(PiControlThread), NULL);
            PeriodicInt.attach(&PiControllerISR, 0.05); // Specify address of the TimerISR
                                                        //   (Ticker) function and the interval
                                                        //    between interrupts
    }

    /* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
    /* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
    /* Manual Control Mode */
    /* 'w' = increase forward velocity */
    /* 's' = decrease forward velocity */
    /* 'a' = increase tightness of left turn */
    /* 'd' = increase tighness of right turn */
    /* 'p' = zero out any turning, drive straight at current set velocity */
    /* ' ' = full stop, reset all values to 0 */
    do{
    if( Mode == 'm' ){
        BluetoothSerial.printf("Manual Control Mode Entered\n\n\r");
        /* Setup SPI communications between MBED and DE0 */
        SpiReset = 1;                   // Set SpiReset;
        wait_us(1);                     // Hold pulse on for clear read of reset pin
        SpiReset = 0;                   // Clear SpiReset;
```

```
DE0.format(16,0);               // Format DE0

/* Setup Bluetooth communications between MBED and Computer */
pc.baud(9600);     //set baut rate of serial communication to DE0

char c='y';
/* Continously read the bluetooth serial channel for new set points */
do {
   if (pc.readable()){
        c=pc.getc();                        // Read 'wasd' controls from PC
        osTimerStart(OneShot, 2000);        // Set the watchdog timer interrupt to 2s.
        led3=0;
        led4=0;
    }

    if(BluetoothSerial.readable()) {
        c = BluetoothSerial.getc();         // Read 'wasd' controls from Bluetooth
    }

    /* ------------------------------------------------------------------------ */
    /* Decode 'wasd' Controls */
    /* Full Stop */
    if(c==' '){
        Set = 0;
        Steer = 0;
        MotorRight(0);
        MotorLeft(0);
    }
    /* Straighten Out */
    else if(c=='f'){
        Set = (SetpointR+SetpointL)/2;      // Average current velocities
        Steer = 0;                          // Set both wheels to same velocity
    }
    /* Velocity Increase */
    else if(c=='w'){
        Set = Set + 0.5;
    }
    else if(c=='t'){
        Set = Set + 2;
    }
    /* Velocity Decrease */
    else if(c=='s'){
        Set = Set - 0.5;
    }
    else if(c=='g'){
        Set = Set - 2;
    }
    /* Turn Left */
    else if(c=='a'){
        Steer = Steer + 0.25;
    }
    /* Turn Right */
    else if(c=='d'){
```

```
            Steer = Steer - 0.25;
        }
        /* Exit Manual control */
        else if(c=='m'){
            Set = 0;
            Steer = 0;
            MotorRight(0);
            MotorLeft(0);
            goto Reset;
        }

        c = 'y';            // reset control character
        BluetoothSerial.printf("%2.2f  %2.2f\n\r", Set, Steer);  // Echo Inputs

        /* Generate Motor Setpoints */
        SetpointR = Set + Steer;
        SetpointL = Set - Steer;

        /* ########################################################################## */

        /* Send Motor Setpoints Directly to Motors (skip PI) if change in speed */
        if (Control==0){
            if (SetpointR != OldSetpointR)
                    MotorRight(SetpointR);
            if (SetpointL != OldSetpointL)
                    MotorLeft(SetpointL);

            OldSetpointR = SetpointR;
            OldSetpointL = SetpointL;
        }
        /* ########################################################################## */
        /* -------------------------------------------------------------------------- */

        Thread::wait(100);         // Wait __ms before checking for new setpoint
    }while(1);
} // end if Manual Control Mode
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */

/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
/* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
/* Automated GMR guidepath following Mode */
/* Robot will follow a current carying wire */
if (Mode == 'g'){
    char c='y';
    int x;
    float Cal1, Cal2;

    /* Calibration */
    // You have 5 seconds to scroll the GMRs over the current carying wire
    // Make sure they are scrolled over the wire at the height that they will
    // be mounted on the robot.
    BluetoothSerial.printf("GMR Sensor PI Controller\n\n\n\r");
```

```
BluetoothSerial.printf("Scroll the GMRs over the current carrying wire for 5
seconds.\n\r");
BluetoothSerial.printf("Ensure they are held at the same height they will be mounted to
the robot.\n\n\r");
BluetoothSerial.printf("Press 's' to start the calibration.\n\n\r");

/* Start Calibration */
/* Wait until user presses 's' */
do{
    c = BluetoothSerial.getc();
}while(c != 's');

/* Init Beta to impossible value */
/* If calibration doesn't update these values then there was an error */
Beta1=0;
Beta2=0;

/* Scan for max values (i.e. Beta) */
for(x=0;x<100;x++)
{
    Cal1 = (GMR1*3.3);
    if (Cal1>Beta1)
        Beta1=Cal1;
    if (Cal1<Off1)
        Off1=Cal1;

    Cal2 = (GMR2*3.3);
    if (Cal2>Beta2)
        Beta2=Cal2;
    if (Cal2<Off2)
        Off2=Cal2;
    wait(0.05);

    /* ?????????????????????????????????????????????????????????????????????? */
    /* Debug Mode 1 */
    /* Prints out Calibration GMR readings */
    if(Debug == 1)
        pc.printf("GMR1 = %f --------- GMR2 = %f\n\r", Cal1, Cal2);
    /* ?????????????????????????????????????????????????????????????????????? */
}

/* Check for Errors */
// If scan reviels max reading of 0V than GMRs are impropperly connected or not
functioning
if (Beta1==0){
    BluetoothSerial.printf("Error GMR1 circuit isn't functioning properly.\n\r");
    return(1);
}
if (Beta2==0){
    BluetoothSerial.printf("Error GMR2 circuit isn't functioning properly.\n\r");
    return(1);
}
```

```
/* Print out calibration results */
BluetoothSerial.printf("GMR1 Max = %f --------- GMR2 Max = %f\n\r", Beta1, Beta2);
BluetoothSerial.printf("Off1 = %f ---------- Off2 = %f\n\r", Off1, Off2);

/* Enter GMR seperation */
BluetoothSerial.printf("\n\rEnter the seperation of the sensors in cm:");
BluetoothSerial.scanf("%f", &L);

/* Initialize and start GMR Control Thread */
GMRControl = osThreadCreate(osThread(GMRControlThread), NULL);
PeriodicInt.attach(&GMRControllerISR, 0.25); // Specify address of the TimerISR
                                             //    (Ticker) function and the
                                             //     interval between interrupts

/* Main operational Thread */
/* Continously read the bluetooth serial channel for new set points */
do {
   if (pc.readable()){
        c=pc.getc();                         // Read PC
        osTimerStart(OneShot, 2000);         // Set the watchdog timer interrupt to 2s.
        led3=0;
        led4=0;
    }
    if(BluetoothSerial.readable()) {
        c = BluetoothSerial.getc();          // Read Right Motor Setpoint
    }

    /* ------------------------------------------------------------------------ */
    /* Decode 'ws ' Controls */
    /* Full Stop */
    if(c==' '){
        Set = 0;
        Steer = 0;
        MotorRight(0);
        MotorLeft(0);
    }
    /* Velocity Increase */
    else if(c=='w'){
        Set = Set + 0.5;
    }
    /* Velocity Decrease */
    else if(c=='s'){
        Set = Set - 0.5;
    }
    /* Exit GMR Automated Control */
    else if(c=='m'){
        Set = 0;
        Steer = 0;
        MotorRight(0);
        MotorLeft(0);
        goto Reset;
    }
```

```
            c = 'y';
            BluetoothSerial.printf("%2.2f  %2.2f\n\r", Set, Steer);// Echo Inputs
            /* ------------------------------------------------------------------------ */

            Thread::wait(100);          // Wait __ms before checking for new setpoint
        }while(1);
    }// end if GMR guidepath following mode
    /* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~*/
    /* ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ */
    }while(1);
} // end main

/* ============================================================================================== */
// ******** GMR Control Thread ********
void GMRControlThread(void const *argument) {
while (true) {
    float Sensor1=0, Sensor2=0, Diff=0;         //GMR sensor readings and difference
    float e, u;                                 //PI control variables
    int n, x;                                   //counter variables and constants

    osSignalWait(SignalGMR, osWaitForever);  // Wait for next cycle
    led2= !led2;                              // Alive status

    /* Read GMRs and Calculate Difference */
    // Amplified GMR signals are read (AtoD) and scaled up to their actual
    // voltage levels (0-3.3V)
    Sensor1 = (GMR1*3.3)-Off1;              // Read and scale GMR1 and remove offset
    Sensor2 = (GMR2*3.3)-Off2;              // Read and scale GMR2 and remove offset
    Diff = Sensor1 - Sensor2;               // Create differential profile

    /* ???????????????????????????????????????????????????????????????????????????? */
    /* Debug Mode 2 */
    /* Continously prints out the GMR readings for visual analysis */
    if (Debug == 2){
        pc.printf("GMR1 = %f  ----------  GMR2 = %f\n\r", Sensor1, Sensor2);
        pc.printf("Off1 = %f -------- Off2 = %f\n\r", Off1, Off2);
    }
    /* ???????????????????????????????????????????????????????????????????????????? */

    /* Calculate the Error */
    /* Use sign of difference to determine which side of sensor plot wire is in */
    /* Alpha/Beta equation is centered about the GMR, since this is a differential */
    /*    sensor there is a shift of +/- L/2 and another +/- 1 for the center */
    if(Diff>0){
       e = (L/2) + 1 - sqrt(-1*(log(Sensor1/Beta1)/Alpha));         // find error from GMR
       1
       if(e<0)
            e=0;
       }
    else if(Diff<0){
       e = -1*(L/2) -1 + sqrt(-1*(log(Sensor2/Beta2)/Alpha));       //Find error from GMR 2
        if(e>0)
            e=0;
```

```
    }
  else
    e = 0;                    // Special case where there is no error, very unlikely

  /* ??????????????????????????????????????????????????????????????????????????????? */
  /* Debug Mode 3 */
  /* Displays a simulated output of where the wire, and two GMRs are located */
  /* Looks like a roadway with a dotted line */
  if (Debug == 3){
      n=L+(e*2);                      // Rough markings at 0.5cm resolution
      pc.printf("x");                 // Print Left GMR representation
      for(x=1;x<2*L;x++){
          if(x==n)
              pc.printf("1");         // Print calculated wire location
          else
              pc.printf("  ");        // Print blanks for other locations
      }
      pc.printf("x\n\r");             // Print Right GMR representation
  } // End Debug Mode 3
  /* ??????????????????????????????????????????????????????????????????????????????? */

  Steer = Diff * gKg;                 //Gain conversion from error in cm to velocity in cm/s

  /* Generate Motor Setpoints */
  SetpointR = Set + Steer;
  SetpointL = Set - Steer;

  /* ############################################################################### */

  /* Send Motor Setpoints Directly to Motors (skip PI) if change in speed */
  if (Control==0){
      if (SetpointR != OldSetpointR)
              MotorRight(SetpointR);
          if (SetpointL != OldSetpointL)
              MotorLeft(SetpointL);
  }
  /* ############################################################################### */

  /* ??????????????????????????????????????????????????????????????????????????????? */
  /* Debug Mode 5 */
  /* Print Out Motor Setpoints */
  if (Debug == 5)
      pc.printf("\n\rSet = %f    Steer = %f", Set, Steer);
  /* ??????????????????????????????????????????????????????????????????????????????? */

  OldSetpointR = SetpointR;
  OldSetpointL = SetpointL;

  /* ??????????????????????????????????????????????????????????????????????????????? */
  /* Debug Mode 4 */
  /* Print Out GMR PI Control Variables for Plotting*/
  if (Debug == 4)
      pc.printf("\n\r%f    %f    %f    %f", e, GMRxState, u, Diff);
```

```
    /* ??????????????????????????????????????????????????????????????????????????????? */
    }
} // end PIControlThread

/* ============================================================================================ */
// ******** Motor PI Control Thread ********
void PiControlThread(void const *argument) {
    float dVelR=0, dVelL=0;

    while (true) {
        osSignalWait(SignalPi, osWaitForever);
        led2= !led2;                            // Alive status

        /* Read the Right Encoder for Position and Time */
        SpiStart = 1;                           // Signal to start SPI read from DE0
        wait_us(2);                             // hold pulse high for clean read
        SpiStart = 0;                           // Pull signal low to start read

        DE0.write(0x8002);                      // Control Word

        dPositionR = DE0.write(Dummy);          //read motor R position change from DE0
        dTimeR = DE0.write(Dummy);              //read motor R time change from DE0 in ms

        /* Read the Left Encoder for Position and Time */
        SpiStart = 1;                           // Signal to start SPI read from DE0
        wait_us(2);                             // hold pulse high for clean read
        SpiStart = 0;                           // Pull signal low to start read

        DE0.write(0x8202);                      // Control Word

        dPositionL = DE0.write(Dummy);          //read motor L position change from DE0
        dTimeL = DE0.write(Dummy);              //read motor L time change from DE0 in ms

        /* Mask Position Reading for relevant bits */
        if(dPositionR&0x00008000)dPositionR=dPositionR|0xFFFF0000;
        if(dPositionL&0x00008000)dPositionL=dPositionL|0xFFFF0000;

        dPositionR = -1*dPositionR;             //-1 correction factor for inverted readings
        dPositionL = -1*dPositionL;             //-1 correction factor for inverted readings

        /* ??????????????????????????????????????????????????????????????????????????????? */
        /* Debug Mode 6 */
        /* Print out encoder readings from DE0 */
        if(Debug == 6)
            pc.printf("dPosR = %d -------- dTimeR = %d -------- dPosL = %d -------- dTimeL =
            %d\n\r", dPositionR, dTimeR, dPositionL, dTimeL);
        /* ??????????????????????????????????????????????????????????????????????????????? */

        /* ########################################################################## */
        /* Convert Encoder readings to wheel velocities (cm/s) */
        /* Vel = (((Ne/400)*2*pi)/(t/1000)) * Ng * Np * Rw */
        /* Vel = (((dPosition/400)*2*pi)/(dTime/1000)) * 189/2365 * 36/90 * 5 */
        /* Ne = # of Encoder Pulses */
```

```
        /* t = Encoder Timer reading in ms */
        /* Ng = Gearbox Ratio */
        /* Np= Pulley Ratio */
        /* Rw = Wheel Radius */

        dVelR = 2.510617*dPositionR/dTimeR;
        dVelL = 2.510617*dPositionL/dTimeL;

        /* ??????????????????????????????????????????????????????????????????????????????????? */
        /* Debug Mode 7 */
        /* Print out calculated wheel velocities */
        if(Debug == 7)
            pc.printf("dVelR = %d ---------------------- dVelL = %d \n\r", dVelR, dVelL);
        /* ??????????????????????????????????????????????????????????????????????????????????? */
        /* ########################################################################## */

        /* ------------------------------------------------------------------------- */
        /* Calculate Velocity Errors */
        eR = SetpointR - dVelR;     // motor R velocity error in 'cm/s'
        eL = SetpointL - dVelL;      // motor L velocity error in 'cm/s'

        /* Check for overflow errors */
        /* if Vel<Setpoint but error is Negative then overflow */
        /* if Vel>setpoint but error is Positive then overflow */
        if ((dVelR < SetpointR) && (eR<0))
            eR = eMax;
        else if ((dVelR > SetpointR) && (eR>0))
            eR = eMin;
        if ((dVelL < SetpointL) && (eL<0))
            eL = eMax;
        else if ((dVelL > SetpointL) && (eL>0))
            eL = eMin;

        /* ------------------------------------------------------------------------- */
        /* Calculate Integration Approximations */
        xStateR = xStateR + eR;             // eular approximation of integration of eR in 'cm/s'
        xStateL = xStateL + eL;             // eular approximation of integration of eR in 'cm/s'

        /* Set Saurations of Integration Approximation */
        if(xStateR>xMax)
            xStateR = xMax;
        if(xStateR<xMin)
            xStateR = xMin;
        if(xStateL>xMax)
            xStateL = xMax;
        if(xStateL<xMin)
            xStateL = xMin;

        /* Prevent Integrator Windup */
        if((uR>=uMax) || (uR<=uMin))
            xStateR = 0;
        if((uL>=uMax) || (uL<=uMin))
            xStateL = 0;
```

```
        /* ------------------------------------------------------------------------ */
        /* Calculate Motor Outputs */
        uR = Kp*eR + Ki*xStateR;            // motor R control signal in 'cm/s'
        uL = Kp*eL + Ki*xStateL;            // motor L control signal in 'cm/s'

        /* Motor Saturation Levels */
        if(uR>uMax)
            uR = uMax;
        if(uR<uMin)
            uR = uMin;
        if(uL>uMax)
            uL = uMax;
        if(uL<uMin)
            uL = uMin;

        if (SetpointR == 0){
            uR = 0;
            xStateR = 0;
        }
        if (SetpointL == 0){
            uL = 0;
            xStateL = 0;
        }
        /* Send Motor Outputs, Only if there is a change */
        if (uR != OlduR)
            MotorRight(uR);
        if (uL != OlduL)
            MotorLeft(uL);

        /* Store Motor Speeds */
        OlduR = uR;
        OlduL = uL;

        /* ------------------------------------------------------------------------ */

        /* ???????????????????????????????????????????????????????????????????????? */
        /* Debug Mode 8 */
        /* Print out dVel, e, x, u, setpoint for each wheel (Right then Left) */
        if(Debug == 8){
            pc.printf("%f    %f   %f   %f    %f   ", dVelR, eR, xStateR, uR, SetpointR);
            pc.printf("%f    %f   %f   %f    %f   \n\r", dVelL, eL, xStateL, uL, SetpointL);
        } //end Debug Mode 8
        /* ???????????????????????????????????????????????????????????????????????? */
    }
} // end PIControlThread

/* ============================================================================================ */

// ******** Right Motor Control Function *********
void MotorRight(float u) {

    int Dir;
```

```
    float pwm, percent;

    /* convert handed speed u to percentage of max speed */
    percent = u/VmaxR*100;

    /* Set max/min limits for motor speed */
    if(percent>100)
        percent=100;
    if(percent<-100)
        percent=-100;

    /* convert percentage speed to PWM pulse width */
    pwm = abs(percent)/100*0.020;         //magnitude is pwm scaler

    /* use magnitude speed as direction control */
    Dir = u/abs(u);
    if (Dir == 1)
        MotorRDir = 1;      //sign controls direction (yields +/- 1)
    else
        MotorRDir = 0;

    /* Assign PWM and Dir to motor output control pins */
    MotorRpwm.period(0.020);
    MotorRpwm.pulsewidth(pwm);
} //end MotorRight

/* ================================================================================================ */
// ******** Left Motorl Control Function *********
void MotorLeft(float u) {

    int Dir;
    float pwm, percent;

    /* convert handed speed u to percentage of max speed */
    percent = u/VmaxL*100;

    /* Set max/min limits for motor speed */
    if(percent>100)
        percent=100;
    if(percent<-100)
        percent=-100;

    /* convert percentage speed to PWM pulse width */
    pwm = abs(percent)/100*0.020;         //magnitude is pwm scaler

    /* use magnitude speed as direction control */
    Dir = u/abs(u);
    if (Dir == 1)
        MotorLDir = 0;      //sign controls direction (yields +/- 1)
    else
        MotorLDir = 1;

    /* Assign PWM and Dir to motor output control pins */
```

```
    MotorLpwm.period(0.020);
    MotorLpwm.pulsewidth(pwm);
} //end MotorLeft

/* ================================================================================================ */
// ******** Collision Thread ********
void ExtCollisionThread(void const *argument) {
while (true) {
    osSignalWait(SignalExtCollision, osWaitForever);
    led4 = 1;
    }
} // end ExtCollisionThread

/* ================================================================================================ */
// ******** Watchdog Interrupt Handler ********
void Watchdog(void const *n) {
    led3=1;
} // end Watchdog

/* ================================================================================================ */
// ******** Period Timer Interrupt Handler ********
void GMRControllerISR(void) {
    osSignalSet(GMRControl,0x1);
} // end PIControllerISR

/* ================================================================================================ */
// ******** Period Timer Interrupt Handler ********
void PiControllerISR(void) {
    osSignalSet(PiControl,0x1);
} // end PIControllerISR

/* ================================================================================================ */
// ******** Collision Interrupt Handler ********
void ExtCollisionISR(void) {
    osSignalSet(ExtCollision,0x1);
} // end ExtCollisionISR

/* ================================================================================================ */
```