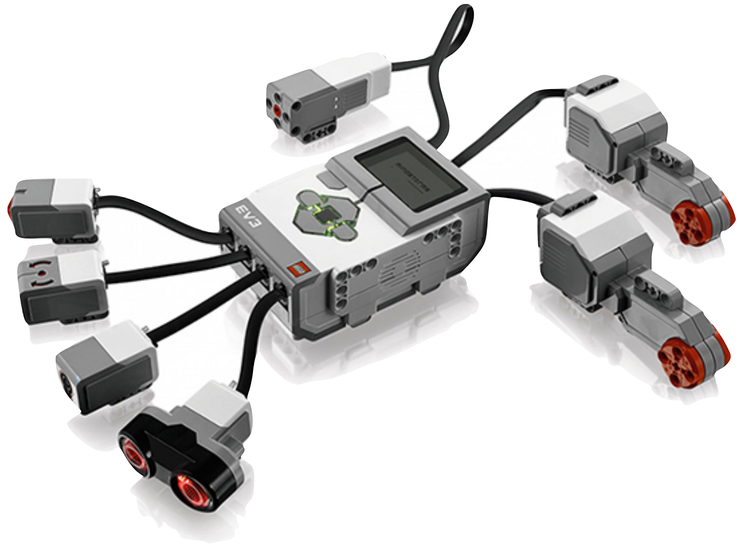


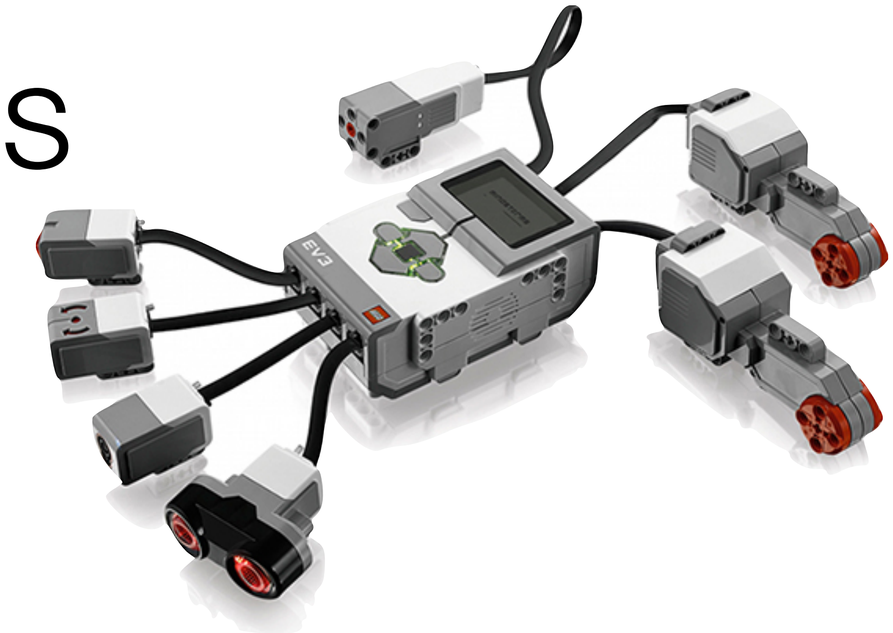
COMP329 Robotics and Autonomous Systems

Lecture 4: LeJOS and EV3

Dr Terry R. Payne
Department of Computer Science

UNIVERSITY OF LIVERPOOL

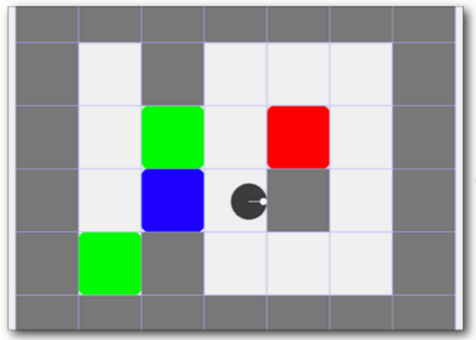
Today's Aims



- Before the labs start, we will take a look at the basics of programming our EV3 robots.
  - There will be three elements:
    - Programming robots in the abstract.
    - The EV3 Environment
    - The LeJOS API
- The EV3 is the robot we will use for the labs and the projects.
  - LeJOS is the language we will use to program the EV3.
- The COMP329 RoboSim environment
  - Provides prototyping environment when Robotics Lab is closed

COMP329: RoboSim

Robotics and Autonomous Systems



Resources

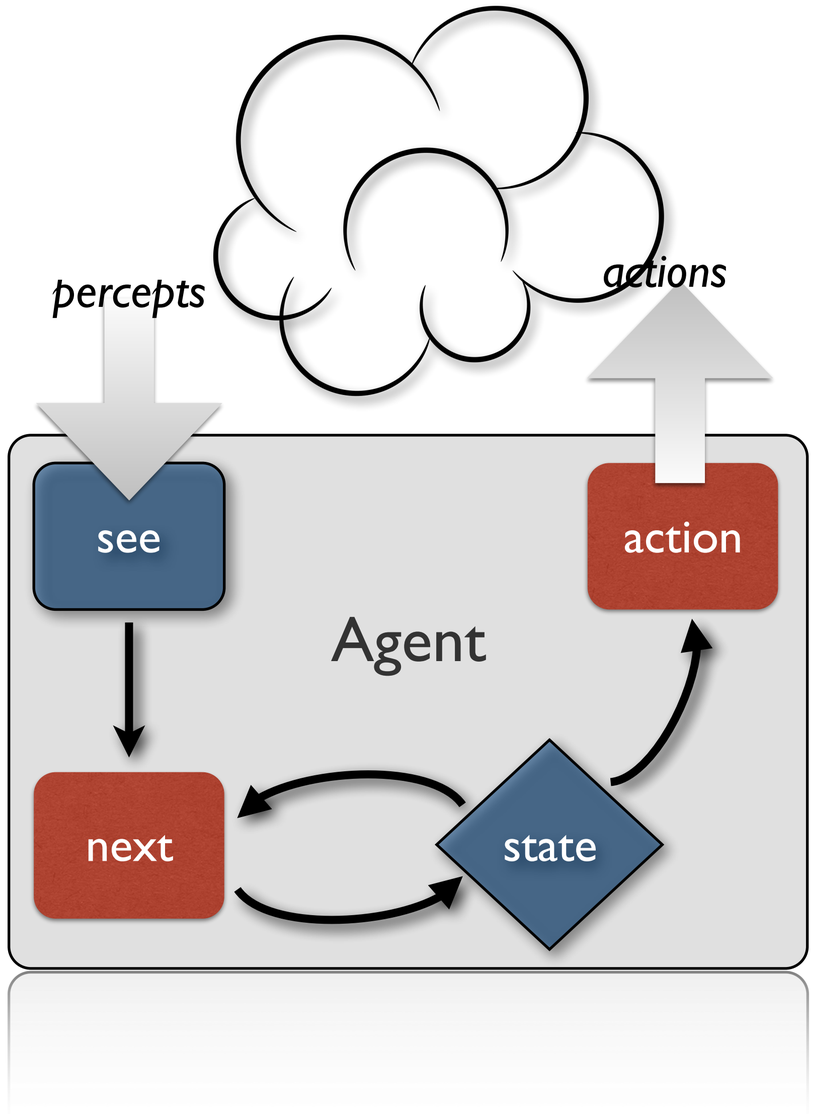
- **Jar:**
  - **comp329robosim.jar**
- **Getting Started:**
  - **RoboSym Guide.pdf**
- **JavaDocs:**
  - **comp329robosym java docs**
  - *Note that these documents could be improved. If there are any queries or inaccuracies, then please contact me*
- **Example config file:**
  - **robosimDefaultConfig.txt**
- **Jason Environment:**
  - Jason Github Site
  - ***Programming Multi-agent Systems in AgentSpeak Using Jason.*** Rafael H. Bordini, Jomi Fred Hubner and Michael Wooldridge (Wiley, 2007)
  - The AgentSpeak / Jason book (pdf)

Simple Robot Simulator for COMP329 Assignments

The **RoboSim Java package** is designed to simulate a simple LeJOS style Lego Robot environment that can be used to test out simple programs, and to simulate robot control within the AgentSpeak / Jason environment. It has been modeled around the APIs used for the Lego Robot, which were taught and used for the first Assignment within COMP329 Robots and Autonomous Systems. Thus, they provide complementary functionality including:

- **Bump Sensor** (in the direction of travel) - `isBumperPressed()`
- **Directional UltraSound Proximity Sensor** - `getDirection()`, `setDirection(int degrees)`, `getUSenseRange()`
- **Colour Sensor** - `getCSenseColor()`
- **Pose Query** methods - `getHeading()`, `getX()`, `getY()`
- **Speed / Rotation / Distance** methods - `setTravelSpeed(int travelSpeed)`, `rotate(int degrees)`, `travel(int distance)`
- **Real Time Robot Diagnostic** Information - `monitorRobotStatus(boolean verbose)`

Basic Control Loop




```
while(true){
  read sensors
  update internal datastructures
  make decisions
  set power on motors
}
```

- Even robots with arms and legs move them by turning motors on and off.

Basic Control Loop

- We'll get into more detail about how exactly ***sensors*** work later in the course.
  - For now, just think of them as generating a value.

```
while(true){
  read sensors
  update internal datastructures
  make decisions
  set power on motors
}
```

- We won't get into more detail about how ***motors*** work.
  - Just think of a motor command as setting the power on the motors.
  - Setting the voltage on a particular port.

Remember Timescales

- Even slow processors work faster than the robot.
  - The code opposite does not allow the robot time to move
- You need to use sensors to determine when to activate/deactivate the motors
  - Assuming ***this*** and ***that*** are suitable values

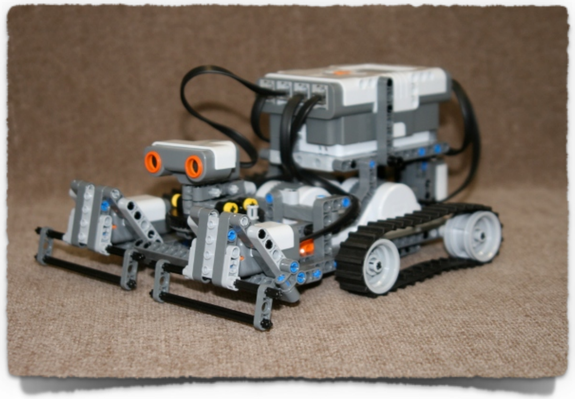
```
while(true){
  switch motors on
  :
  do stuff
  :
  switch motors off
}
```

```
while(true){
  read sensors
  :
  do stuff
  :
  if sensors say "this"{
    switch motors on
  }
  if sensors say "that"{
    turn motors off
  }
}
```

Our Robot

- The ***Lego EV3 brick*** is a small computer
  - powered by a Li-Ion rechargeable battery
    - intended for robot control.
  - 32-bit ARM9 CPU running Linux, running at 300 MHz
  - 16 Mbytes non-volatile flash storage
    - for storing programs
  - 64 Mbytes RAM
    - for runtime memory

- Speaker & Simple Display
  - 178×128 pixel monochrome LCD



The Brick Schematics

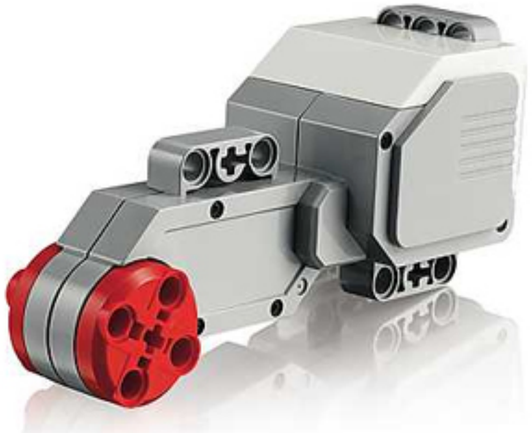
- Input
  - 4 ports (1 to 4) for sensors.
  - 6 input buttons (left, right, up, down, centre, escape)

- Output
  - 4 ports (A, B, C and D) for actuators/motors.
  - 178×128 pixel monochrome LCD

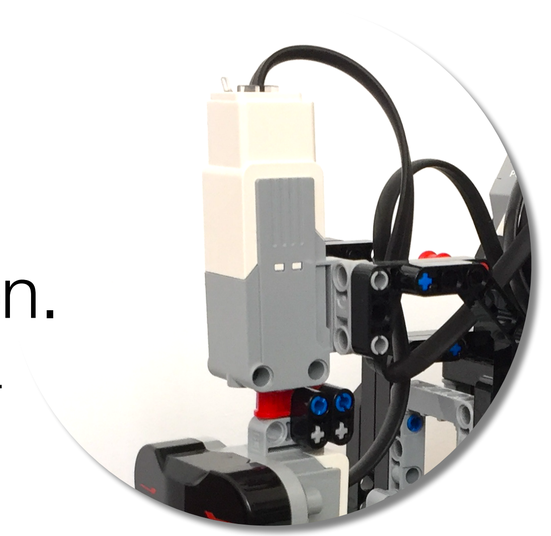
- Connectivity
  - 1 USB port (for downloading programs on the brick)
  - Bluetooth connection (for communication with computer)
  - Micro SD card (storing programs and booting OS)

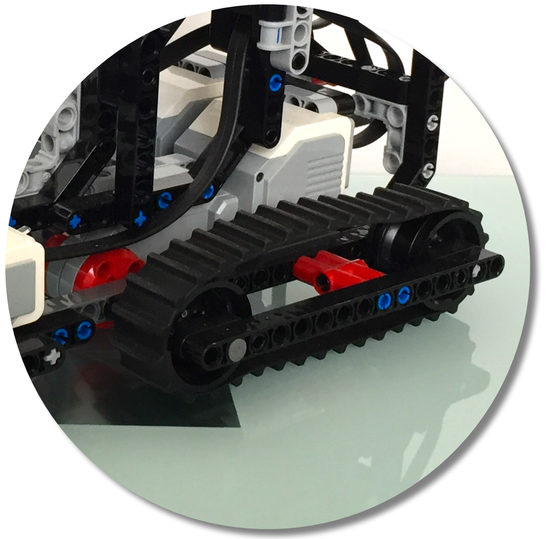
Motors and Locomotion

- These are the basic actuators of the EV3 kit.



- These are servomotors
  - Motor plus feedback
  - A sensor measures information about rotation and supplies it back to the brick.
  - The motor can be controlled very precisely.



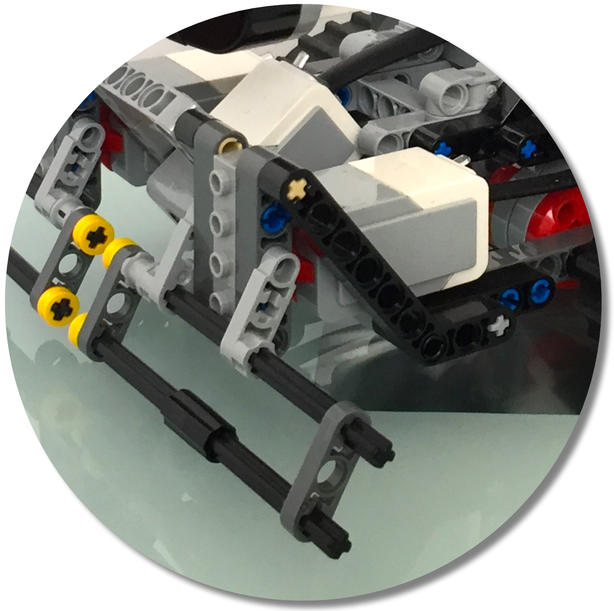


- We will use these to provide locomotion.
  - Other uses involve robotic arms, or active sensor support

Sensors - Touch

- Detects when the orange button is pressed.
  - Returns a Boolean
    - TRUE = sensor is pressed
    - FALSE = sensor is not pressed

- Mechanically the sensors work better if there is a bumper that presses the sensor.



- Our robot has two bumpers:
  - left and right.

Sensors - Infrared

- Standard range sensor.
  - We will talk more about how sonar work in a later lecture.
  - Reports distance to object.

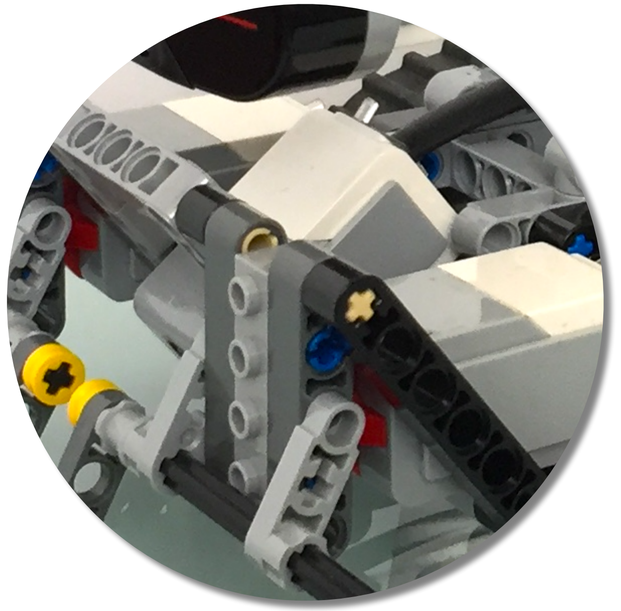
- Works effectively for objects in close proximity:
  - Effective range is 5 - 50cm
  - Precision of ±3cm

- Sensitive to the kind of objects it is detecting and subject to errors due to specular reflection.

- Our robot has one infrared sensor.

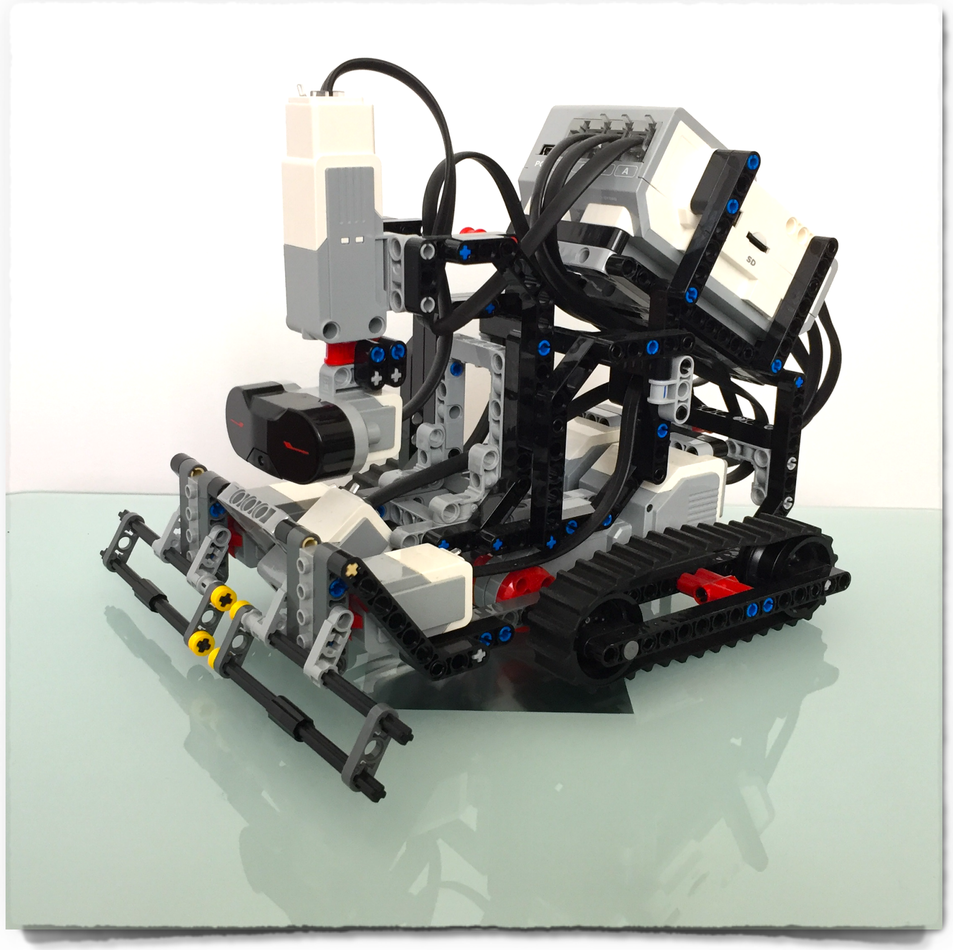
Sensors - Light/Colour

- Emits light and measures the reflection.
  - Analyses the colour of the reflection.
  - Can distinguish between a number of different colours.



Overall…

- Might not seem much, BUT:
  - Has pretty much the same capabilities as any mobile robot.
  - Can do all the mobile robot things I mentioned last time.
- Just as fun and frustrating.



LeJOS

- Lego Java Operating Systems
  - Spanish pronounciation:
    - “Lay-Hoss"
  - A small JVM is booted from the Micro SD card and allows Java programs to be executed.
  - Runs Java 1.7
- Some standard Java things are missing.
  - Original LeJOS VM was called “Tiny VM”
- However LeJOS has lots of useful robot-specific stuff.
- Java-based, so OO.
  - The robot-specific things are implemented as classes + objects.
- Classes to represent, for example:
  - Buttons
  - Motors
  - Sensors
- Access to these devices is then through method calls.

LeJOS

- In many ways LeJOS is just like other robot programming languages.

- Provides function calls that interface with with the robot hardware.
  - This means no robot programming language can be completely general…
  - …aspects of the language are specific to the hardware.

Hello World

- The completely standard program runs:

```
import lejos.hardware.Button;

public class HelloWorld {
   public static void main (String[] args) {
      System.out.println("Hello World");
      Button.waitForAnyPress();
   }
}
```

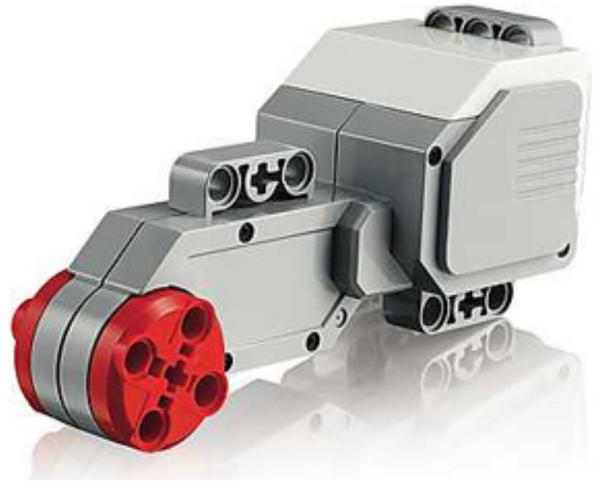
- However, without waiting, the message will flick by too fast to see.
  - The method `Button.waitForAnyPress()` is needed to stop the program from ending and clearing the display.

Using Motors

```
import lejos.hardware.Button;
import lejos.hardware.lcd.LCD;
import lejos.hardware.motor.Motor;

public class SimpleDriver {
   public static void main (String[] args) {
      System.out.println("Press any button to start");
      Button.waitForAnyPress();
      LCD.clear();
      Motor.B.forward();
      Motor.C.forward();
      System.out.println("Press any button to stop");
      Button.waitForAnyPress();
      Motor.B.stop();
      Motor.C.stop();
   }
}
```

This is a **simple** version!
Better to use the regulated class (used in the lab example)

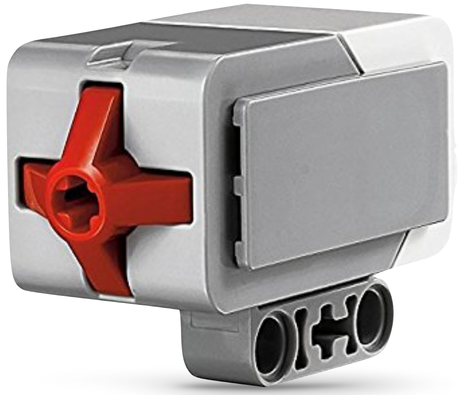
- Note that motors have to be connected to ports B and C for this to have any effect.
  - This is the case for our EV3 robot
- Robot control programs are very sensitive not only to the robot chassis, but also to the way the robot is wired.



```
import lejos.hardware.Button;
import lejos.hardware.lcd.Font;
import lejos.hardware.ev3.LocalEV3;
import lejos.hardware.sensor.EV3TouchSensor;
import lejos.robotics.SampleProvider;
import lejos.hardware.motor.Motor;

public class SimpleSensor {
	public static void main(String[] args) {
		EV3TouchSensor leftBump = new EV3TouchSensor(LocalEV3.get().getPort("S2"));
		EV3TouchSensor rightBump = new EV3TouchSensor(LocalEV3.get().getPort("S1"));
		SampleProvider leftTouch= leftBump.getMode("Touch");
		SampleProvider rightTouch= rightBump.getMode("Touch");

		float[] leftSample = new float[leftTouch.sampleSize()];
		float[] rightSample = new float[rightTouch.sampleSize()];

		while (!Button.ESCAPE.isDown()) {
			leftTouch.fetchSample(leftSample, 0);
			if (leftSample[0] == 1.0) {
				Motor.B.forward();
				Motor.C.forward();
			}
			rightTouch.fetchSample(rightSample, 0);
			if (rightSample[0] == 1.0) {
				Motor.B.stop();
				Motor.C.stop();
			}
		}
		leftBump.close();
		rightBump.close();
	}
}
```

Using Sensors

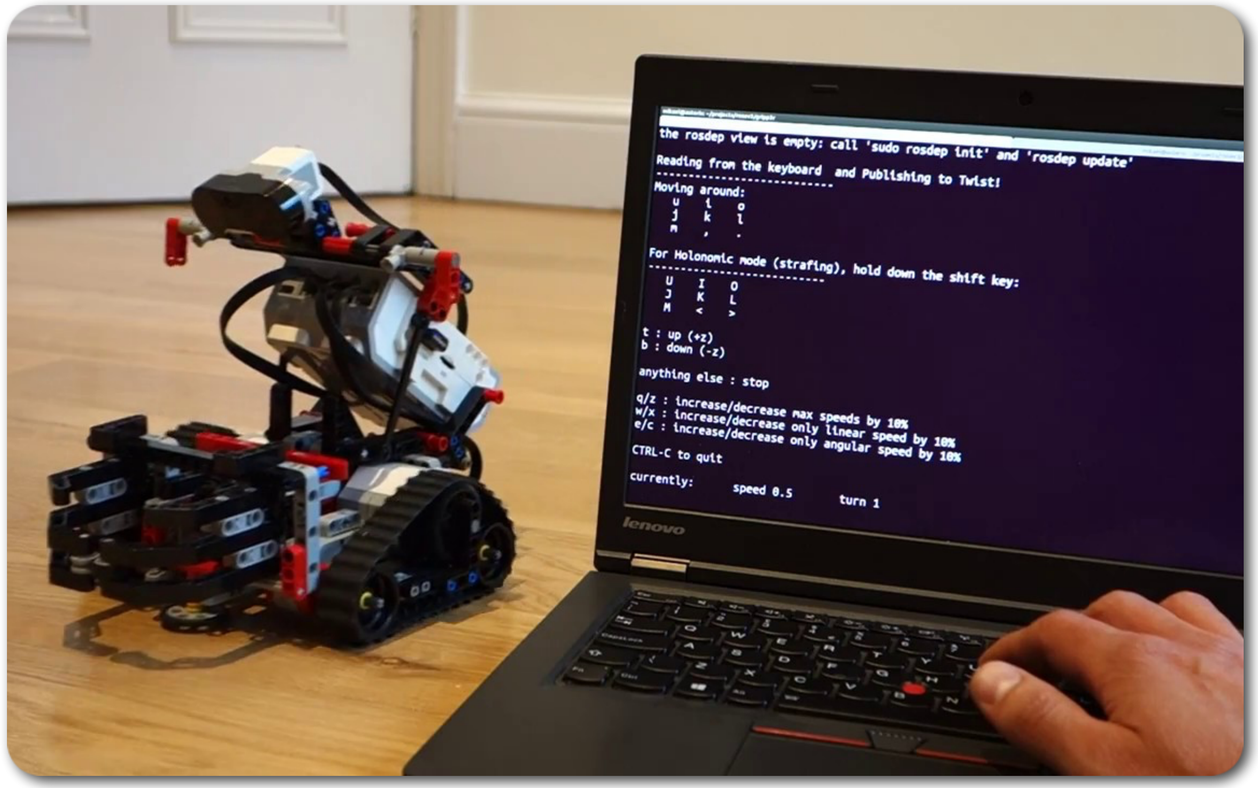
- To use a touch sensor we need to introduce a couple more objects.
  - The touch sensors themselves; and
  - The sensor port they are connected to.
- A basic loop is used in which sensors are read and a decision made based on the sensor state.



Development Cycle

- Another difference between the programs you have written before and robot programs.
  - You do the usual:
    - Write
    - Compile + Link
    - Debug
  - as before.

- However, there is one more step…
  - …before running the program you have to download it onto the robot.
    - Through USB or Bluetooth.



Development Cycle

- If you use Eclipse, the download step is mainly hidden.
  - ***But...***
  - The robot must be turned on.
  - And the USB cable must be plugged in if you are using it.
  - You may also notice that compilation produces a `.nxj` file rather than a `.class` file.
- If you use don't use Eclipse, you have to run commands to:
  - Compile
  - Link
  - Download
- from the command line.

All this will be explained in the Practical Sessions

Summary

- This lecture took a look at some of the issues in programming robots.
  - It presented a mixture of high-level concepts and practical advice.
  - The lecture also presented some information on LeJOS, which you will use to program the EV3 robots.

Available from Springer - check electronic availability through the library (Shibboleth)

https://link.springer.com/book/10.1007%2F978-1-4842-2005-4

- In the next lecture, we will look at agents
  - Including the agent control loop!