MS4SSA
Robotics Module:
Programming and Sensors

Kim Hollan
Worcester Polytechnic Institute
Why Program a Robot?

• Building a robot teaches many valuable skills; however, the learning doesn’t stop there

• Programming also teaches valuable life skills
  — Problem Solving
  — Creative and Computational Thinking
  — Team Building

• Robotics provides hands-on activities that help stimulate thinking, excite and engage students

• Robots help students see how what they are learning has a direct impact on the world – and how the math and engineering elements can help guide solutions for real life problems
System Components

- VEX Microcontroller
- VEX Joystick
- Sensors
- VEXnet Key
- USB Tether Cable
- Actuators (motors and servos)

Worcester Polytechnic Institute
What is a Program?

- Programs are steps, or instructions that you want the robot to follow

STEPS

- Start driving forward
- Wait 2 seconds
- Turn for 1000ms

CODE

```c
motorSet(1, 60);
motorSet(10, -60);
delay(2000);
motorSet(1, -60);
delay(1000);
```

- There are many different options for languages to use for programming. Today we are using C programming language and Purdue Robotics Operating System
Functions

• A name given to a group of steps so you can refer to it later
• Makes the program more readable
• Simplifies multiple lines of code into a single command
• The steps for a function appear inside of braces 
• The steps can optionally produce a result or just do some work
Functions

• For the program today, we created these functions:

```c
void driveForward(int time){
    motorSet(1,60);
    motorSet(10,-60);
    delay(time);
    motorStopAll();
}

void turnRight(int time){
    motorSet(1,60);
    motorSet(10,60);
    delay(time);
    motorStopAll();
}
```
Your First Challenge

You start with this program...

and make the robot drive in a square instead
void driveForward(int time){
    motorSet(1,60);
    motorSet(10,-60);
    delay(time);
    motorStopAll();
}

void turnRight(int time){
    motorSet(1,60);
    motorSet(10,60);
    delay(time);
    motorStopAll();
}

void operatorControl() {
    while (1) {
        driveForward(1000);
        delay(500);
        turnRight(500);
        delay(500);
    }
}
• Now, your job is to change the program to make the robot turn in a square
Your Second Challenge

Drive until 10cm from wall, then stop
Use a Sensor

Sensors allow the robot to understand its state and the world around it.

Ultrasonic rangefinder gives the distance to an object in centimeters.
Rangefinder

Ultrasonic sonar;

```c
void operatorControl() {
    sonar = ultrasonicInit(9, 8);
    while(1){
        if(ultrasonicGet(sonar)>10){
            motorSet(1,60);
            motorSet(10,-60);
            delay(100);
        }
        else{
            motorStopAll();
        }
    }
}
```
What went wrong?

• Inertia carries robot past 10 cm
• What can we do?
  – Stop 12cm from wall to allow 2cm of coasting?
• What’s wrong with this strategy?
  – Differences in battery charge
  – Differences in driving surface
  – Differences in slope
  – etc...

Use proportional control!
Proportional Control

We want the robot to stop 10cm from the wall so the target distance or set point is 10.

Compute the distance from the set point and call it the error.

<table>
<thead>
<tr>
<th>Distance from Set Point</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>24cm</td>
<td>14</td>
</tr>
<tr>
<td>16cm</td>
<td>6</td>
</tr>
<tr>
<td>12cm</td>
<td>2</td>
</tr>
<tr>
<td>10cm</td>
<td>0</td>
</tr>
<tr>
<td>8cm</td>
<td>-2</td>
</tr>
</tbody>
</table>

Make the robot driving speed proportional to the error. As the error gets smaller, the robot drives more slowly. The speed is a function of the error.
These error values are too small to make the motors move.

Solution:
We can multiply the values by some constant (Kp) to make the values big enough to drive the motors for example: Kp = 3
What if \( K_p \) is too small?

\[ K_p = 1 \]

Error = 16  
Speed = 16
What if $K_p$ is too small?

$K_p = 1$

error = 3
speed = 3
What if Kp is too small?

Kp = 1

error=2
speed=2

A speed of 2 is too little to make the motors turn
What if $K_p$ is too small?

$K_p = 1$

error = 2
speed = 2

The robot never reaches the set point!
What if $K_p$ is too large?

$K_p = 10$

error = 24
speed = 240
What if $K_p$ is too large?

$K_p = 10$

error = 5
speed = 50
What if Kp is too large?

Kp = 10

error = 2
speed = 20
What if Kp is too large?

Kp = 10

But we’re going so fast that the robot can’t stop.
What if $K_p$ is too large?

$K_p = 10$

Now the value is negative, and big, so the robot starts backing up at high speed.
What if Kp is too large?

Kp = 10

error = 2
speed = 20

Now the value is big and positive again, so the robot starts driving forwards fast.
Finding the right value

- Keep increasing Kp until the system oscillates then back it down a little

- There are other techniques, look online at PID control
Adding Proportional Control

```c
void operatorControl() {
    sonar = ultrasonicInit(9, 8);
    int Kp = 2;
    int setPoint = 10;
    ultrasonicGet(sonar);
    delay(1000);
    while(1) {
        int error = ultrasonicGet(sonar) - setPoint;
        driveWheels(Kp * error);
        delay(50);
    }
}
```

Compute the distance from the set point (error)

Drive at a speed proportional to the set point