VEX Robotics Programming & Sensors with Robot C

Author: Zijin Wang

Mentor: Yifei Han

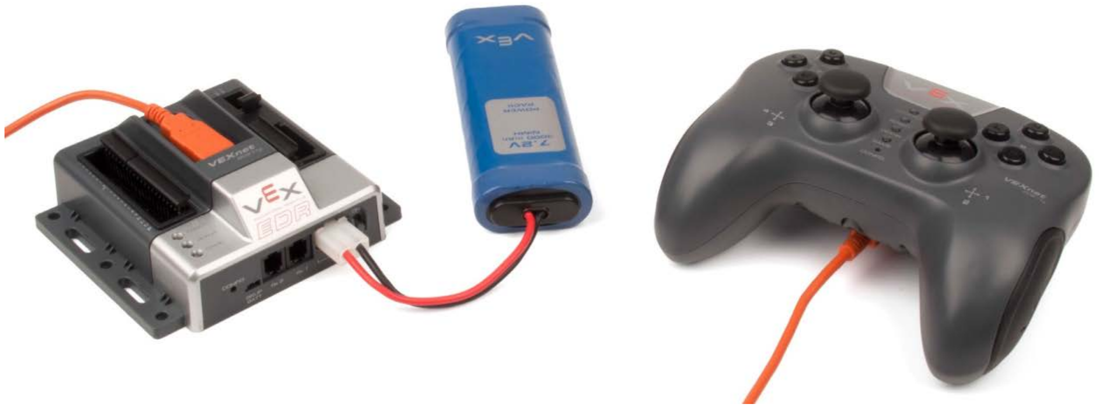
April 2018

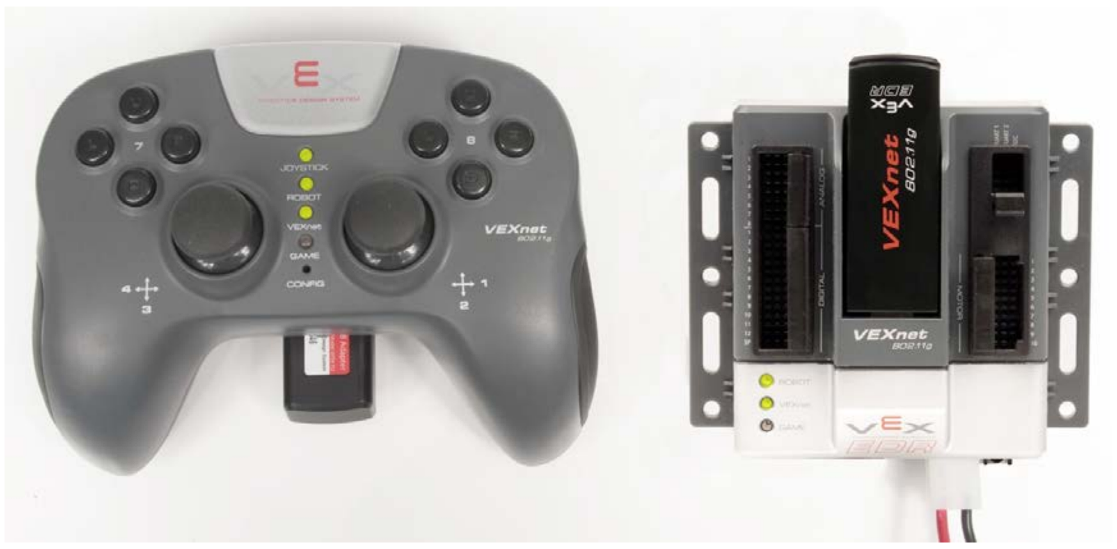
1. Joystick Control Programming

a. Basic Connections

The VEXnet Joystick allows a human operator to control a robot in real-time using the VEXnet wireless link. The joystick has two 2-axis analog joysticks, 4 trigger buttons and two 4-button directional pads.

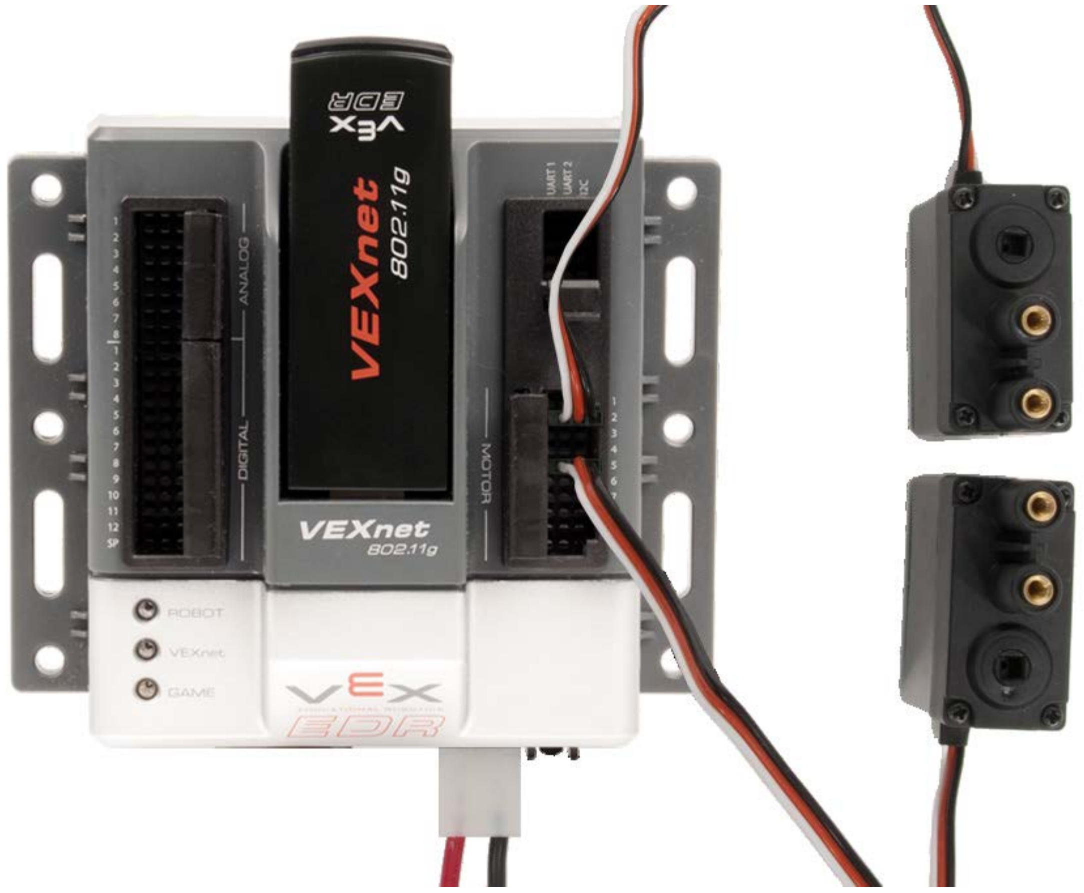
The Joystick must first be paired to the VEX ARM® Cortex®-based Microcontroller before they will work using VEXnet Keys. Pairing requires a USB A-A Cable and a VEX 7.2V Battery. This process must be completed each time you use a Joystick or Microcontroller with a new mate. A Joystick can only communicate with the Microcontroller that it has been paired with. During the Pairing Process, the ID from the VEX ARM® Cortex®-based Microcontroller is transferred to the Joystick, thus mating the two units together.





b. Basic Configuration

The following picture shows two VEX 3-Wire Motors plugged into Motor Port 2 and Motor Port 5. With the Default Code, pushing Joystick Channel 3 up will cause Motor 2 to turn clockwise. Pushing Joystick Channel 2 up will cause Motor 5 to turn counterclockwise.



c. Information of each port

① Analog Information:

Analog lines are input only and read about 0.2 volts when open. Circuit details are as follows:

a. Analog input range is 0 to +5 volts.

b. Analog to digital resolution is 12-bit, compiler resolution may vary.

c. Analog circuit has a 470k pull-up to +5 volts, a series 10k resistor and a 20k resistor to ground.

d. Analog inputs also have a 1000 pF capacitor to ground on the processor side of the 10k resistor.

e. 3 dB bandwidth: 16 kHz

② Digital Information:

When configured as an input, digital lines have a weak pull-up. When configured as an output,

digital lines drive 0 volts for a low and 3.3 volts for a high. Circuit details are as follows:

a. Digital input range is 0 to +5 volts.

b. Digital drive is primarily limited by the 1k series resistor, so it can output a 2v high into 2k-ohms to ground or a 0.8v

low into 7k-ohms to 3.3v.

c. Digital inputs also have a 1000 pF capacitor to ground on the processor side of the 1k resistor.

d. 3 dB input bandwidth: 150 kHz.

③ 2-Wire Motor Port outputs:

a. Motor Port 1 and Motor Port 10.

b. Maximum motor stall current: 3.0 amps at 8.5 volts.

c. Motor chop rate: determined by the compiler. Default code chop rate: 1 kHz.

d. Overcurrent protection: Motor Port 1 through Motor Port 5 shares one 4 amp circuit breaker. Motor Port 6 through

Motor Port 10 shares a second 4 amp circuit breaker.

④ 3-Wire Motor Port outputs:

a. Motor Ports 2 through 9.

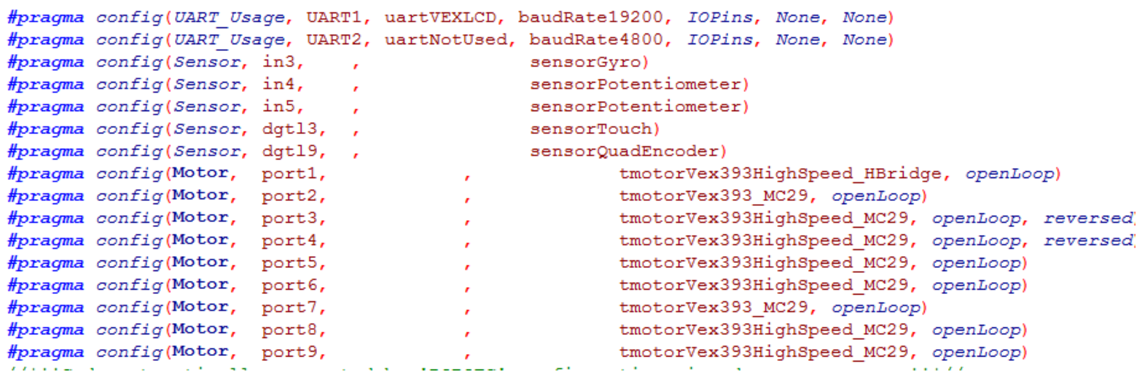
b. Maximum motor stall current: internally limited by motor assembly.

c. Motor PWM output: determined by the compiler. Default is 1 to 2 milliseconds high time and a 17 millisecond period.

d. Overcurrent protection: Motor Port 1 through Motor Port 5 shares one 4 amp circuit breaker. Motor Port 6 through

Motor Port 10 shares a second 4 amp circuit breaker.

d. Cortex Platform Programming for Motors and Sensors System


```
#pragma config(UART_Usage, UART1, uartVEXLCD, baudRate19200, IOPins, None, None)
#pragma config(UART_Usage, UART2, uartNotUsed, baudRate4800, IOPins, None, None)
#pragma config(Sensor, in3,    ,               sensorGyro)
#pragma config(Sensor, in4,    ,               sensorPotentiometer)
#pragma config(Sensor, in5,    ,               sensorPotentiometer)
#pragma config(Sensor, dgtl3,  ,               sensorTouch)
#pragma config(Sensor, dgtl9,  ,               sensorQuadEncoder)
#pragma config(Motor,  port1,           ,             tmotorVex393HighSpeed_HBridge, openLoop)
#pragma config(Motor,  port2,           ,             tmotorVex393_MC29, openLoop)
#pragma config(Motor,  port3,           ,             tmotorVex393HighSpeed_MC29, openLoop, reversed
#pragma config(Motor,  port4,           ,             tmotorVex393HighSpeed_MC29, openLoop, reversed
#pragma config(Motor,  port5,           ,             tmotorVex393HighSpeed_MC29, openLoop)
#pragma config(Motor,  port6,           ,             tmotorVex393HighSpeed_MC29, openLoop)
#pragma config(Motor,  port7,           ,             tmotorVex393_MC29, openLoop)
#pragma config(Motor,  port8,           ,             tmotorVex393HighSpeed_MC29, openLoop)
#pragma config(Motor,  port9,           ,             tmotorVex393HighSpeed_MC29, openLoop)
```

e. Definition of each motor used in the robot

```
#include "Vex_Competition_Includes.c"!!*//

#define data vexRT
#define Hand motor[port6]
#define Arm motor[port2]
#define Taker motor[port5]
#define Left motor[port4] = motor[port3]
#define Right motor[port8] = motor[port9]
#define Roller motor[port7]

#define Gyro SensorValue[in3]
#define Pot_Hand SensorValue[in5]
#define Pot_Arm SensorValue[in4]
#define Ecd SensorValue[dgtl9]
#define Limit SensorValue[dgtl3]

short mode = 1;
short auto_point,auto_cone;
char auto_side;
```

f. Joystick Control System Programming

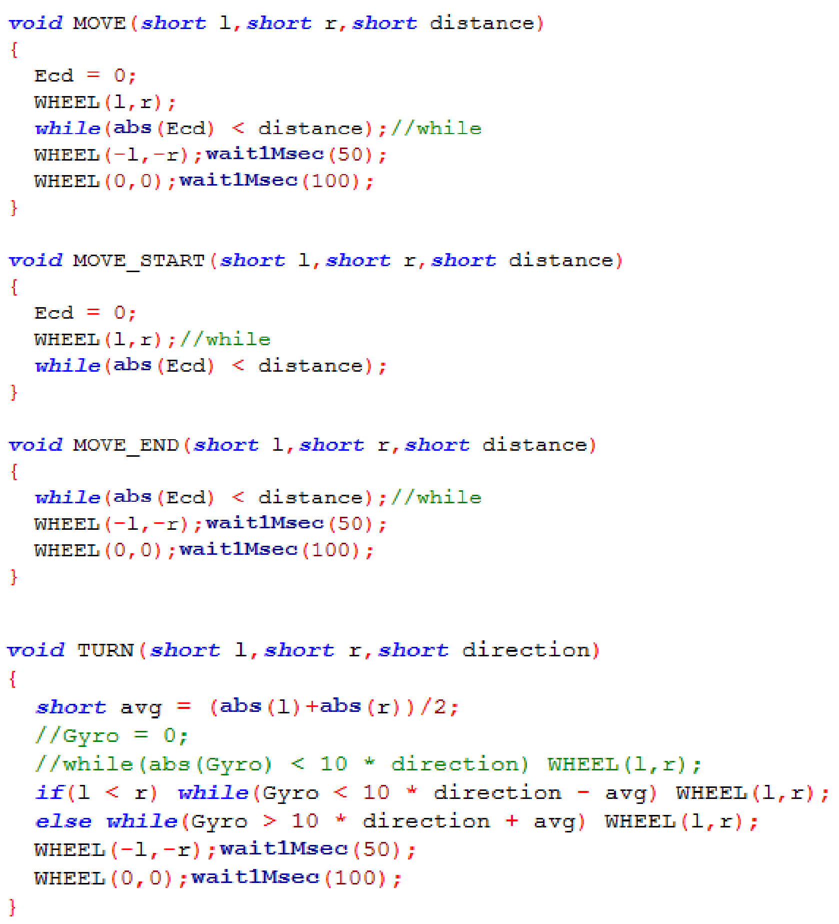
```
void Joystick()
{
  short x = vexRT[Ch4],y = vexRT[Ch3];
  if(auto_cone == 2)WHEEL(vexRT[Ch1]*100/127,vexRT[Ch1]*100/127);
  else  WHEEL(y * 100/127 + x * 90/127,y * 100/127 - x * 90/127);
  TAKER(vexRT[Btn8U] * 100 - vexRT[Btn8D] * 100);//U for out, L for in
  ARM(vexRT[Btn6U] * 85 - vexRT[Btn6D] * 85);//90,85, U for up, D for down
  HAND(vexRT[Ch2] * 90/127);//up for out, down for in
  ROLLER(vexRT[Btn5U] * 120 - vexRT[Btn5D] * 120 + 20);//U for in, D for out

  if(vexRT[Btn7U] == 1) {auto_point=10;auto_side='l';}
  if(vexRT[Btn7D] == 1) {auto_point=10;auto_side='r';}
  if(vexRT[Btn7L] == 1) {auto_point=20;auto_side='l';}
  if(vexRT[Btn7R] == 1) {auto_point=20;auto_side='r';}
  if(vexRT[Btn8L] == 1) auto_cone=2;
  if(vexRT[Btn8R] == 1) auto_cone=3;

  /*if(vexRT[Btn8L] == 1) HAND_AUTO(1);
  if(vexRT[Btn8R] == 1) HAND_AUTO(2);

  if(vexRT[Btn5U] == 1) test1+=10;
  else if(vexRT[Btn5D] == 1) test1-=10;
  else if(vexRT[Btn6U] == 1) test2+=10;
  else if(vexRT[Btn6D] == 1) test2-=10;
  wait1Msec(50);*/
}
```

h. Functional element program for each driven section

```
void TAKER(short speed)
{
  if(Limit == 1 && speed <= 0) Taker = -25;
  else Taker = speed;
}

void WHEEL(short l,short r)
{
    Left = l;
    Right = r;
}

void ARM(short speed)
{
  /*if (Pot_Arm < 700 && speed <= 0) Arm = -30;
  else*/ Arm = speed;
}
```

```
void HAND(short speed)
{
  switch(mode)
  {
    case 1:
      if(vexRT[Btn8U] == 1 &&  Pot_Hand < 3000) Hand = 25;
      else if(speed <= 0 && Pot_Hand < 1350)  Hand = 15;
      else if(speed <= 0 && Pot_Hand <= 2220) Hand = speed * 0.8;
      else Hand = speed;
      //if (speed<=0 && Pot_Hand >= 1000) Hand = speed * 0.7;
      break;

    case 2:
      if(speed <= 0 &&Pot_Hand > 2800) Hand = 16;
      else if(speed >= 0 && Pot_Hand < 1150) Hand = -20;
      else Hand = speed;
      break;
  }
}
```

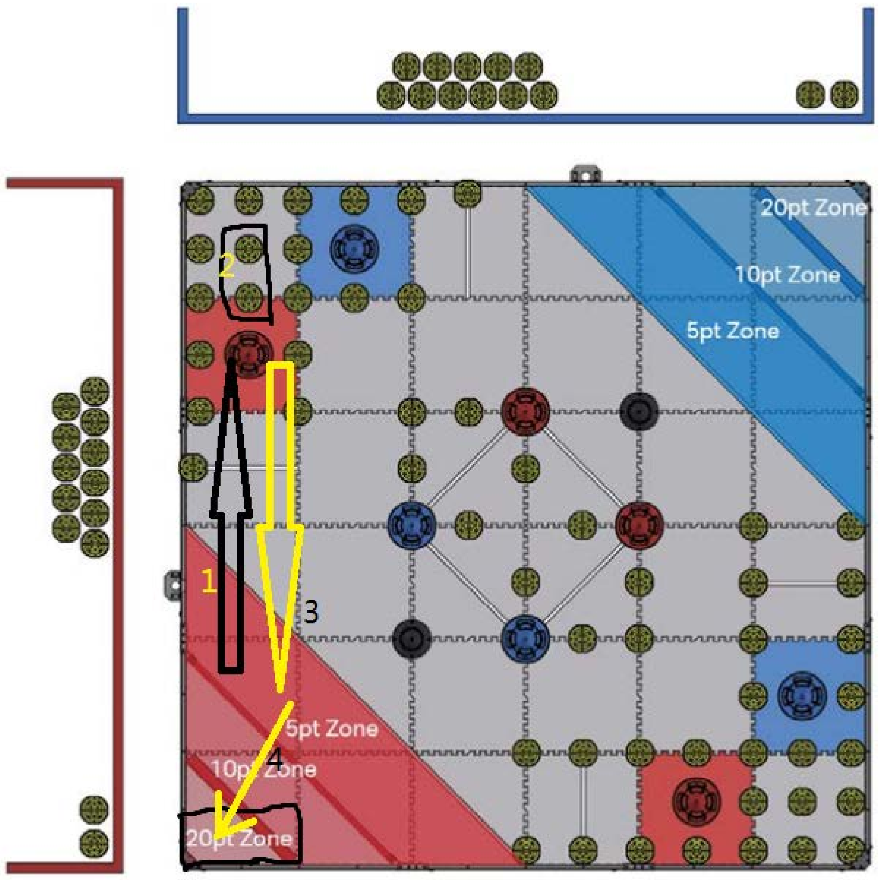
```
void HAND_AUTO(short auto_mode)
{
  switch(auto_mode)
  {
    case 1:
      Hand = 100;
      while(Pot_Hand < test1);
      Hand = -15;
      wait1Msec(100);
      Hand = 0;
      break;

    case 2:
      Hand = -90;
      while(Pot_Hand > test2);
      Hand = 15;
      wait1Msec(100);
      Hand = 0;
      break;
  }
}

void ROLLER(short speed)
{
  Roller = speed;
}
```



```
void MOVE(short l,short r,short distance)
{
  Ecd = 0;
  WHEEL(l,r);
  while(abs(Ecd) < distance);//while
  WHEEL(-l,-r);wait1Msec(50);
  WHEEL(0,0);wait1Msec(100);
}

void MOVE_START(short l,short r,short distance)
{
  Ecd = 0;
  WHEEL(l,r);//while
  while(abs(Ecd) < distance);
}

void MOVE_END(short l,short r,short distance)
{
  while(abs(Ecd) < distance);//while
  WHEEL(-l,-r);wait1Msec(50);
  WHEEL(0,0);wait1Msec(100);
}


void TURN(short l,short r,short direction)
{
  short avg = (abs(l)+abs(r))/2;
  //Gyro = 0;
  //while(abs(Gyro) < 10 * direction) WHEEL(l,r);
  if(l < r) while(Gyro < 10 * direction - avg) WHEEL(l,r);
  else while(Gyro > 10 * direction + avg) WHEEL(l,r);
  WHEEL(-l,-r);wait1Msec(50);
  WHEEL(0,0);wait1Msec(100);
}
```

2. Automation Control Programming (EXAMPLE: OUR ROBOT)

All the code below is written for our robot in 2018 VEX World Championships. We’ve considered all the situations we could encounter in the game. Our task is to take the goal, stack 2 or 3 cones on the goal and move the goal to the 10 or 20 points zone.



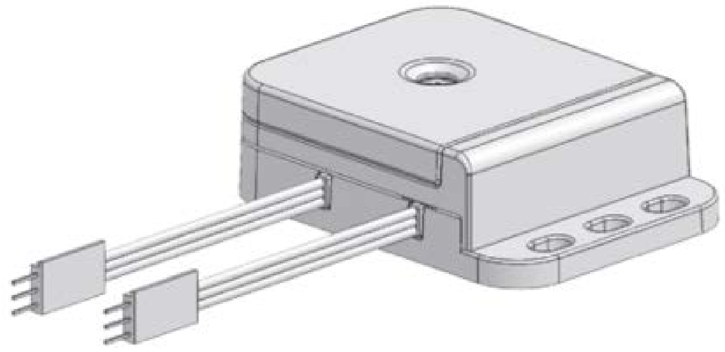
```
void auto(short point,short cone,char side)
{
  if(point == 0) return;
  Gyro = 0;

  ROLLER(30);TAKER(110);ARM(50);
  Ecd=0;WHEEL(70,70);while(Pot_Arm < 1730);ARM(5);
  MOVE_END(70,70,1300);//move to the goal
  TAKER(-100);//take the goal
  HAND_AUTO(1);
  while(Limit == 0);//wait the taker
  TAKER(0);//done

  ARM(-80);wait1Msec(100);ROLLER(-100);wait1Msec(250);//stack the 1st cone
  ROLLER(100);HAND(90);MOVE(70,80,50);//move to the 2nd cone
  while(Pot_Hand < 3300);wait1Msec(300);
  ROLLER(30);ARM(100);HAND(-90);wait1Msec(150);
  ARM(0);HAND_AUTO(2);
  ARM(-50);wait1Msec(100);ROLLER(-100);wait1Msec(250);//stack the 2nd cone
  switch(cone)
  {
    case 2:
      ROLLER(0);
      MOVE(-80,-80,1400);//get back
      break;

    case 3:
      ARM(80);wait1Msec(100);
      ARM(-50);ROLLER(100);HAND(90);MOVE(70,80,200);//move to the 3nd cone
      while(Pot_Hand < 3300);wait1Msec(300);
      ROLLER(30);ARM(80);HAND(-90);wait1Msec(200);
      ARM(0);HAND_AUTO(2);
      ARM(-50);wait1Msec(100);ROLLER(-100);wait1Msec(250);//stack the 3nd cone
      ROLLER(0);
      MOVE(-80,-80,1600);//get back
      break;
  }


  switch(point)
  {
    case 20:
      switch(side)
      {
        case 'r':TURN(80,-80,-45);MOVE(-80,-80,500);TURN(80,-80,-135);break;
        case 'l':TURN(-80,80,45);MOVE(-80,-80,500);TURN(-80,80,135);break;
      }


      WHEEL(90,90);TAKER(60);ARM(100);ROLLER(-50);wait1Msec(200);//put the goal
      ARM(0);wait1Msec(1500);
      TAKER(0);WHEEL(0,0);
      WHEEL(-100,-100);wait1Msec(400);
      switch(side)
      {
        case 'r':TURN(0,-100,-180);break;
        case 'l':TURN(-100,0,180);break;
      }
      WHEEL(-100,-100);wait1Msec(200);
      WHEEL(0,0);
      break;


    case 10:
      switch(side)
      {
        case 'r':TURN(40,-90,-135);break;
        case 'l':TURN(-90,40,135);break;
      }

      WHEEL(80,80);ARM(100);ROLLER(-50);wait1Msec(400);//put the goal
      WHEEL(0,0);ARM(0);TAKER(100);wait1Msec(800);
      TAKER(0);
      WHEEL(-80,-80);wait1Msec(500);
      WHEEL(0,0);
      break;
  }
  auto_point = 0;
  auto_cone = 0;
}
```

3. Optical Shaft Encoder

a. Theory

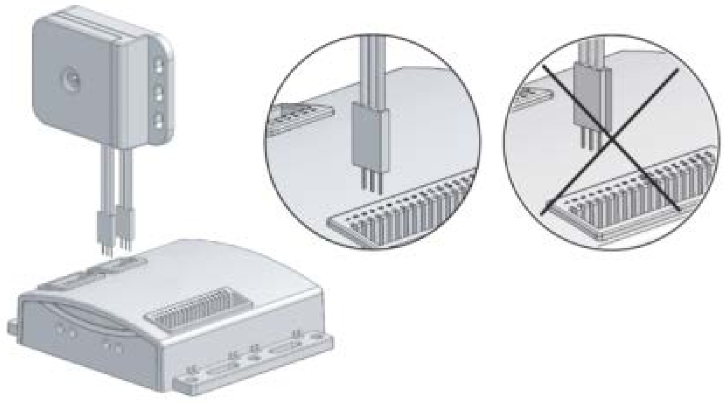
Basic Optical Shaft Encoders are commonly used for position and motion sensing. Basically, a disc with a pattern of cutouts around the circumference is positioned between an LED and a light detector; as the disc rotates, the light from the LED is blocked in a regular pattern. This pattern is processed to determine how far the disc has rotated. If the disc is then attached to a wheel on a robot, it is possible to determine the distance that wheel traveled, based on the circumference of the wheel and the number of revolutions it made.

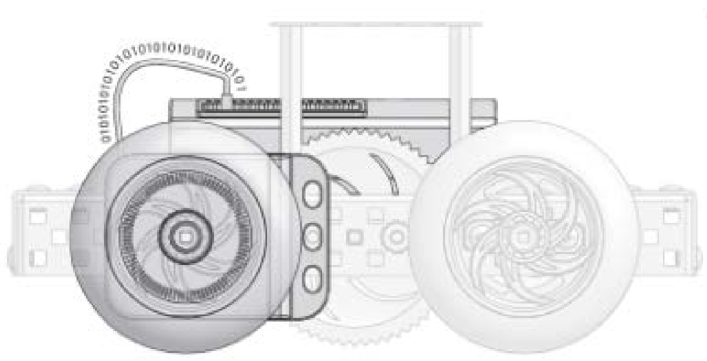
With the Quadrature Encoder, there are 2 output channels. Only one output can be used as a basic Optical Shaft Encoder. The term quadrature refers to the situation where there are two output channels; that is, two square waves 90 degrees out of phase with each other, being outputted by the unit. The two output channels of the Quadrature Encoder can be used to indicate both position and direction of rotation.



b. Reprogramming the Microcontroller to read the sensor

Shaft encoder can be plugged into any port in the Interrupt bank on a VEX Microcontroller. Depending on your specific application, you may be able to use any port in the Analog /Digital bank.





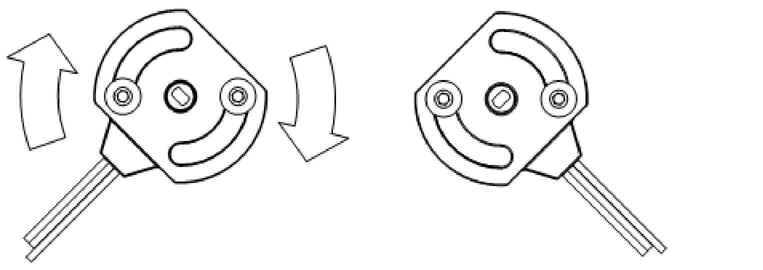
c. Sample program: Movement by Rotation (Advanced)

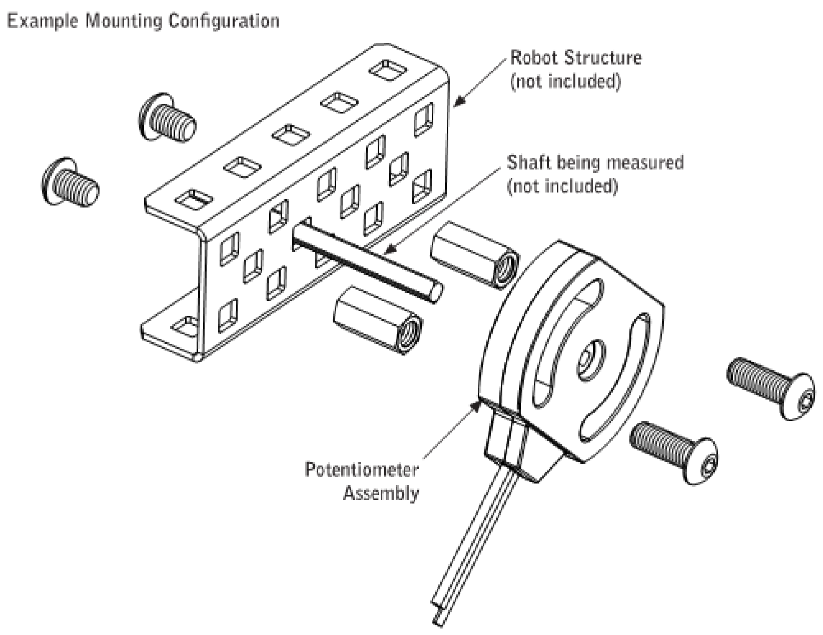
```
#pragma config(Sensor, dgtl1,  rightEncoder,        sensorQuadEncoder)
#pragma config(Sensor, dgtl3,  leftEncoder,         sensorQuadEncoder)
#pragma config(Motor,  port2,           rightMotor,    tmotorNormal, openLoop, reversed)
#pragma config(Motor,  port3,           leftMotor,     tmotorNormal, openLoop)

// Functions Prototypes
void Forward(float r);
void TurnLeft(float r);

// Declare Global Variables
const float _rotations = 360.0;

task main()
{
  wait1Msec(2000);          // Wait 2000 milliseconds before continuing.

  int i;
  for(i=0; i<4; i++)        // While 'i' is less than 4:
  {
    Forward(3.0);           // Call function 'Forward(float)' and pass the float value '3.0' through.
    TurnLeft(1.3);          // Call function 'TurnLeft(float)' and pass the float value '1.3' through.
  }
}

void Forward(float r)
{
  SensorValue[rightEncoder] = 0;    /* Clear the encoders for    */
  SensorValue[leftEncoder]  = 0;    /* consistancy and accuracy. */

  // While the encoders have not yet met their goal: (r * _rotations) ie (3.0 * 360.0) or "three _rotations"
  while(SensorValue[rightEncoder] < (r * _rotations) && SensorValue[leftEncoder] < (r * _rotations))
  {
    motor[rightMotor] = 63;         /* Run both motors        */
    motor[leftMotor]  = 63;         /* forward at half speed. */
  }
  motor[rightMotor] = 0;            /* Stop both motors          */
  motor[leftMotor]  = 0;            /* can act independantly as a "chunk" of code, without any loose ends. *
}

void TurnLeft(float r)
{
  SensorValue[rightEncoder] = 0;    /* Clear the encoders for    */
  SensorValue[leftEncoder]  = 0;    /* consistancy and accuracy. */

  // While the encoders have not yet met their goal: (left is compared negativly since it will in reverse)
  while(SensorValue[rightEncoder] < (r * _rotations) && SensorValue[leftEncoder] > (-1 * r * _rotations))
  {
    motor[rightMotor] = 63;         // Run the right motor forward at half speed
    motor[leftMotor]  = -63;        // Run the left motor backward at half speed
  }
  motor[rightMotor] = 0;            /* Stop both motors!  This is important so that each function       *
  motor[leftMotor]  = 0;            /* can act independantly as a "chunk" of code, without any loose ends. *
}
```

4. Potentiometer

a. Theory

A potentiometer (or "pot") is an electrical device used to measure angular position. The user can therefore adjust the degree to which the potentiometer opposes electric current through it, simply by turning a shaft that is attached to the center of the potentiometer. As the resistance of the potentiometer changes, so does the voltage, which thus causes the potentiometer to act as a variable voltage divider. This varying voltage can be measured by the VEX microcontroller and is directly proportional to the angular position of the shaft connected to the center of the potentiometer. This allows you to obtain an analog measurement of an angular position. The VEX potentiometer is designed with a “D-hole” in the center, which should slide easily over the VEX square shafts. The potentiometer includes two arcs, each ½in from the center hole; these arcs exist to assist with mounting the potentiometer to the robot structure.





b. Sample Program: Arm Control via Potentiometer

```
#pragma config(Sensor, in5,     ,                sensorPotentiometer)

#define Pot_Hand SensorValue[in5]
```

```
void HAND(short speed)
{
  switch(mode)
  {
    case 1:
      if(vexRT[Btn8U] == 1 &&  Pot_Hand < 3000) Hand = 25;
      else if(speed <= 0 && Pot_Hand < 1350)  Hand = 15;
      else if(speed <= 0 && Pot_Hand <= 2220) Hand = speed * 0.8;
      else Hand = speed;
      //if (speed<=0 && Pot_Hand >= 1000) Hand = speed * 0.7;
      break;

    case 2:
      if(speed <= 0 &&Pot_Hand > 2800) Hand = 16;
      else if(speed >= 0 && Pot_Hand < 1150) Hand = -20;
      else Hand = speed;
      break;
  }
}
```

5. Limit Switch

a. Theory

Limit Switch Sensor Signal: The limit switch sensor is a physical switch. It can tell the robot whether the sensor's metal arm is being pushed down or not.

Switch Type: SPDT microswitch, configured for SPST Normally Open behavior. Behavior: When the limit switch is not being pushed in, the sensor maintains a digital HIGH signal on its sensor port. This High signal is coming from the Microcontroller. When an external force (like a collision or being pressed up against a wall) pushes the switch in, it changes its signal to a digital LOW until the limit switch is released. An unpressed switch is indistinguishable from an open port.

Benefits: Limit switches expand the functionality of robots by allowing controlled motion in moving components (e.g., gripper arm). They also allow the robot to better detect its surroundings, by detecting collisions with external objects.

b. Sample Program: Wait for Release

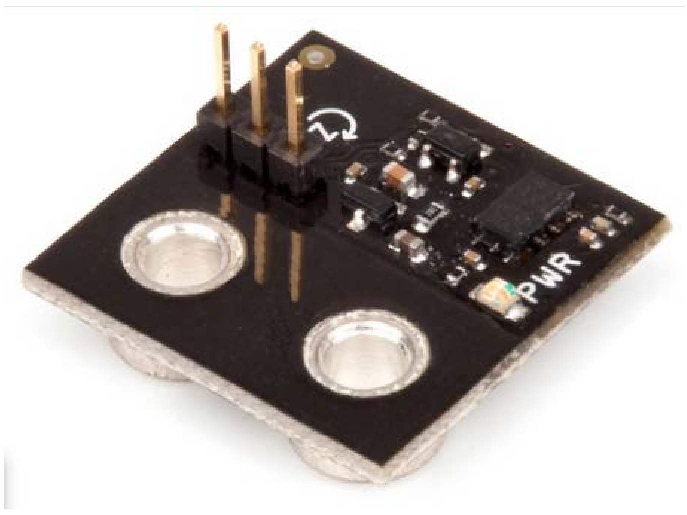
This program instructs the robot to move forward at half speed until the bumper sensor is released. There is a two second pause at the beginning of the program.

```
#pragma config(Sensor, dgtl6,  touchSensor,         sensorTouch)
#pragma config(Motor,  port2,           rightMotor,    tmotorNormal, openLoop, reversed)
#pragma config(Motor,  port3,           leftMotor,     tmotorNormal, openLoop)

task main()
{
  wait1Msec(2000);

  while(SensorValue(touchSensor) == 0)
  {
  }

  wait1Msec(250);

  while(SensorValue(touchSensor) == 1)
  {
    motor[rightMotor] = 63;
    motor[leftMotor]  = 63;
  }
}
```

6. Yaw Rate Gyroscope Sensor

a. Theory

Connect the Yaw Rate Gyro to an analog input on the VEX Microcontroller using a standard servo extension cable. The black (ground) wire goes adjacent to the 'B' silkscreened on the board. The center wire is for +5 volts. The white wire is signal. The mounting holes are electrically isolated from the circuit.



b. Sample Program: Gyro Based Turn

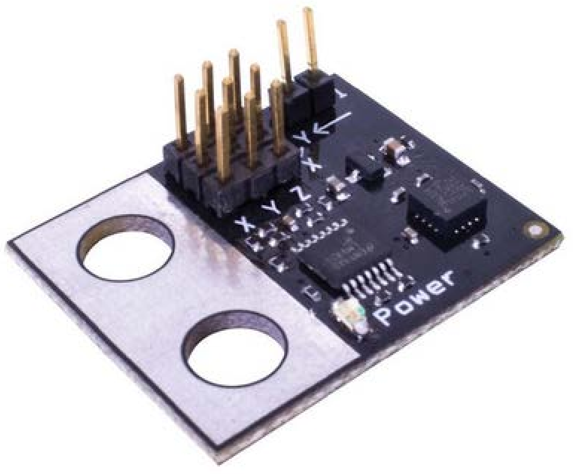
```
#pragma config(Motor,  port2,           rightMotor,    tmotorNormal, openLoop)
#pragma config(Motor,  port3,           leftMotor,     tmotorNormal, openLoop, reversed)
```

```
task main()
{
  //Completely clear out any previous sensor readings by setting the port to "sensorNone"
  SensorType[in8] = sensorNone;
  wait1Msec(1000);
  //Reconfigure Analog Port 8 as a Gyro sensor and allow time for ROBOTC to calibrate it
  SensorType[in8] = sensorGyro;
  wait1Msec(2000);

  //Specify the number of degrees for the robot to turn (1 degree = 10, or 900 = 90 degrees)
  int degrees10 = 900;
  //Specify the amount of acceptable error in the turn
  int error = 5;

  //While the absolute value of the gyro is less than the desired rotation - 100...
  while(abs(SensorValue[in8]) < degrees10 - 100)
  {
    motor[rightMotor] = 50;
    motor[leftMotor] = -50;
  }
  //Brief brake to eliminate some drift
  motor[rightMotor] = -5;
  motor[leftMotor] = 5;
  wait1Msec(100);

  //Second while loop to move the robot more slowly to its goal, also setting up a range
  //for the amount of acceptable error in the system

  while(abs(SensorValue[in8]) > degrees10 + error || abs(SensorValue[in8]) < degrees10 - error)
  {
    if(abs(SensorValue[in8]) > degrees10)
    {
      motor[rightMotor] = -30;
      motor[leftMotor] = 30;
    }
    else
    {
      motor[rightMotor] = 30;
      motor[leftMotor] = -30;
    }
  }
  //Stop
  motor[rightMotor] = 0;
  motor[leftMotor] = 0;
  wait1Msec(250);
}
```

7. Analog Accelerometer

a. Introduction

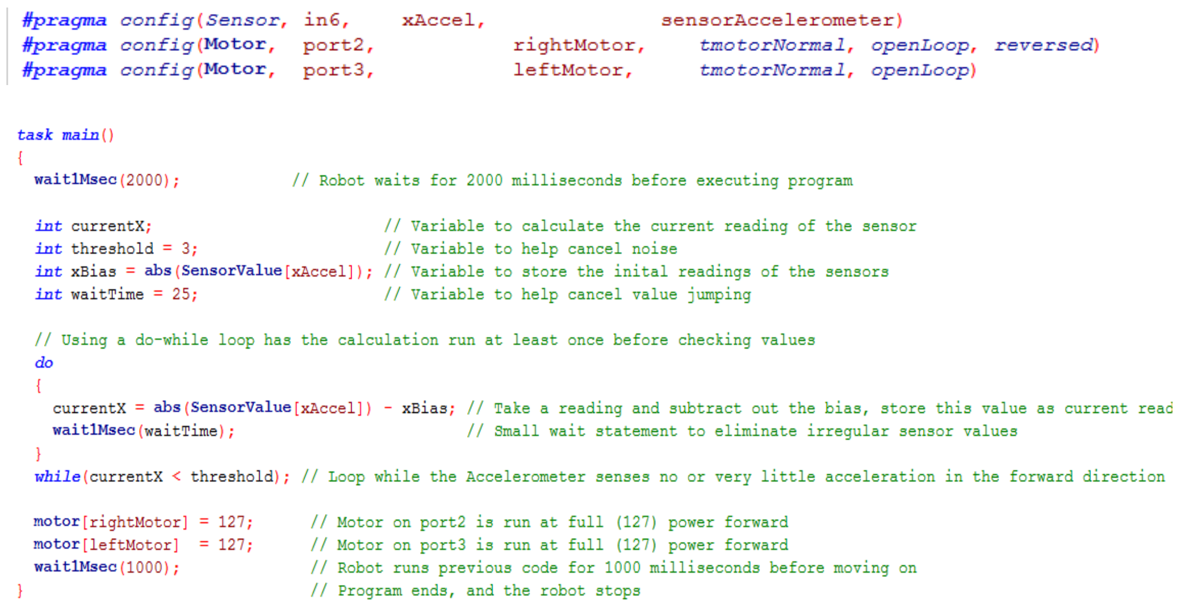
Measures accelerations on three axes simultaneously. By measuring the acceleration of the robot, one can calculate the velocity of the robot or the distance this robot has traveled. Accelerometers are also great for detecting collisions and determining if the robot is stopped or moving.



b. Theory

The LIS344ALH capacitive micromachined accelerometer features signal conditioning, a 1-pole low pass filter, temperature compensation and g Select which allows for the selection among 2 sensitivities. Zero-g offset full scale span and filter cut-off are factory set and require no external devices. The sensor will measure acceleration in both directions along each of the 3 axis. Acceleration along the X or Y axis in the direction of the silkscreened arrows will produce a larger reading, while acceleration in the opposite direction will produce a smaller reading. For the Z axis, upward acceleration (in the direction of the board's face) produces larger values, and downward acceleration (toward the board's back) produces lower values.

c. Sample program: Wait for Acceleration



```
#pragma config(Sensor, in6,     xAccel,              sensorAccelerometer)
#pragma config(Motor,  port2,           rightMotor,    tmotorNormal, openLoop, reversed)
#pragma config(Motor,  port3,           leftMotor,     tmotorNormal, openLoop)


task main()
{
  wait1Msec(2000);            // Robot waits for 2000 milliseconds before executing program

  int currentX;                         // Variable to calculate the current reading of the sensor
  int threshold = 3;                    // Variable to help cancel noise
  int xBias = abs(SensorValue[xAccel]); // Variable to store the inital readings of the sensors
  int waitTime = 25;                    // Variable to help cancel value jumping

  // Using a do-while loop has the calculation run at least once before checking values
  do
  {
    currentX = abs(SensorValue[xAccel]) - xBias; // Take a reading and subtract out the bias, store this value as current read
    wait1Msec(waitTime);                         // Small wait statement to eliminate irregular sensor values
  }
  while(currentX < threshold); // Loop while the Accelerometer senses no or very little acceleration in the forward direction

  motor[rightMotor] = 127;      // Motor on port2 is run at full (127) power forward
  motor[leftMotor]  = 127;      // Motor on port3 is run at full (127) power forward
  wait1Msec(1000);              // Robot runs previous code for 1000 milliseconds before moving on
}                               // Program ends, and the robot stops
```

8. Ultrasonic Range Finder

a. Introduction

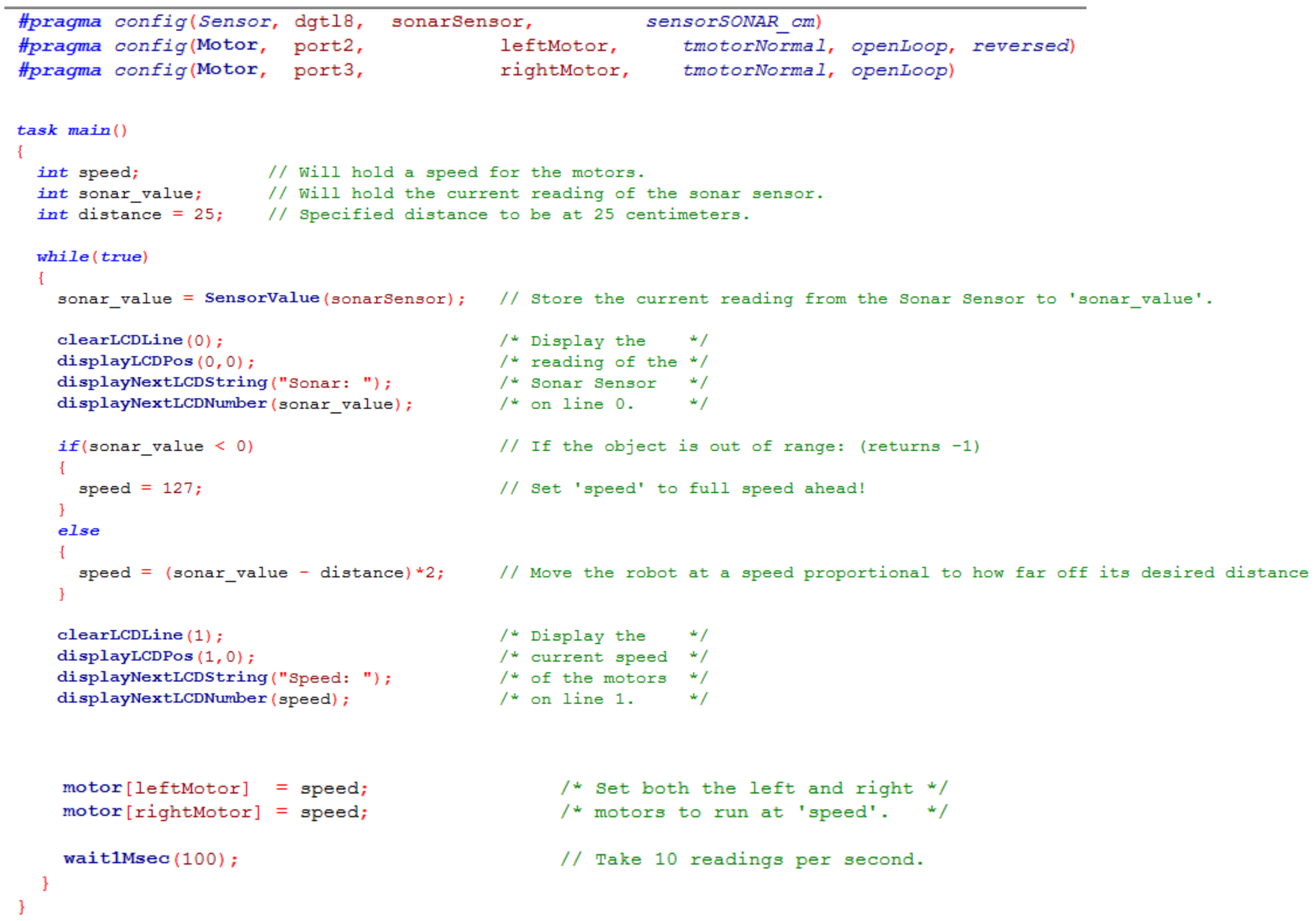
Avoid obstacles and measure distances with the ultrasonic range finder. Device emits a high-frequency sound wave that alerts the robot to things in its path. A Programming Kit is needed to change the program in the VEX Controller.

b. Theory

An ultrasonic range finder sensor enables a robot to detect obstacles in its path by utilizing the propagation of high-frequency sound waves. The sensor emits a 40kHz sound wave, which bounces off a reflective surface and returns to the sensor. Then, using the amount of time it takes for the wave to return to the sensor, the distance to the object can be computed. To increase the sensing range, the sensor can be mounted to a servo to allow it to rotate.

c. Sample program: Smooth Sonar Tracking

This program uses the Ultrasonic sensor to detect objects, and maintain a desired distance from that object.



```
#pragma config(Sensor, dgtl8,  sonarSensor,         sensorSONAR_cm)
#pragma config(Motor,  port2,           leftMotor,     tmotorNormal, openLoop, reversed)
#pragma config(Motor,  port3,           rightMotor,    tmotorNormal, openLoop)


task main()
{
  int speed;            // Will hold a speed for the motors.
  int sonar_value;      // Will hold the current reading of the sonar sensor.
  int distance = 25;    // Specified distance to be at 25 centimeters.

  while(true)
  {
    sonar_value = SensorValue(sonarSensor);   // Store the current reading from the Sonar Sensor to 'sonar_value'.

    clearLCDLine(0);                          /* Display the    */
    displayLCDPos(0,0);                       /* reading of the */
    displayNextLCDString("Sonar: ");          /* Sonar Sensor   */
    displayNextLCDNumber(sonar_value);        /* on line 0.     */

    if(sonar_value < 0)                       // If the object is out of range: (returns -1)
    {
      speed = 127;                            // Set 'speed' to full speed ahead!
    }
    else
    {
      speed = (sonar_value - distance)*2;     // Move the robot at a speed proportional to how far off its desired distance
    }

    clearLCDLine(1);                          /* Display the    */
    displayLCDPos(1,0);                       /* current speed  */
    displayNextLCDString("Speed: ");          /* of the motors  */
    displayNextLCDNumber(speed);              /* on line 1.     */


    motor[leftMotor]  = speed;                /* Set both the left and right */
    motor[rightMotor] = speed;                /* motors to run at 'speed'.   */

    wait1Msec(100);                           // Take 10 readings per second.
  }
}
```