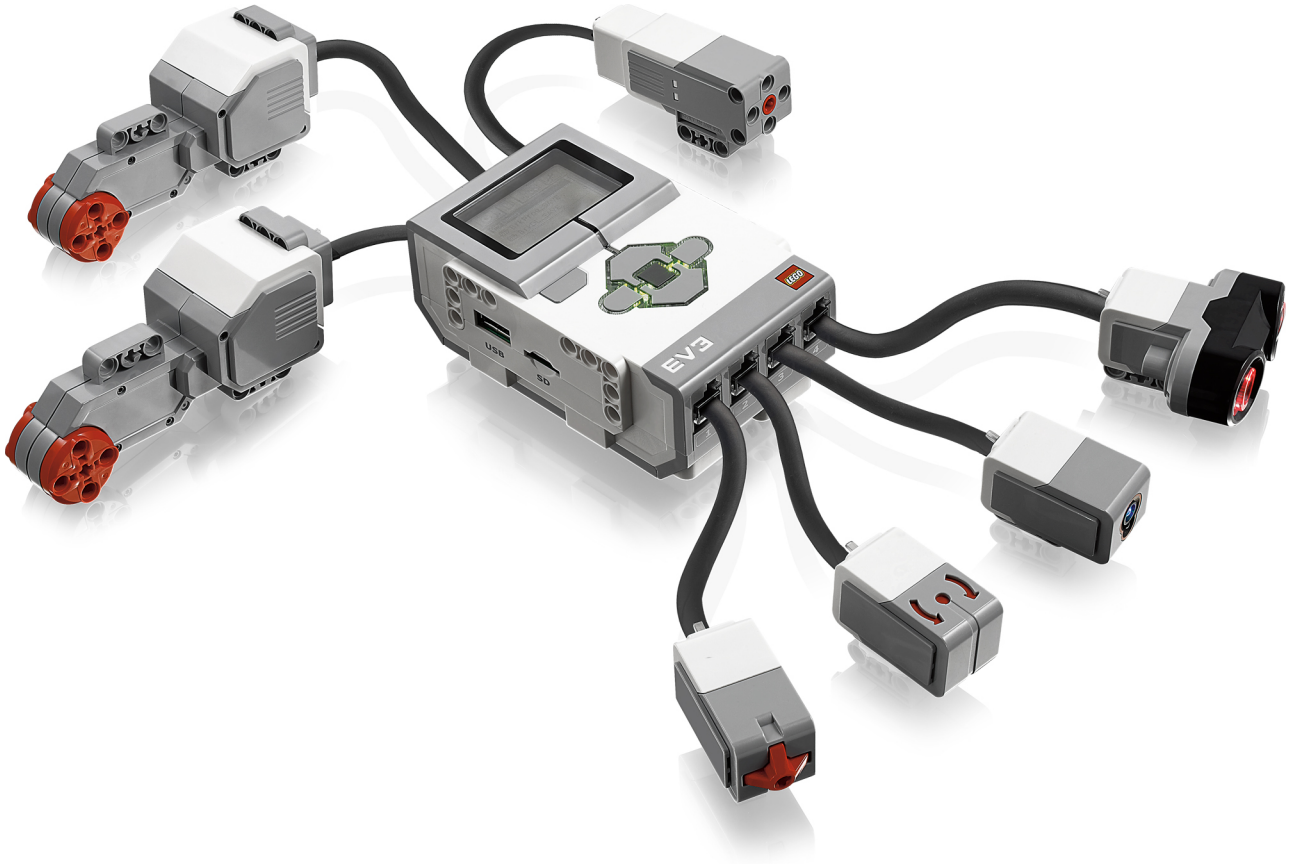
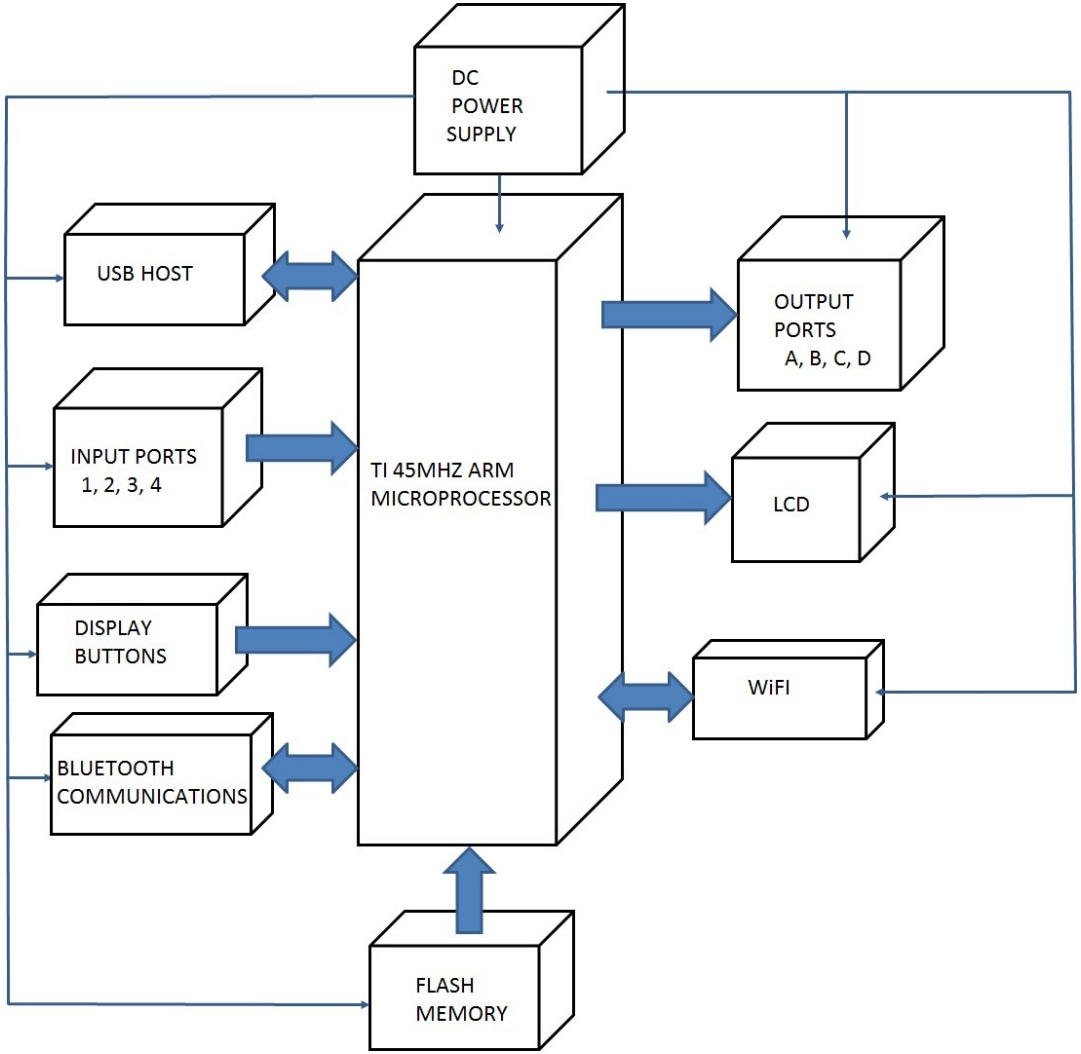
Introduction to leJOS and the EV3

Basic Components



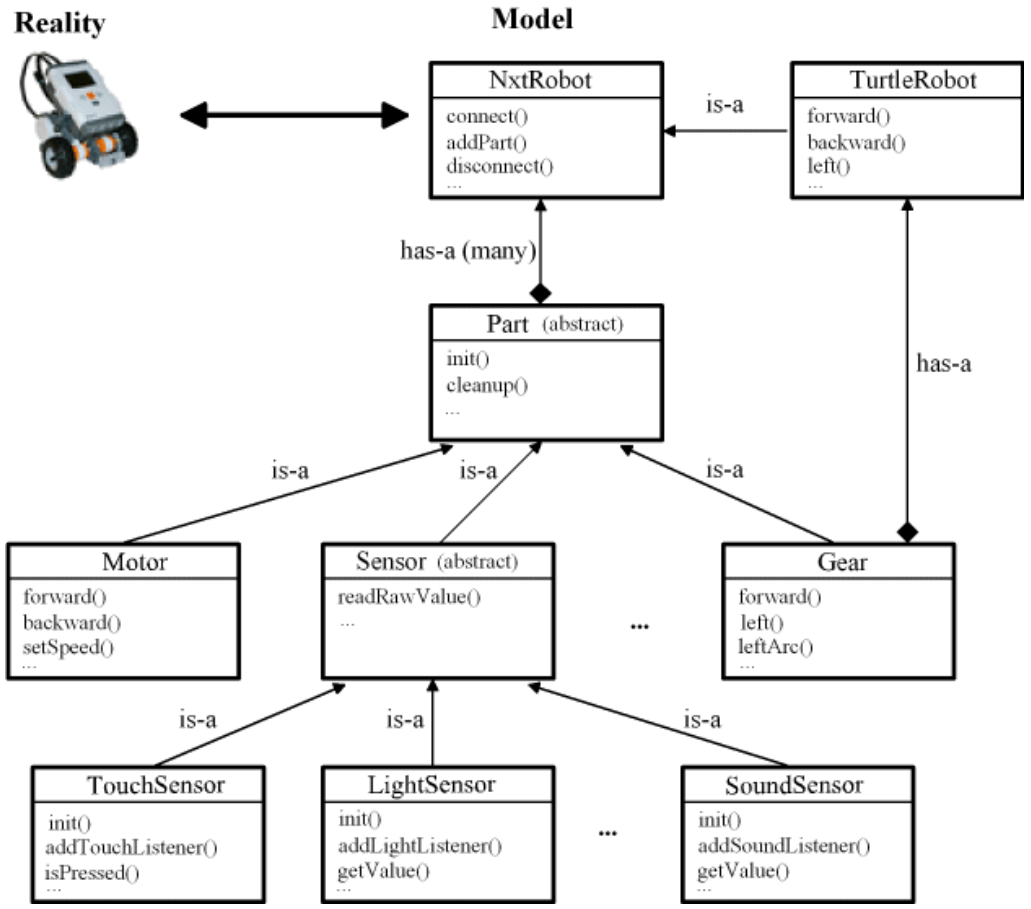
Lego Inc., Mindstorms

System Model



Don Wilcher, http://makezine.com/projects/hacking-the-lego-ev3-build-your-own-object-sensor-eyes/

Software Developer's Model



leJOS

- Object oriented language (Java)
- Preemptive threads (tasks)
- Arrays, including multi-dimensional
- Recursion
- Synchronization
- Exceptions
- Java types including float, long, and String
- Most of the java.lang, java.util and java.io classes
- A Well-documented Robotics API

leJOS cont.

- leJOS classes available for entities in the Mindstorms environment.
- An autonomous system (e.g. robot) is defined using instances of the leJOS classes.
- Each class contains methods for determining the configuration and operating parameters of each component.
- leJOS API can be found at http://www.lejos.org/ev3/docs/

leJOS cont.

leJOS runs on top of a TI AM1808 (ARM926EJ-S core) @ 300 Mhz:

- Avoid undue complexity.
- Keep class hierarchies tight.
- Pay attention to how many threads you generate.

leJOS cont.

- Relative to a 2.8 Ghz Core i7 (middle of the road), the EV3 is 40 times slower on integer calculations and 120 times slower doing floating point calculations!
- Benchmark your classes.
- The overall computational load > ∑ individial class loads due to overhead.
- Test, test, test…

Lejos Program Example

- Write a simple program to drive a two-wheeled robot forward until the touch sensor is activated.

```
package myPackage;

import lejos.hardware.ev3.LocalEV3;                                    // Inserted automatically by Eclipse
import lejos.hardware.motor.Motor;
import lejos.hardware.port.Port;
import lejos.hardware.sensor.EV3TouchSensor;
import lejos.hardware.sensor.SensorModes;
import lejos.robotics.RegulatedMotor;
import lejos.robotics.SampleProvider;

public class demo1 {

// Use the default constructor; allocate resources used by the program here.

// Motor classes are static; there is one for each port A-D.  It's useful to create alternate references
// as shown below for descriptive purposes.

        static RegulatedMotor leftMotor = Motor.A;
        static RegulatedMotor rightMotor = Motor.D;

// Create an instance of a touch sensor connected to Port 2
// This is somewhat complicated; treat as a pattern for now.

        static Port portTouch = LocalEV3.get().getPort("S2");                  // 1. Get port
        static SensorModes myTouch = new EV3TouchSensor(portTouch);            // 2. Get sensor instance
        static SampleProvider myTouchStatus = myTouch.getMode(0);              // 3. Get sample provider
        static float[] sampleTouch = new float[myTouchStatus.sampleSize()];    // 4. Create data buffer
```

```
// Class variables and constants

        public static final int FWDSPEED = 200;                 // Motors rotate at FWDSPD degrees/second

//  Program entry point

        public static void main(String args[]) {

                leftMotor.setSpeed(FWDSPEED);                   // Start moving forward
                rightMotor.setSpeed(FWDSPEED);
                leftMotor.forward();
                rightMotor.forward();

//  Move forward until switch closes

                while (true) {
                        myTouchStatus.fetchSample(sampleTouch, 0);      // Get switch state
                        if (sampleTouch[0] == 1) break;                 // Exit loop on contact
                }

                System.exit(0);                                 // Done!
        }
}
```

Some Important Classes

- Motor class

  is static and does not have to be instanced, e.g., you can simply call the methods associated with this class directly:

```
Motor.A.setSpeed(720); // 2 RPM
Motor.C.setSpeed(720);
Motor.A.forward();
Motor.C.forward();
```

  Note that A, B, D, and D refer to the motor ports so labeled in the EV3.

Some Important Classes

- Regulated Motor class

  provides an interface to the motor class that includes encoders (like the ones in your kits).

```
static RegulatedMotor leftMotor = Motor.A;
static RegulatedMotor rightMotor = Motor.D;
    leftMotor.setSpeed(FWDSPEED);
    rightMotor.setSpeed(FWDSPEED);
    leftMotor.forward();
    rightMotor.forward();
```

Some Important Classes

Sensors are a bit more involved compared to the motor class; there are 4 steps involved:

- Get an instance of a **port**.
- Get an instance of the **sensor** connected to this port.
- Get an instance of a **sample provider** for this sensor and set measurement modes.
- Allocate a memory **buffer** for the received data.

One this setup is performed, data can be read as shown in the following example.

Some Important Classes

Example: set up the ultrasonic sensor connected to Port 1 (done once):

Step 1: Create an instance that points to the sensor port used for the US.

```
static Port portUS = LocalEV3.get().getPort("S1");
```

Step 2: Create an instance of a US sensor (SensorModes is the interface corresponding to the US sensor).

```
static SensorModes myUS = new EV3UltrasonicSensor(portUS);
```

Step 3: Get an instance of a sample provider object in the measurement mode specified by the argument to the getMode method.

```
static SampleProvider myDistance = myUS.getMode("Distance");
```

Step 4: Create an array in which to receive the ultrasonic sensor data.

```
static float[] sampleUS = new float[myDistance.sampleSize()];
```

Some Important Classes

- To read from the sensor

```
myDistance.fetchSample(sampleUS,0);
```

- The SampleProvider class handles data from different sensors in a uniform way. Data are returned as arrays and units are standardized MKS.
- For the EV3UltraSonic sensor, data are in the range of 0-2.55 m. To convert to CGS, simply multiply by 100, e.g.,

```
static float distanceCGS = sampleUS[0]*100.0;
```

About EV3 Sensors

- Some sensors have a lot of options (and corresponding methods to exploit them).
- Unfortunately, a lot of this is poorly documented. Online search useful in finding out these details.
- Considerable variation in output from one sensor to another.
- Essential to characterize each sensor in your kit (again, testing is essential).

Other Topics (later)

- Timer-based sampling
- Importance and use of concurrency (threads)
- Code management (GIT is your friend)
- Testing (unit)
- Testing (system)
- And much more…