WPI

MS4SSA Robotics Module:

Programming and Sensors

Brad Miller and Kim Hollan
Worcester Polytechnic Institute

What are we doing today?

- Talk about why to program robots
- Learn about basic RobotC programming
- Learn how to make the robot move in predetermined ways
- Learn how to use a sensor to understand the robots surroundings
- Give you a taste of robot programming to experience the student excitement and engagement

Why Program a Robot?

MS4SSA
Math and Science for Sub-Saharan Africa

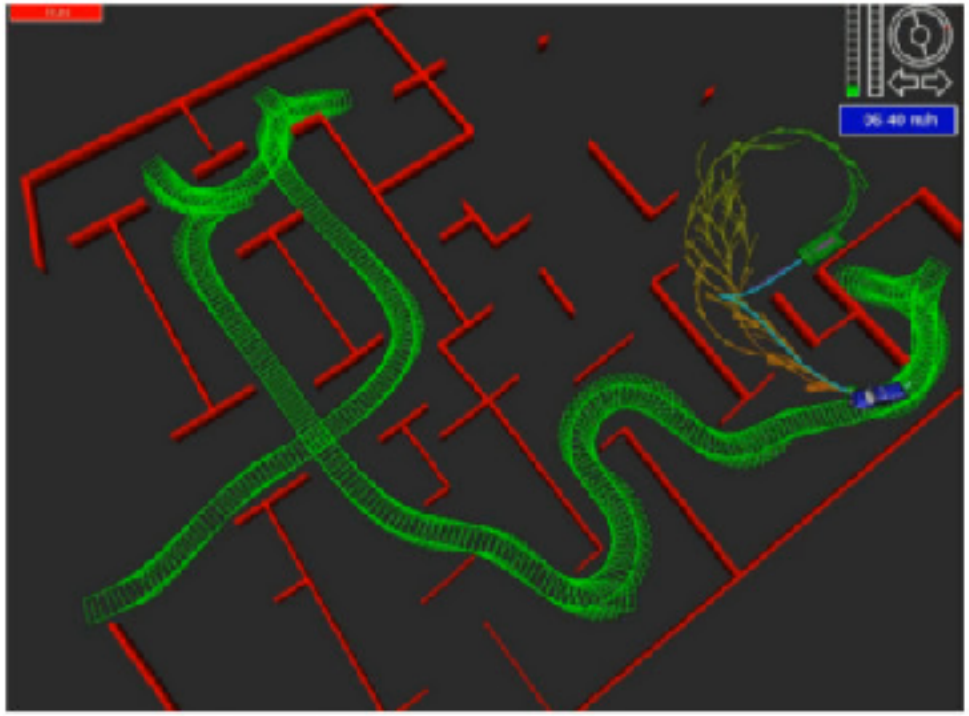
- Building a robot teaches many valuable skills; however, the learning doesn't stop there
- Programming also teaches valuable life skills
  - Problem Solving
  - Creative and Computational Thinking
  - Team Building
- Robotics provides hands-on activities that help stimulate thinking, excite and engage 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

Why Robotics

- Students often are asked to learn concepts that they might not see applications

- In Robotics, especially with competitions, students learn because they need the concepts to win

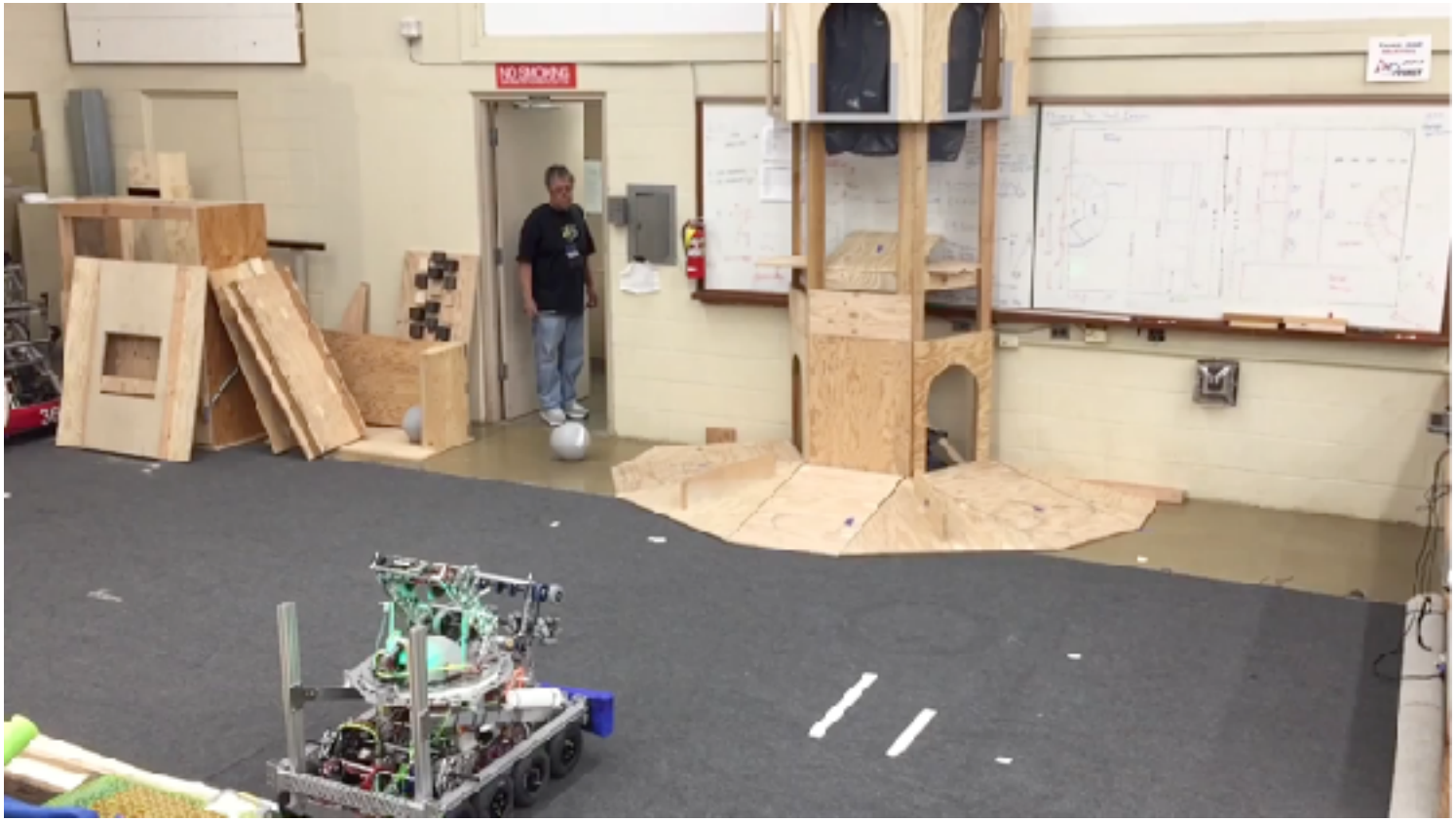
Why Robots?



Path planning requires trigonometry and programming

Why Robots?

Aiming a ball shooter uses a camera and needs trigonometry to position the robot and aim



Why robots?

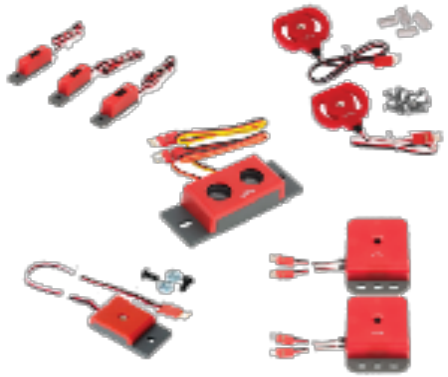
- Climbing robots requires torque, gear ratios, and speed calculations

System Components

MS4SSA
Math and Science for
Sub-Saharan Africa

VEX Microcontroller

VEX Joystick



Sensors

VEXnet Key

USB Tether Cable

Actuators
(motors and servos)

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
```

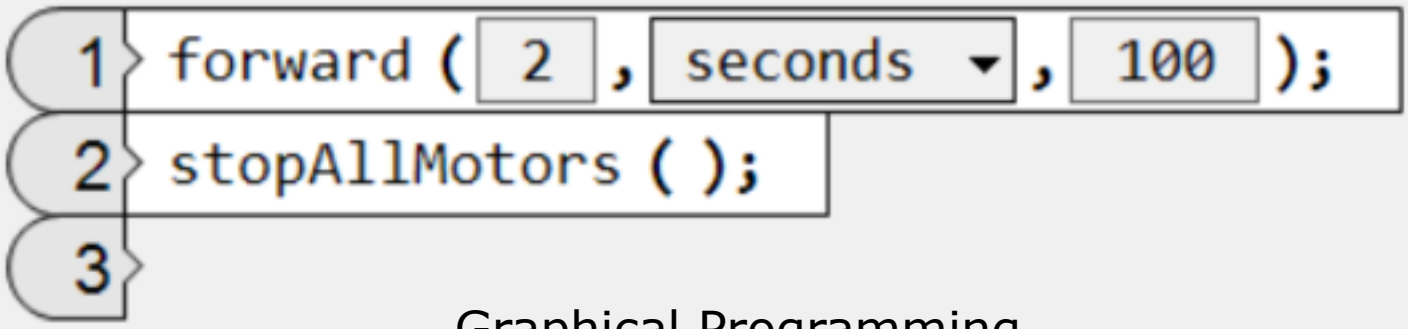
CODE

```
motor[leftMotor] = 100;
motor[rightMotor] = 100;
sleep(1000);

motor[leftMotor] = 0;
motor[rightMotor] = 0;
```

- There are many different programming languages. Today we are using the C language with RobotC

Making it easier



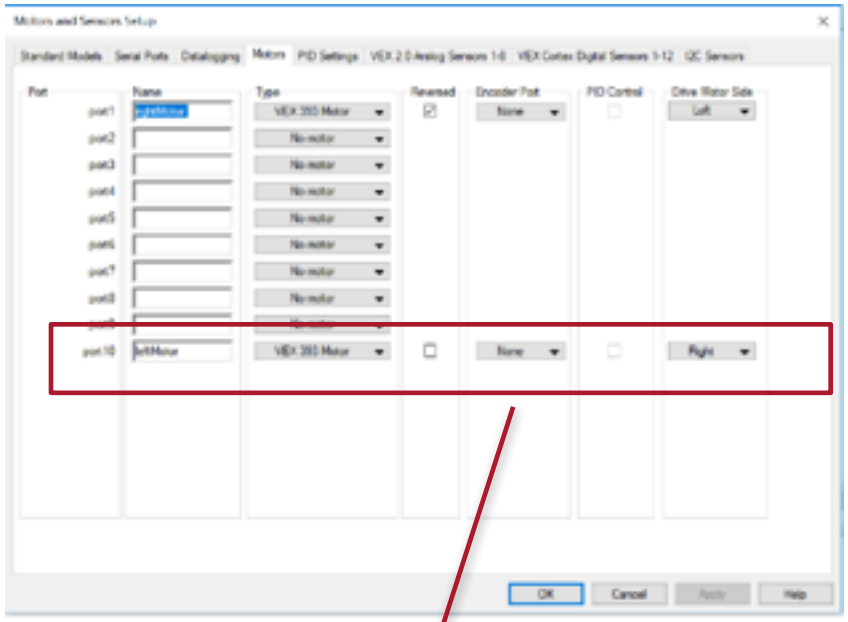
Graphical Programming

```
task main()
{
  forward(2, seconds, 100);
  stopAllMotors();
}
```

Text Programming

Setting motor speeds

Motors and Sensors Setup




```
motor[leftMotor] = 100;
motor[rightMotor] = -100;
```

Functions

- For the program today, we're using these functions:

forward(*time*, *units*, *motorSpeed*);
backward(*time*, *units*, *motorSpeed*);
delay(*milliseconds*);

turnLeft(*time*, *units*, *motorSpeed*);
turnRight(*time*, *units*, *motorSpeed*);

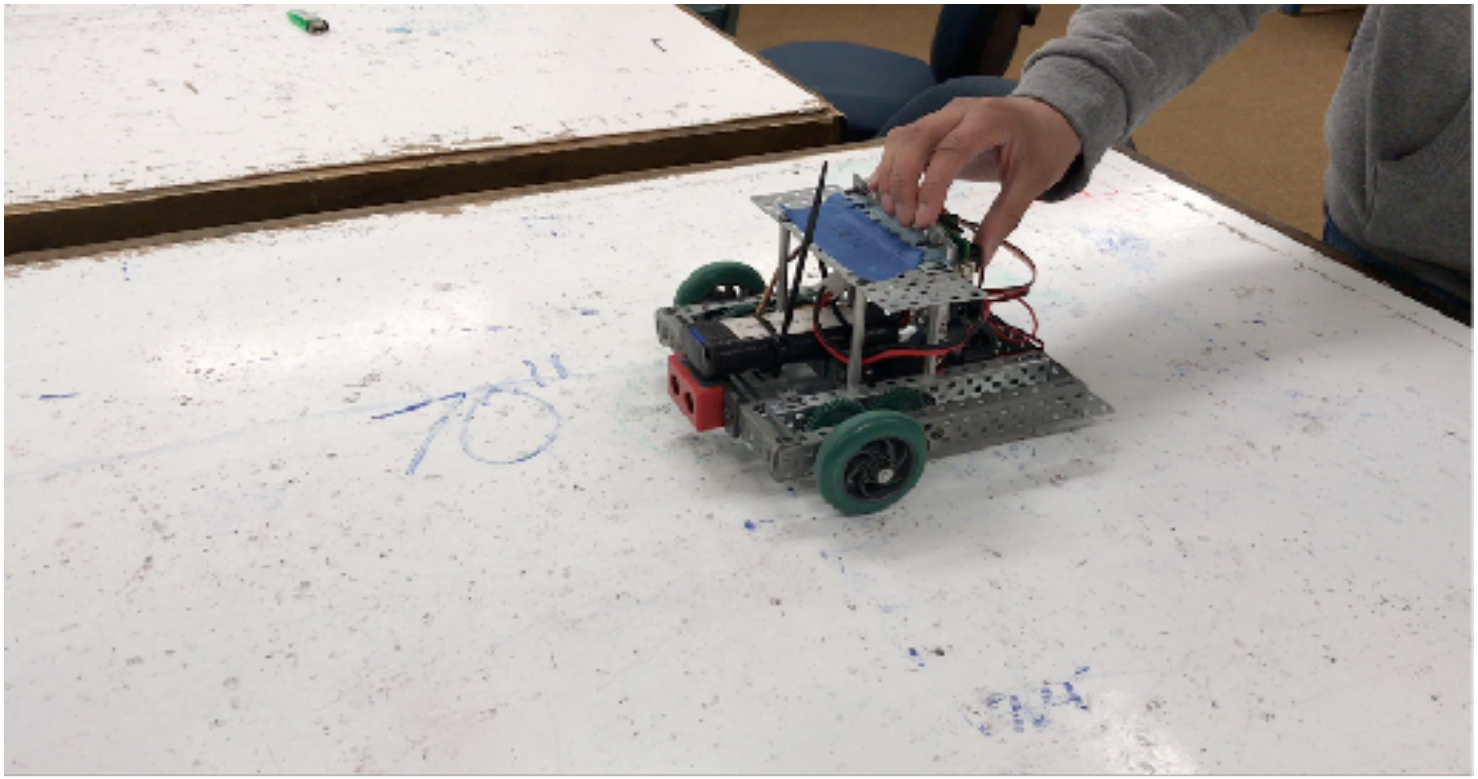
time in seconds or milliseconds
units is "seconds" or "milliseconds"
motorSpeed is -127 to 127
(0 = stopped)

Examples

```
forward(1, second, 100);
turnRight(500, milliseconds);
delay(2000);
```

Your First Challenge

You start with this program…



and make the robot drive in a square instead

RobotC programs

- A RobotC program starts with task main()
- Put your program in between the braces

```
task main()
{
    while (true) {
        forward(2, seconds, 100);
        turnRight(900, milliseconds, 60);
    }
}
```

Repeating some steps

- Use the word "while" followed by an expression to repeat a group of program steps
- Everything in the braces is repeated *while* the expression is true

```
task main()
{
    while (true) {
        forward(2, seconds, 100);
        turnRight(900, milliseconds, 60);
    }
}
```

Driving the robot

- Use the command, “forward” to make the robot drive forward
- You supply the time to drive, the units of time, and the speed (-127 full backwards, 0 stopped, and +127 full forwards)

```
task main()
{
    while (true) {
        forward(2, seconds, 100);
        turnRight(900, milliseconds, 60);
    }
}
```

Making the robot turn

- Use the command, “turnRight” to make the robot turn right
- You supply the time to turn, the units of time, and the speed (-127 full backwards, 0 stopped, and +127 full forwards)

```
task main()
{
    while (true) {
        forward(2, seconds, 100);
        turnRight(900, milliseconds, 60);
    }
}
```

Make a Square

- Now, your job is to change the program to make the robot turn in a square

Start with:

\documents\ms4ssa rawanda\drivingStraight

Drawing a square: a solution

Fill in the program here

How do forward and turnRight work?

- These are functions that are built-in to RobotC
- What they really do is control the left and right motors the way you said

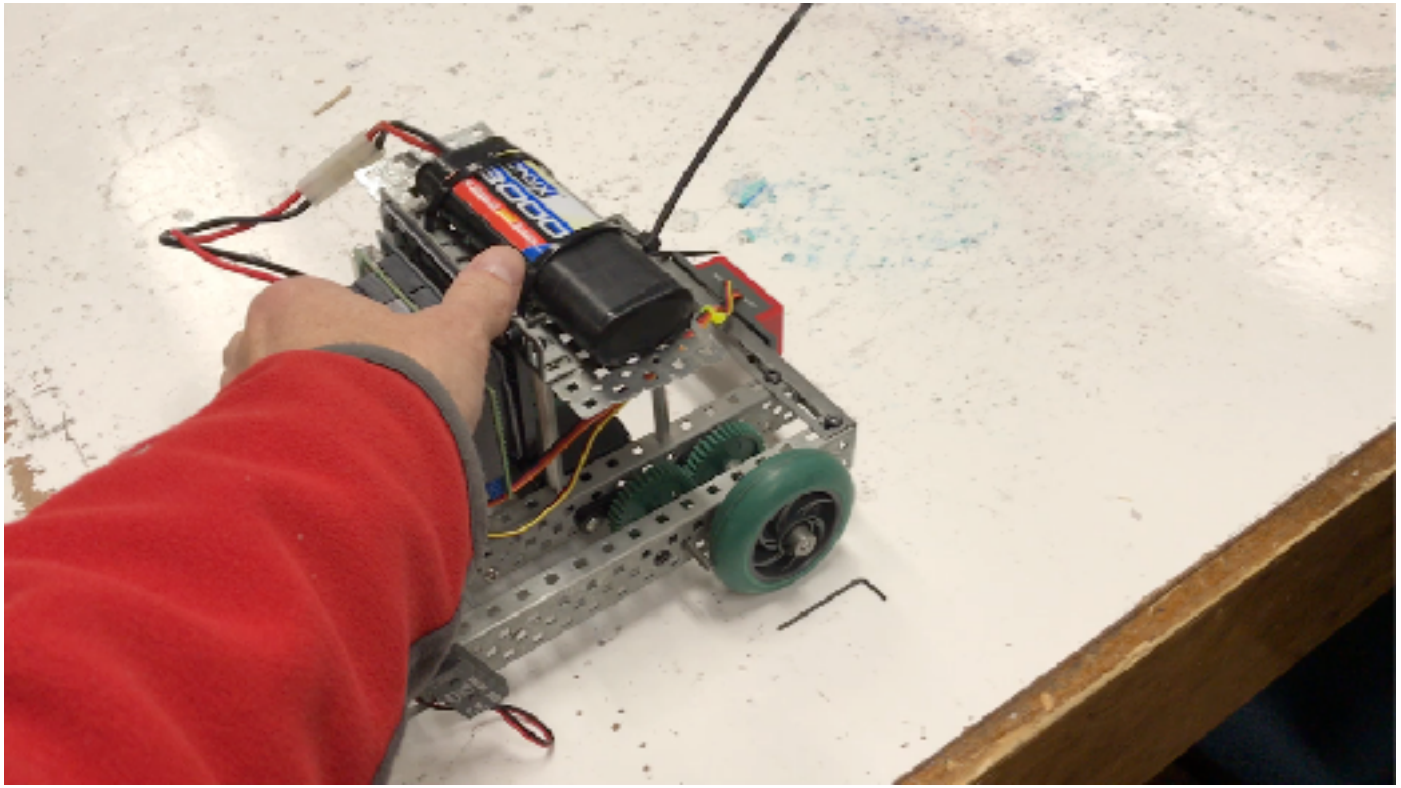
Function	What it does

forward(1, seconds, 100);	motor[rightMotor] = 100;
	motor[leftMotor] = 100;
	sleep(1000);

turnRight(500, milliseconds, 60);	motor[rightMotor] = 60;
	motor[leftMotor] = -60;
	sleep(500);

Your Second Challenge

Drive until 10cm from wall, then stop

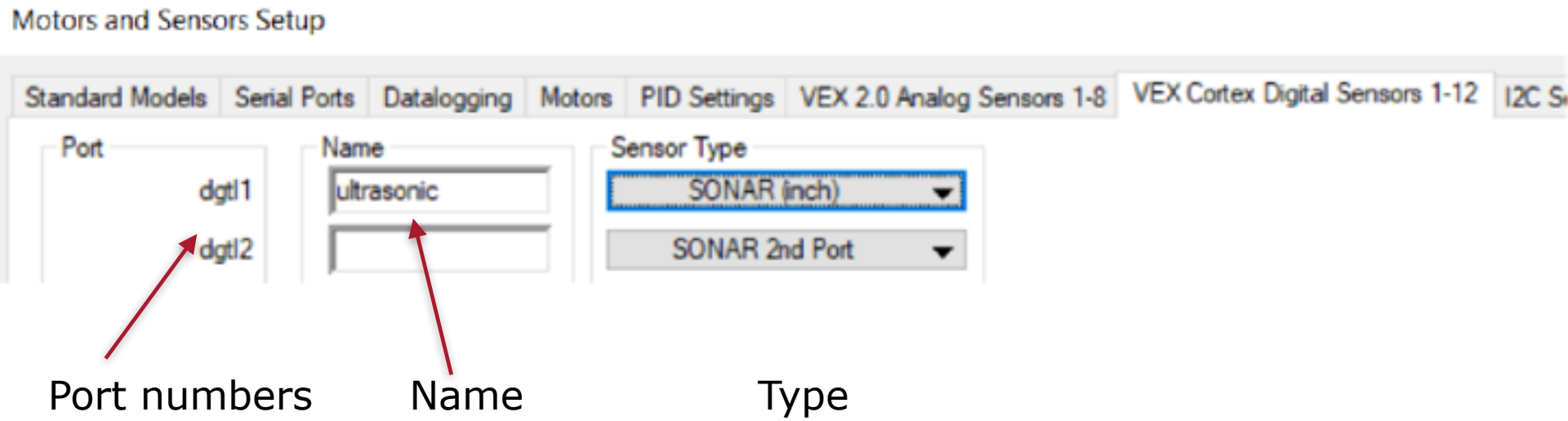


Use a Sensor

Sensors allow the robot to understand it's state and the world around it

Ultrasonic rangefinder gives the distance to an object in centimeters

Edit setup to include rangefinder



Using a Rangefinder

- Start with the task main() as usual

```
task main()
{
    while (true) {
        float distance = SensorValue[ultrasonic];
        if (distance > 11) {
            forward(10, milliseconds, 60);
        } else {
            stopAllMotors();
        }
    }
}
```

Using a Rangefinder

- Do the driving and range finding forever

```
task main()
{
    while (true) {
        float distance = SensorValue[ultrasonic];
        if (distance > 11) {
            forward(10, milliseconds, 60);
        } else {
            stopAllMotors();
        }
    }
}
```

Using a Rangefinder

- Get the distance to the object in front of the robot

```
task main()
{
    while (true) {
        float distance = SensorValue[ultrasonic];
        if (distance > 11) {
            forward(10, milliseconds, 60);
        } else {
            stopAllMotors();
        }
    }
}
```

Using a Rangefinder

- If the robot is greater than 11 inches from the object, then drive forward otherwise stop the motors

```
task main()
{
    while (true) {
        float distance = SensorValue[ultrasonic];
        if (distance > 11) {
            forward(10, milliseconds, 60);
        } else {
            stopAllMotors();
        }
    }
}
```

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…

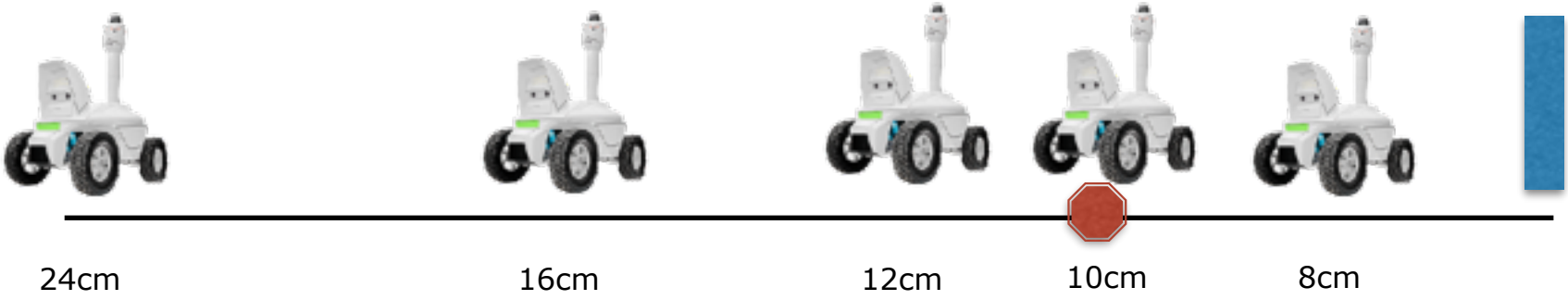
Use proportional control!

Proportional Control

MS4SSA
Math and Science for Sub-Saharan Africa

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**

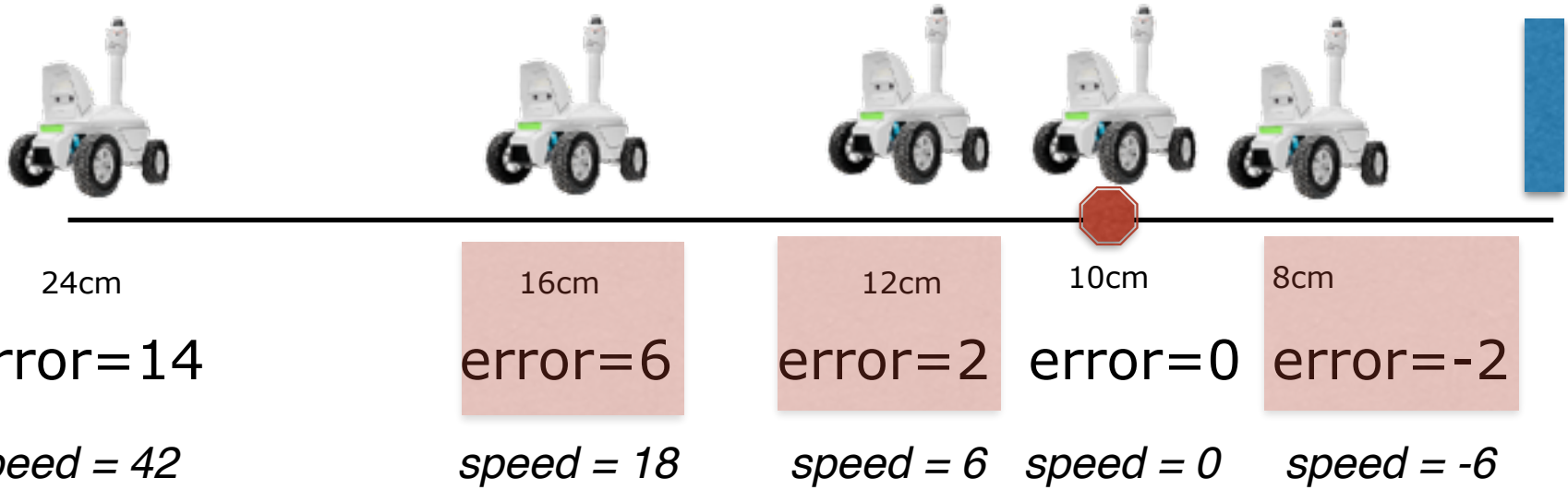


error=14 error=6 error=2 error=0 error=-2

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.

Proportional Gain

MS4SSA
Math and Science for Sub-Saharan Africa



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 Kp is too small?

Kp = 1

error=16
speed=16

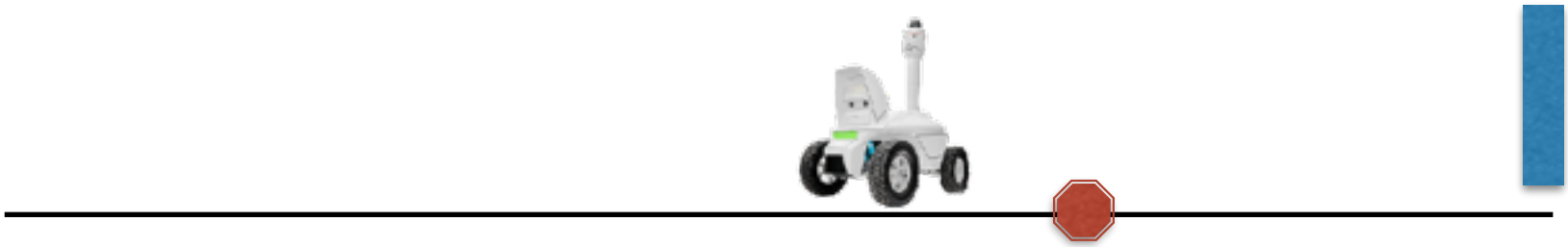
What if Kp is too small?

Kp = 1

error=3
speed=3

What if Kp is too small?

Kp = 1

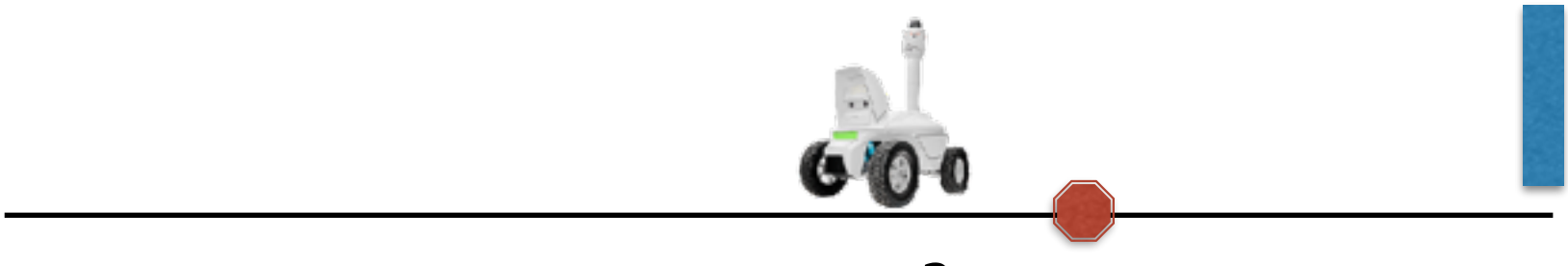


error=2
speed=2

A speed of 2 is too little
to make the motors turn

What if Kp is too small?

Kp = 1



error=2
speed=2

The robot never reaches the set point!

What if Kp is too large?

Kp = 10

error=24
speed=240

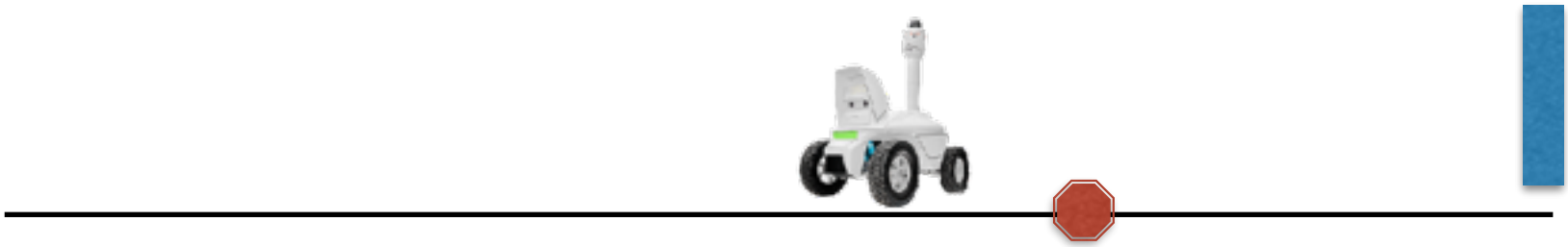
What if Kp is too large?

Kp = 10

error=5
speed=50

What if Kp is too large?

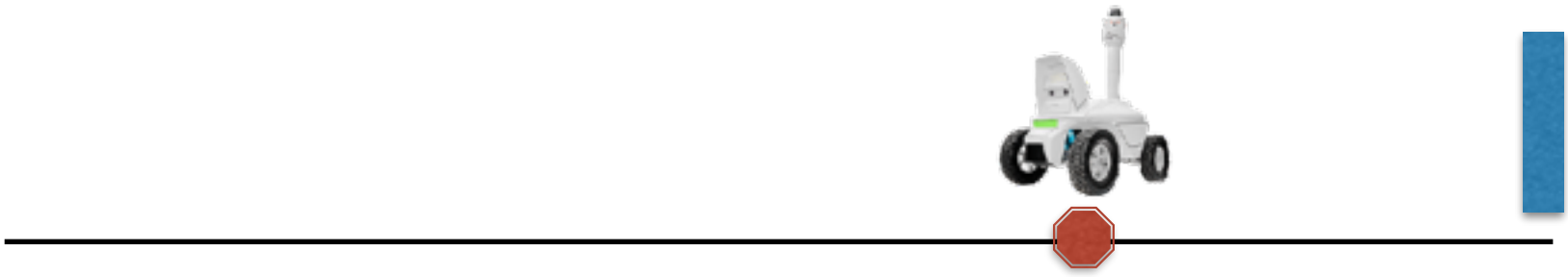
Kp = 10



error=2
speed=20

What if Kp is too large?

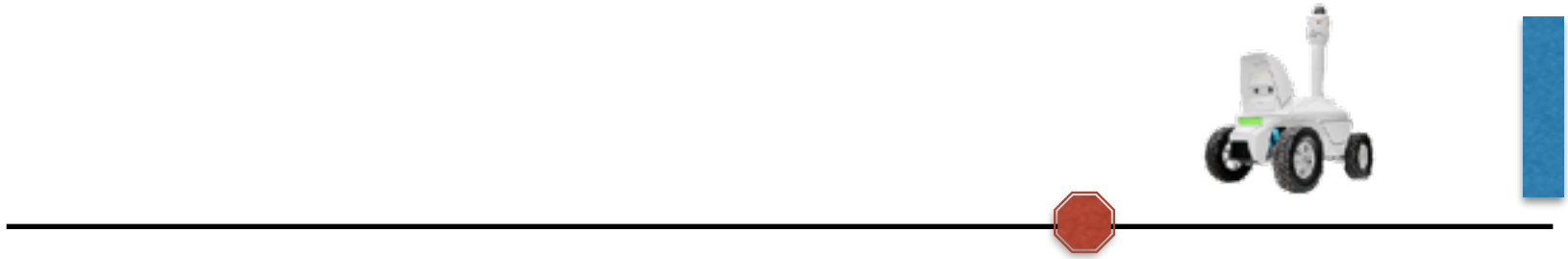
Kp = 10



But we're going so fast that the robot can't stop

What if Kp is too large?

Kp = 10

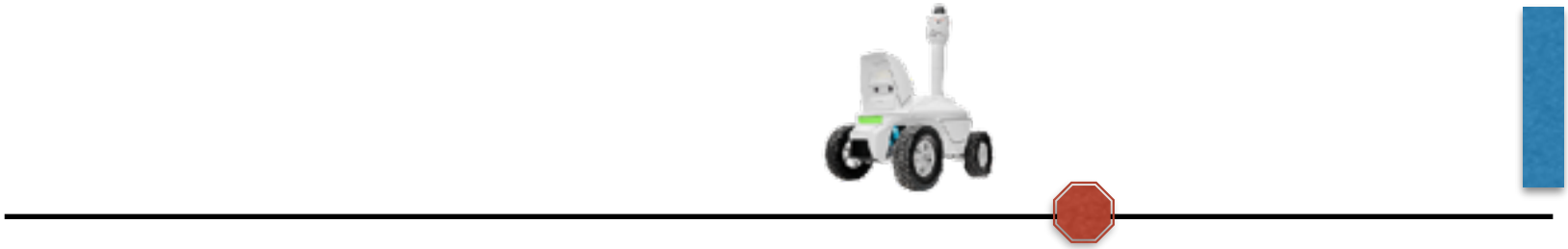


error=-2
speed=-20

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

```
const float Kp = 1.0;
const float setPoint = 11;

task main()
{
        while (true) {
                float error = SensorValue[ultrasonic] - setPoint;
                forward(10, milliseconds, error * Kp);
        }
}
```

Compute the error

Drive at a speed proportional to the error

Modify Program 2

- Edit the program that is provided to that it stops on the line.
  - Modify the Kp value to find a value that doesn't oscillate or stop short of the line
  - Try smaller and larger values of Kp and observe what happens if it's too big or too small

\documents\MS4SSARwanda\ultrasonicNoProportional

What we accomplished

- Talked about why robot programming
- You wrote programs to drive the robot and use sensors for understanding the world around the robot