Programming and Sensors

Brad Miller
Associate Director, WPI Robotics Resource Center
System Components

- Arduino
- Actuators (motors and servos)
- Sensors
- Controller
Arduino

- Inexpensive and easy to use prototyping platform for electronics projects
- Programmed in C or C++
- Two models at WPI
  - Uno in student take-home kits
  - Mega in the lab kit
## Arduino Specifications

<table>
<thead>
<tr>
<th></th>
<th>Mega 2560</th>
<th>Uno</th>
</tr>
</thead>
<tbody>
<tr>
<td>Digital I/O (including PWMs)</td>
<td>54</td>
<td>14</td>
</tr>
<tr>
<td>Analog inputs</td>
<td>16</td>
<td>6</td>
</tr>
<tr>
<td>PWM outputs</td>
<td>15</td>
<td>6</td>
</tr>
<tr>
<td>Serial ports</td>
<td>4</td>
<td>1</td>
</tr>
<tr>
<td>Flash memory</td>
<td>256Kb</td>
<td>32Kb</td>
</tr>
<tr>
<td>SRAM</td>
<td>8Kb</td>
<td>2Kb</td>
</tr>
</tbody>
</table>
Aduino Mega

USB Port

Power connector

PWM ports

Communications

Digital I/O

Analog inputs
RBE Shield

Notice Sig, PWR, GND references on connectors
Controller (gamepad)

XBee based controller

Links to receiver on the Arduino shield
Xbee module

XBeep module connects the Arduino to the gamepad. Must be matched pair.
Servos and Motors

• Motors: continuous rotation
• Servos: rotate about 100 degrees
• Both need speed controllers and connect to PWM ports
**Programming**

Programs are simply a series of steps that you want to robot to follow:

- Start driving forward
- Wait 2 seconds
- Stop driving
- Start turning left
- Wait 750 ms
- Move arm down
- Grab ball
- Move arm up

For these robots they are written in the C programming language:

```c
robot.drive(100);
delay(2000);
robot.stop();
robot.turn(100);
delay(750);
arm.moveto(30);
gripper.close();
arm.moveto(120);
```

...
Functions

• Giving a name to a group of steps so you can refer to it later
Arduino programs

• The code inside the setup() function is run once at the start of the program

• The code inside the loop() function is run repeatedly
Your first challenge
You start with this program

and make the robot drive in a square instead
RobotDrive object

#include <RobotDrive.h>

const int leftMotorPin = 5;
const int rightMotorPin = 4;

RobotDrive robot(leftMotorPin, rightMotorPin);
RobotDrive object

```c
void setup()
{
    robot.setup();
}
```

Set up the RobotDrive to work
Then the driving

```c
void loop() {
    robot.drive(100);
    delay(1000);
    robot.stop();
    delay(1000);
}
```

Drive full speed forward and wait 1 second (1000ms)

...then stop the robot and wait 1 second
Add a turn

```java
void loop() {
    robot.drive(100);
    delay(1000);

    robot.stop();
    delay(1000);

    robot.turn(100);
    delay(800);

    robot.stop();
    delay(1000);
}
```

Start the robot doing a turn in place for 800ms

Then stop the robot for 1 second
Your second challenge

Drive until 10” from wall, then stop
Use a sensor

Ultrasonic rangefinder gives the distance to an object in inches

Sensors allow the robot to understand its state and the world around it
The driving setup is the same

```cpp
const int leftMotorPin = 5;
const int rightMotorPin = 4;

RobotDrive robot(leftMotorPin, rightMotorPin);
```
Add the rangefinder

```c
const int pingPin = 22;
const int echoPin = 2;

const int leftMotorPin = 5;
const int rightMotorPin = 4;

RobotDrive robot(leftMotorPin, rightMotorPin);

ultrasonic ultra(pingPin, echoPin);
```

The ping pin and the echo pin

Set up the ultrasonic rangefinder just like the RobotDrive
And drive while more than 10 inches

```cpp
void loop() {
    if (ultra.distance() > 10)
        robot.drive(50);
    else
        robot.stop();
}
```
What went wrong?

• Inertia carries robot past 10 inches

• What can we do?
  • Stop 12” from wall to allow 2” of coasting?

• What’s wrong with this strategy?
  • Differences in battery charge
  • Differences in driving surface
  • Differences in slope
  • etc…
Proportional Control

We want the robot to stop 10” from the wall so the **set point** is 10

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

<table>
<thead>
<tr>
<th>Distance</th>
<th>Error</th>
</tr>
</thead>
<tbody>
<tr>
<td>24”</td>
<td>14</td>
</tr>
<tr>
<td>16”</td>
<td>6</td>
</tr>
<tr>
<td>12”</td>
<td>2</td>
</tr>
<tr>
<td>10”</td>
<td>0</td>
</tr>
<tr>
<td>8”</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
Proportional Gain

These error values are too small to make the motors move.

We can multiply the values by some constant to make the values bigger.
Adding proportional control

```c
const int Kp = 1;

const int setPoint = 10;

void loop() {
    int error = ultra.distance() - setPoint;
    robot.drive(Kp * error);
    delay(50);
}
```

Compute the set point

Drive at a speed proportional to the set point
What if Kp is wrong

• What if Kp is too large?

• What if Kp is too small?

• We want you to try to find a value that’s correct

  • But look what happens if it’s too big and what happens if it’s too small