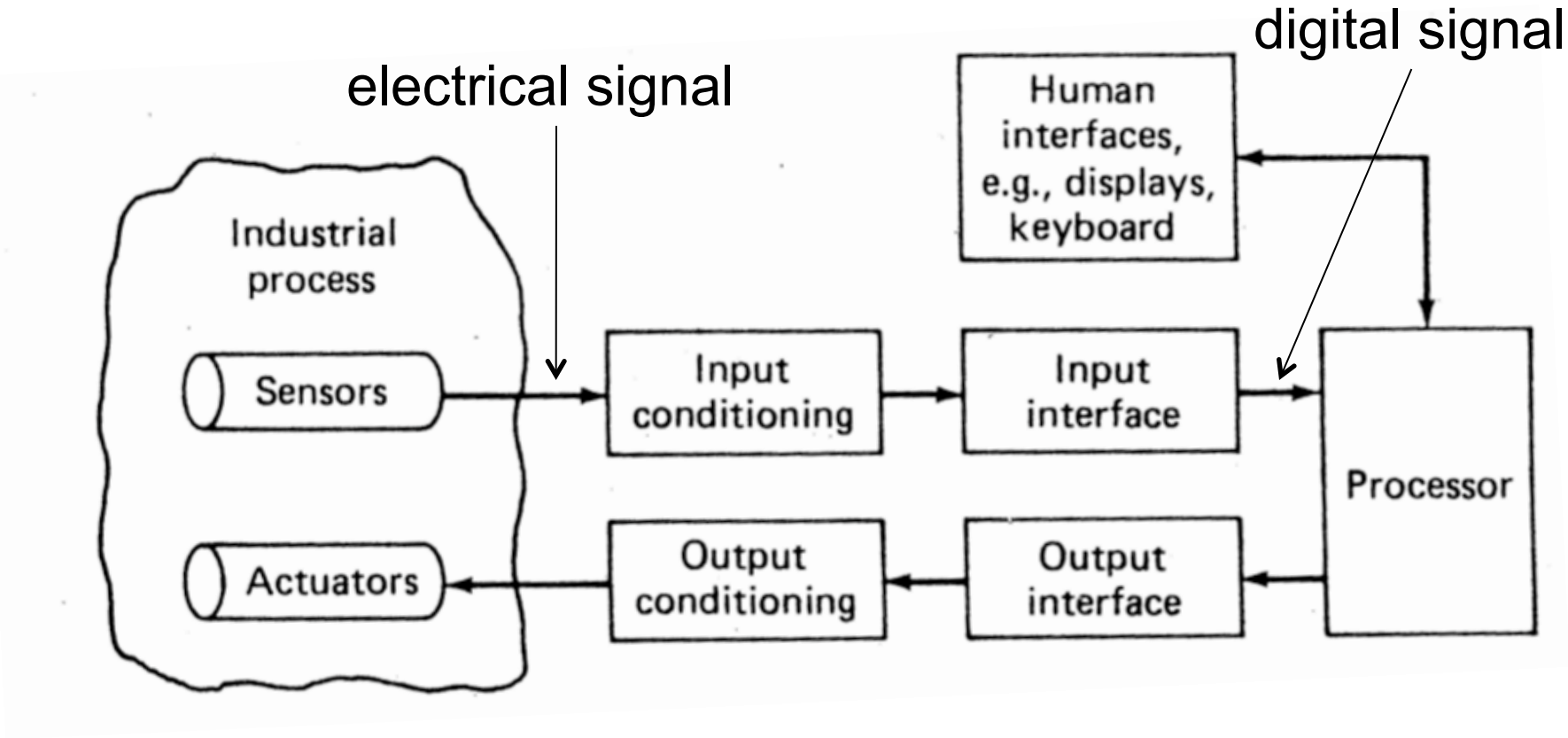
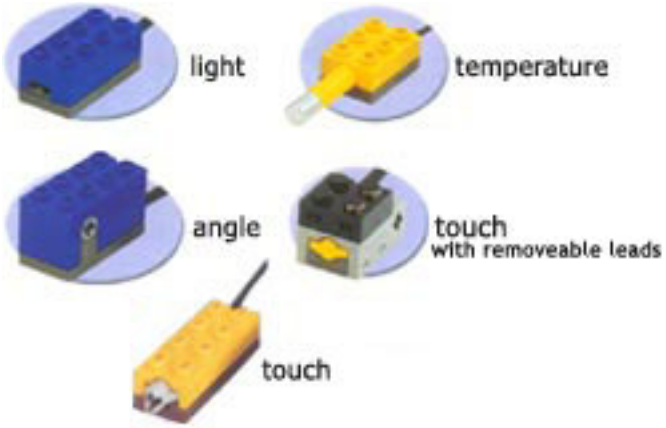
Sensors and sensing



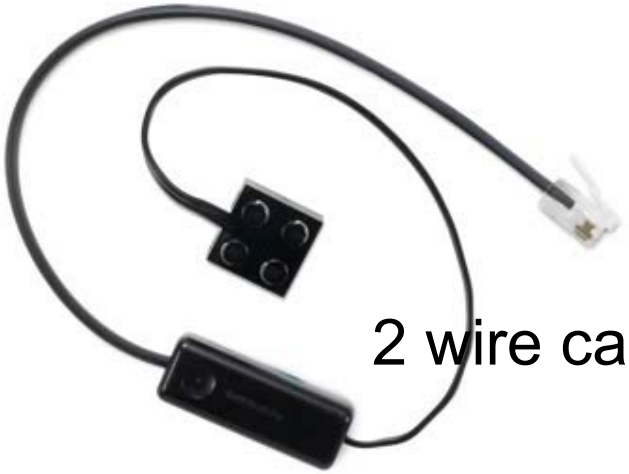
LEGO sensors

RCX sensors



NXT sensors

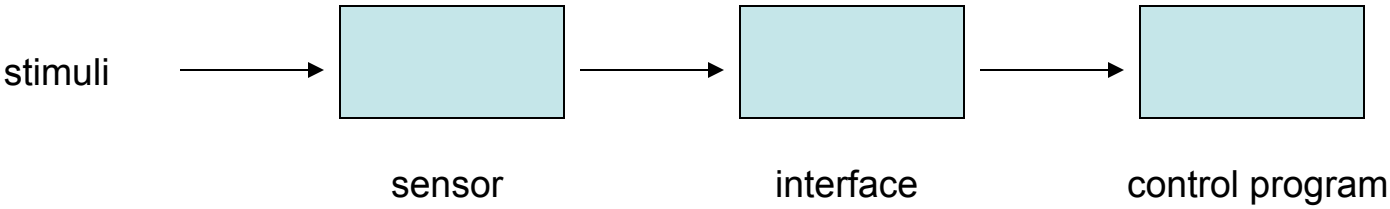
2x2 plate connector



2 wire cable

RJ12 connector

6 wire cable



analog quantity,
e.g. light

digital quantity,
e.g. light %

AD sensors

Temperature
Sound
Light

I2C sensors

Ultrasonic

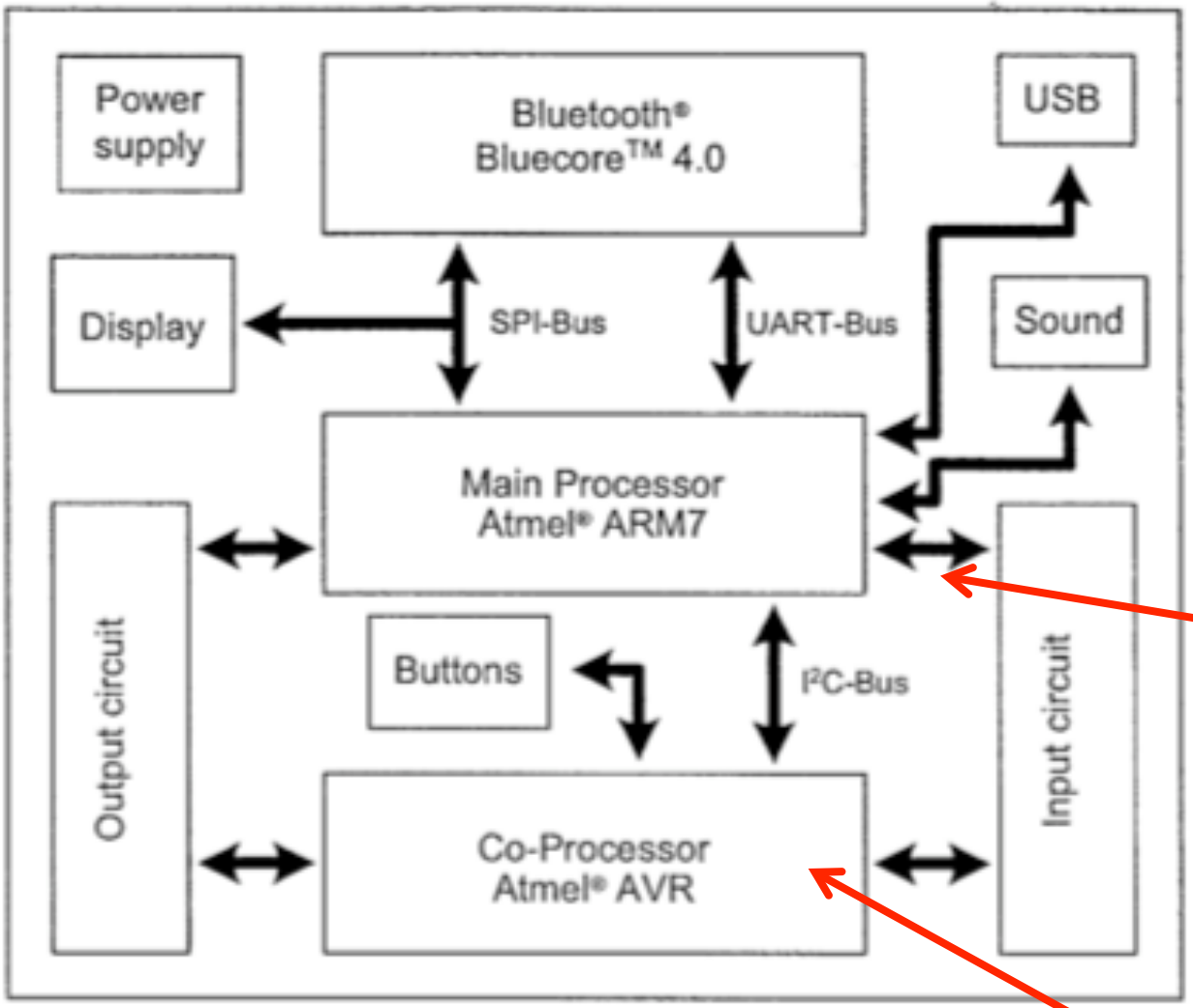
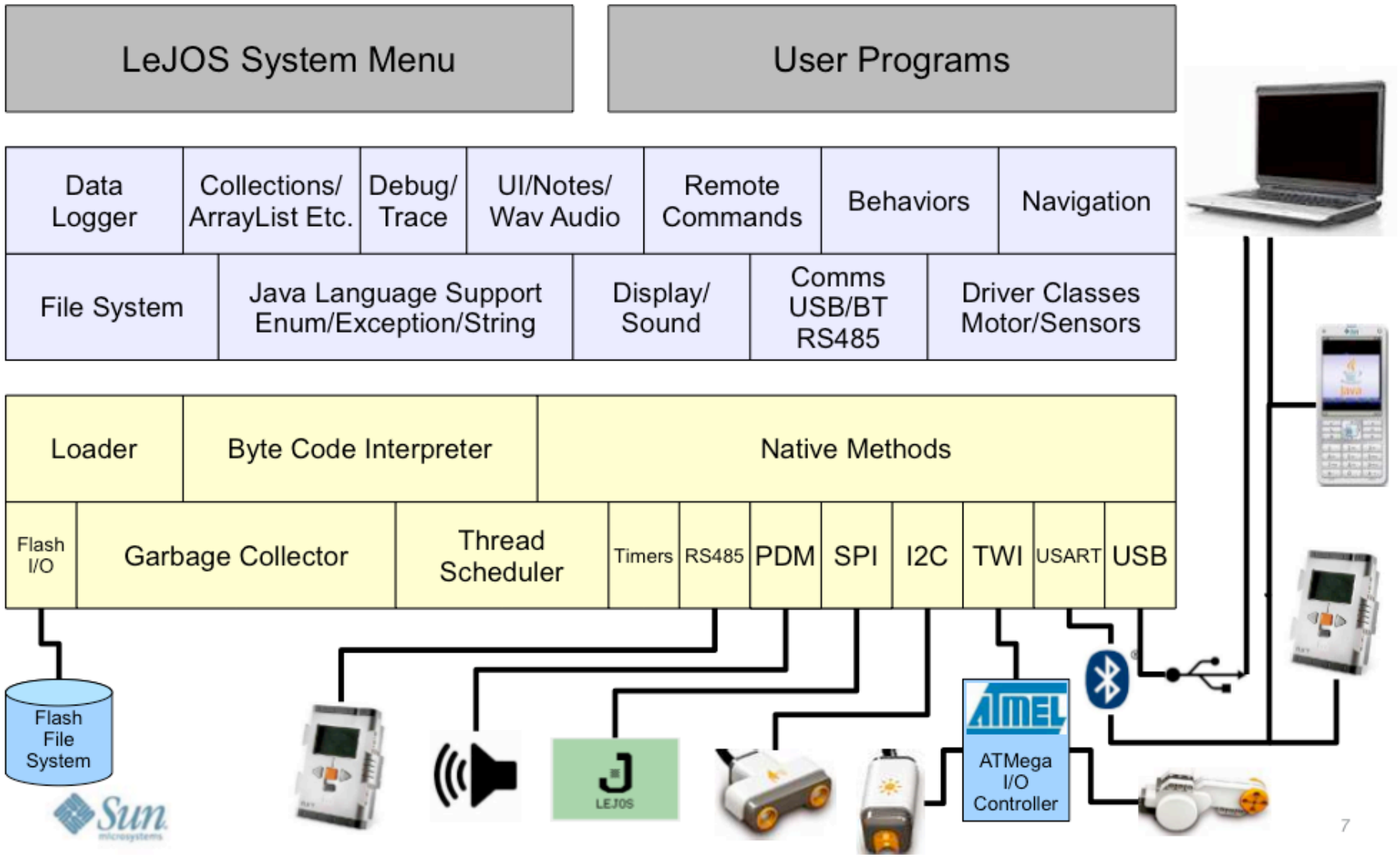


Figure 1: Hardware block diagram of the NXT brick

I2C

AD converter

LeJOS Architecture



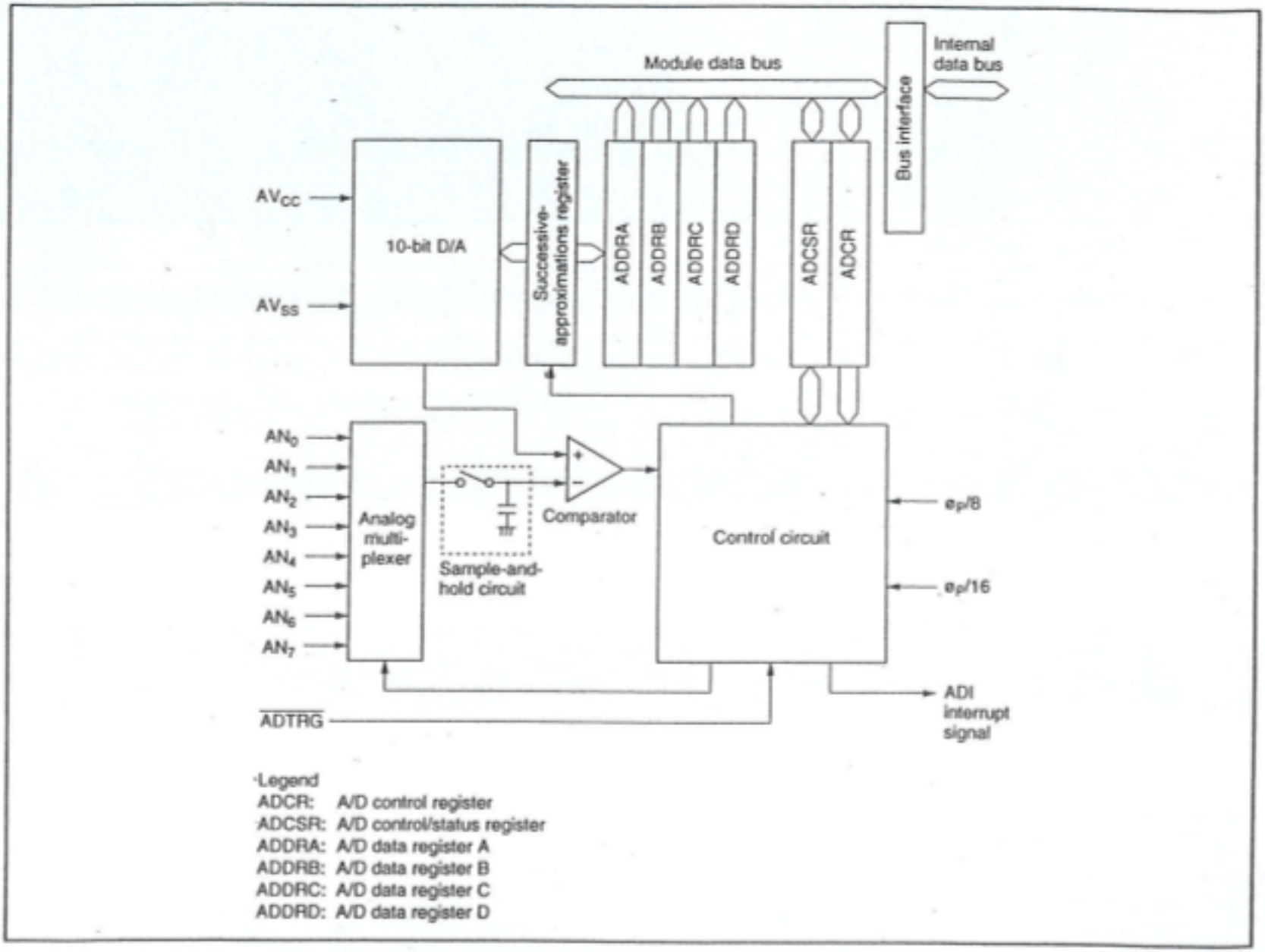
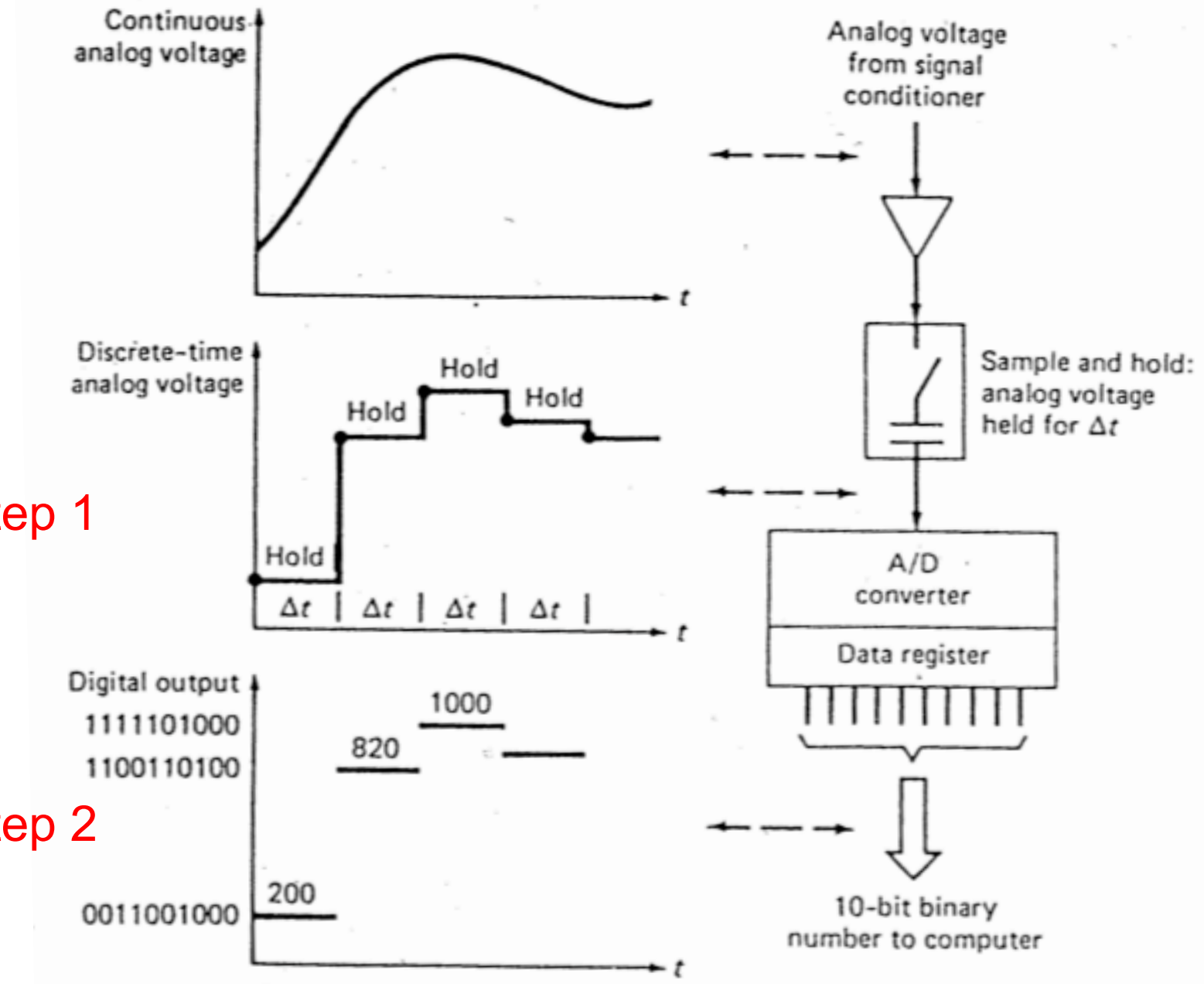
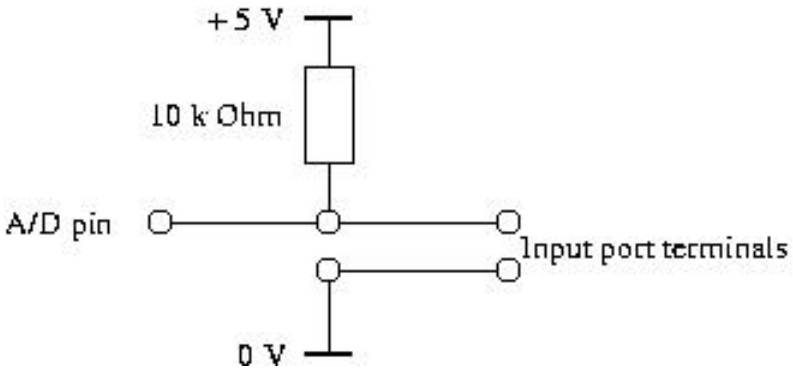


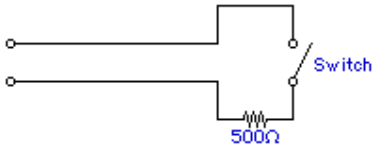
Figure 12-1 A/D Converter Block Diagram

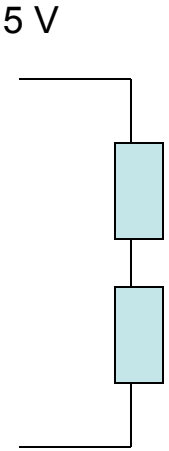

Continuous analog voltage
t
Analog voltage from signal conditioner
Discrete-time analog voltage
Hold
Hold
Hold
Hold
Δt
Δt
Δt
Δt
t
Sample and hold: analog voltage held for Δt
Step 1
A/D converter
Data register
Digital output
1111101000
1100110100
0011001000
1000
820
200
t
Step 2
10-bit binary number to computer

RCX input port



RCX touch sensor



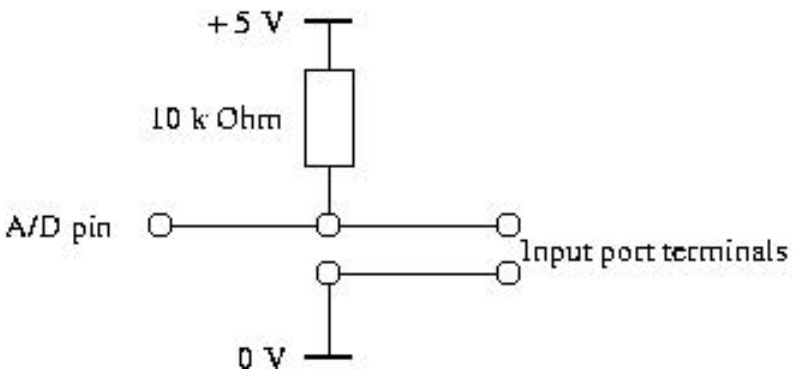


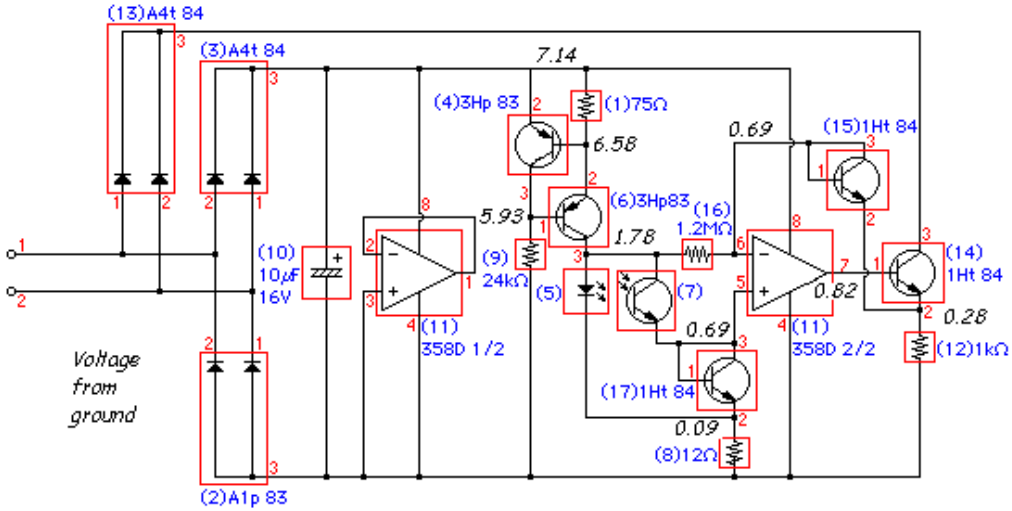
V(off) = 5 V
V(on) = 5 *500/(500+10000) V = 0.24 V

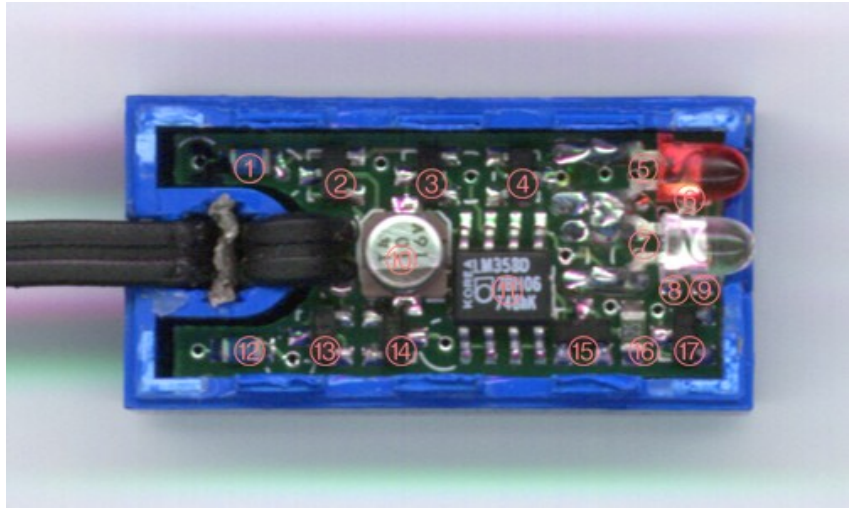
V(through R) = 5*R/(R+10000)
Raw Value(R) = 1023 * R/(R+10000)

Raw Value(500) = 1023 * 500/(500+10000) = 49
Raw Value(inf) = 1023 * inf/(inf+10000) = 1023

Voltage divider





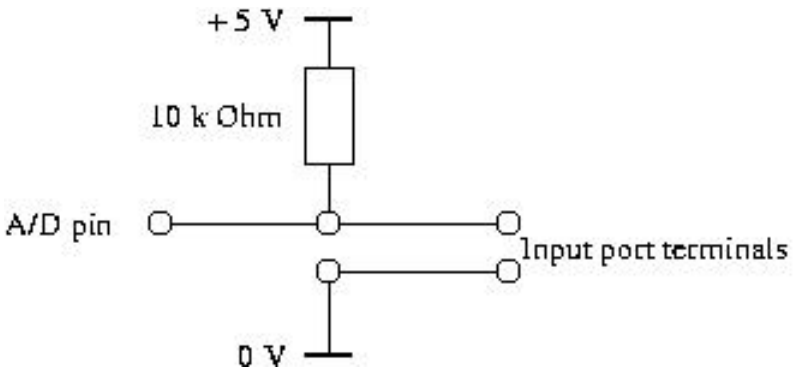


Passive sensors

touch, temperature

Active sensors

light, rotation

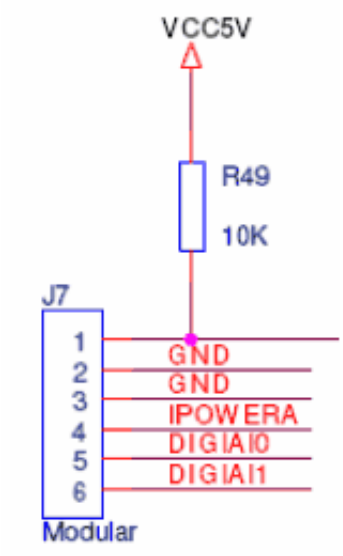


Passive sensors

touch, temperature

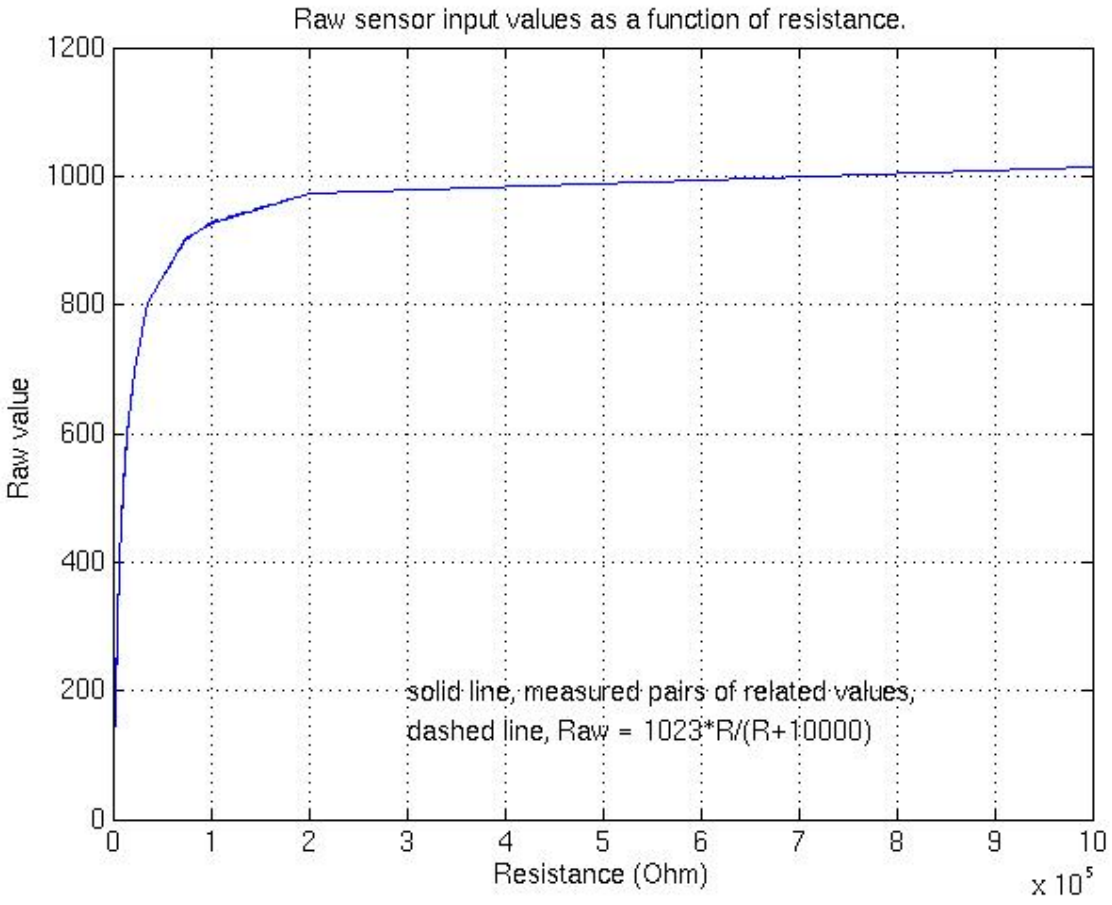
Active sensors

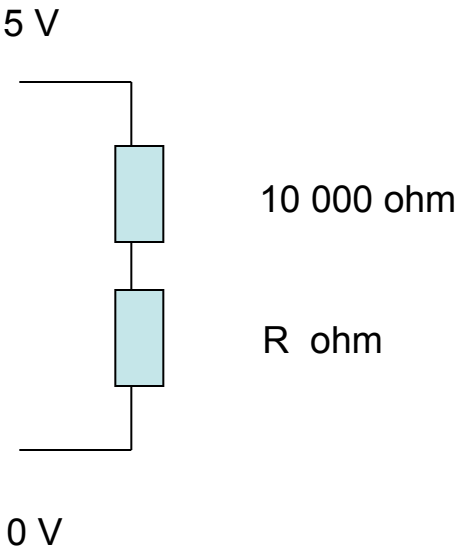
light, rotation



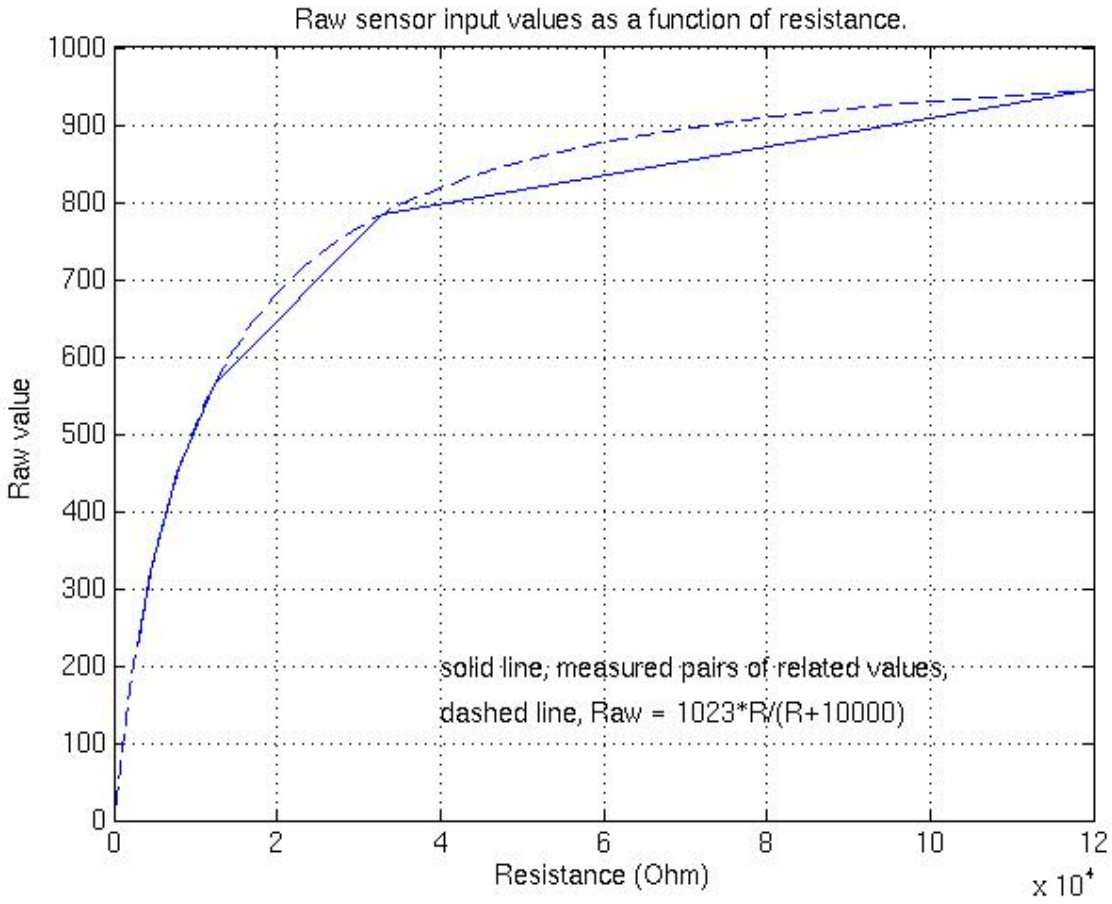
- Pin 1, ANA — Analog input and possible current output signal
- Pin 2, GND — Ground signal
- Pin 3, GND — Ground signal
- Pin 4, IPOWERA — 4.3 Volt output supply
- Pin 5, DIGIAI0 — Digital I/O pin connected to the ARM7 processor
- Pin 6, DIGIAI1 — Digital I/O pin connected to the ARM7 processor

Resistance (Ohm)	Raw value
1000	98
2250	202
4000	300
6400	402
10000	520
14000	600
21500	699
36300	802
74000	902
100000	927
200000	974
1000000	1014

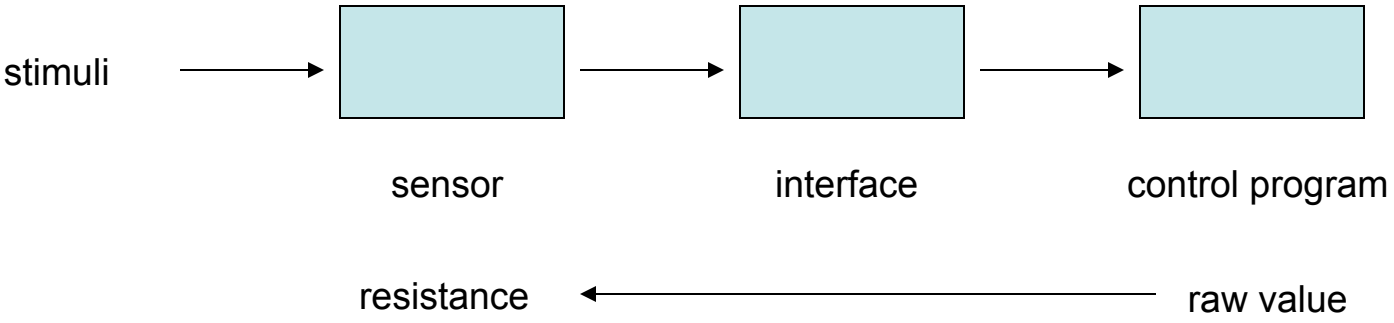


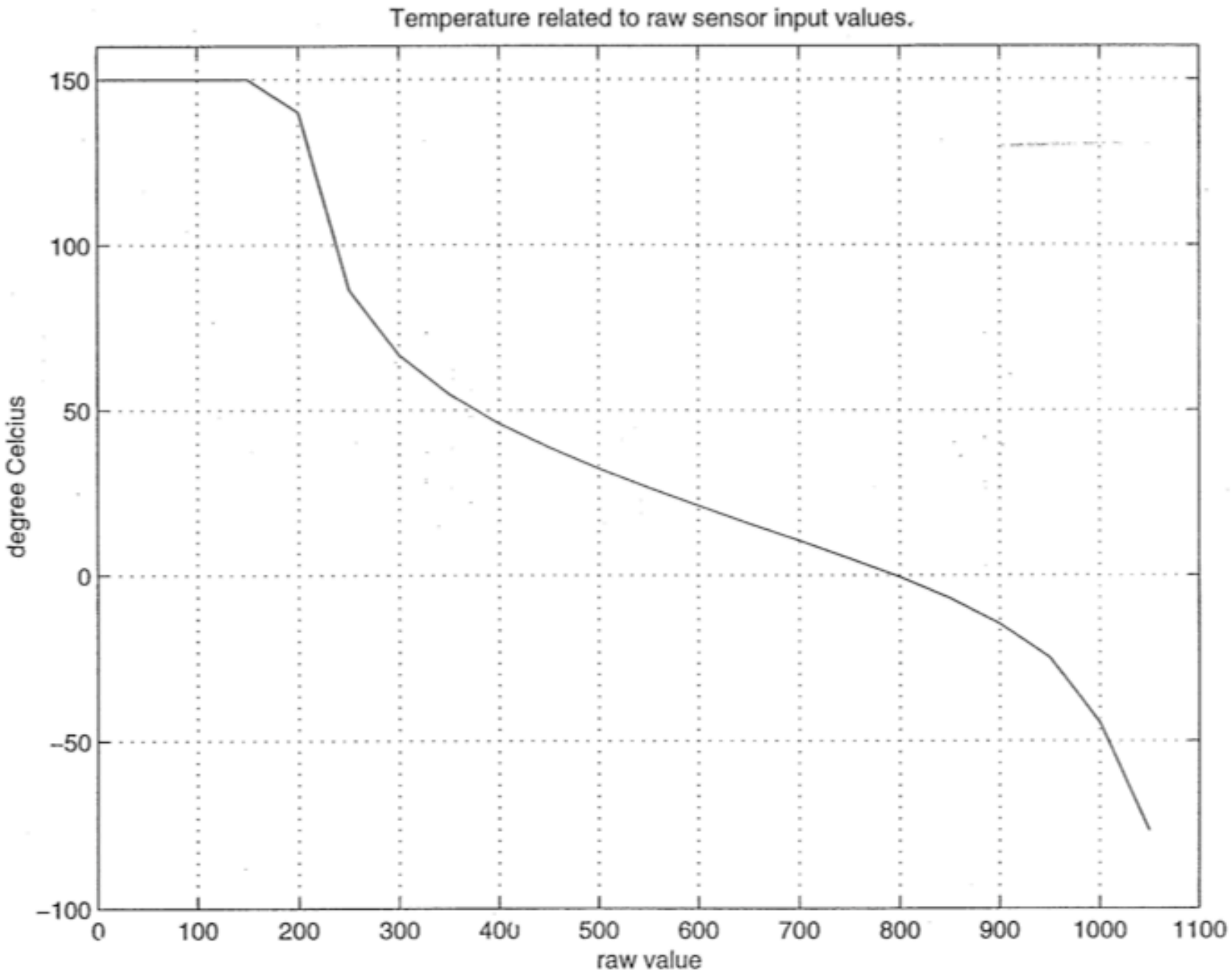


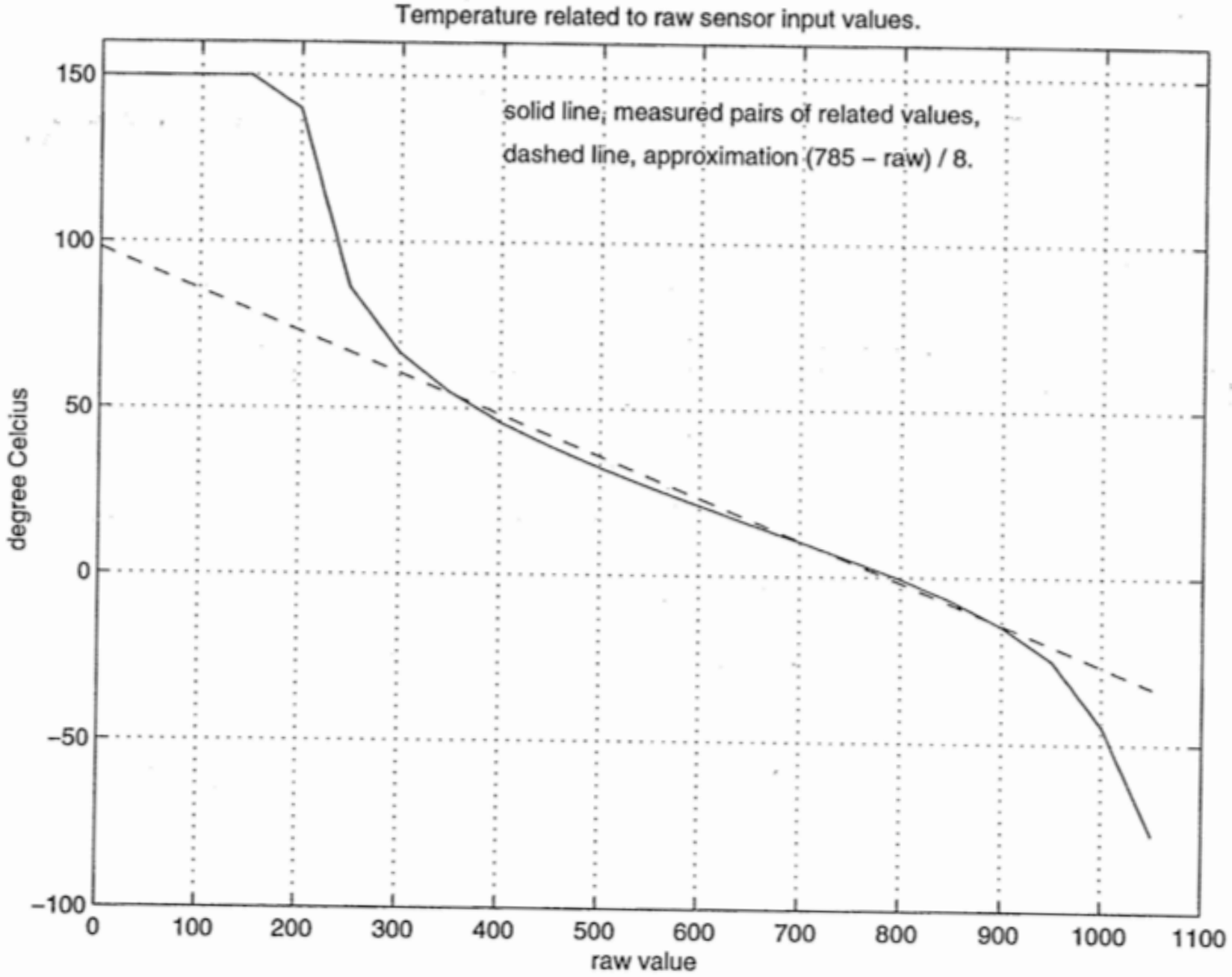
Raw Value(R) = 1023 * R/(R+10000) ohm


Raw sensor input values as a function of resistance.
Raw value
Resistance (Ohm)
x 10^4
solid line, measured pairs of related values,
dashed line, Raw = 1023*R/(R+10000)
0
100
200
300
400
500
600
700
800
900
1000
0
2
4
6
8
10
12

Signal-to-symbol




Temperature related to raw sensor input values.
150
100
50
0
−50
−100
degree Celcius
0
100
200
300
400
500
600
700
800
900
1000
1100
raw value


Temperature related to raw sensor input values.
solid line, measured pairs of related values,
dashed line, approximation (785 – raw) / 8.
150
100
50
0
−50
−100
degree Celcius
0
100
200
300
400
500
600
700
800
900
1000
1100
raw value

Natural light	Lux
Bright sun	50.000 - 100.000
Hazy day	25.000 - 50.000
Cloudy bright	10.000 - 25.000
Cloudy dull	2000 - 10.000
Very dull	100 - 2000
Sunset	1-100
Full moon	0.01 - 0.2
Starlight	0.001 - 0.001
Artificial light	**Lux**
Operating room	5000 - 10.000
Shop windows	1000 - 5000
Drawing office	300-500
Office	200-300
Living room	50-200
Corridors	50-100
Good street light	20
Poor street light	0.1

Table 1: Examples of typical lux values

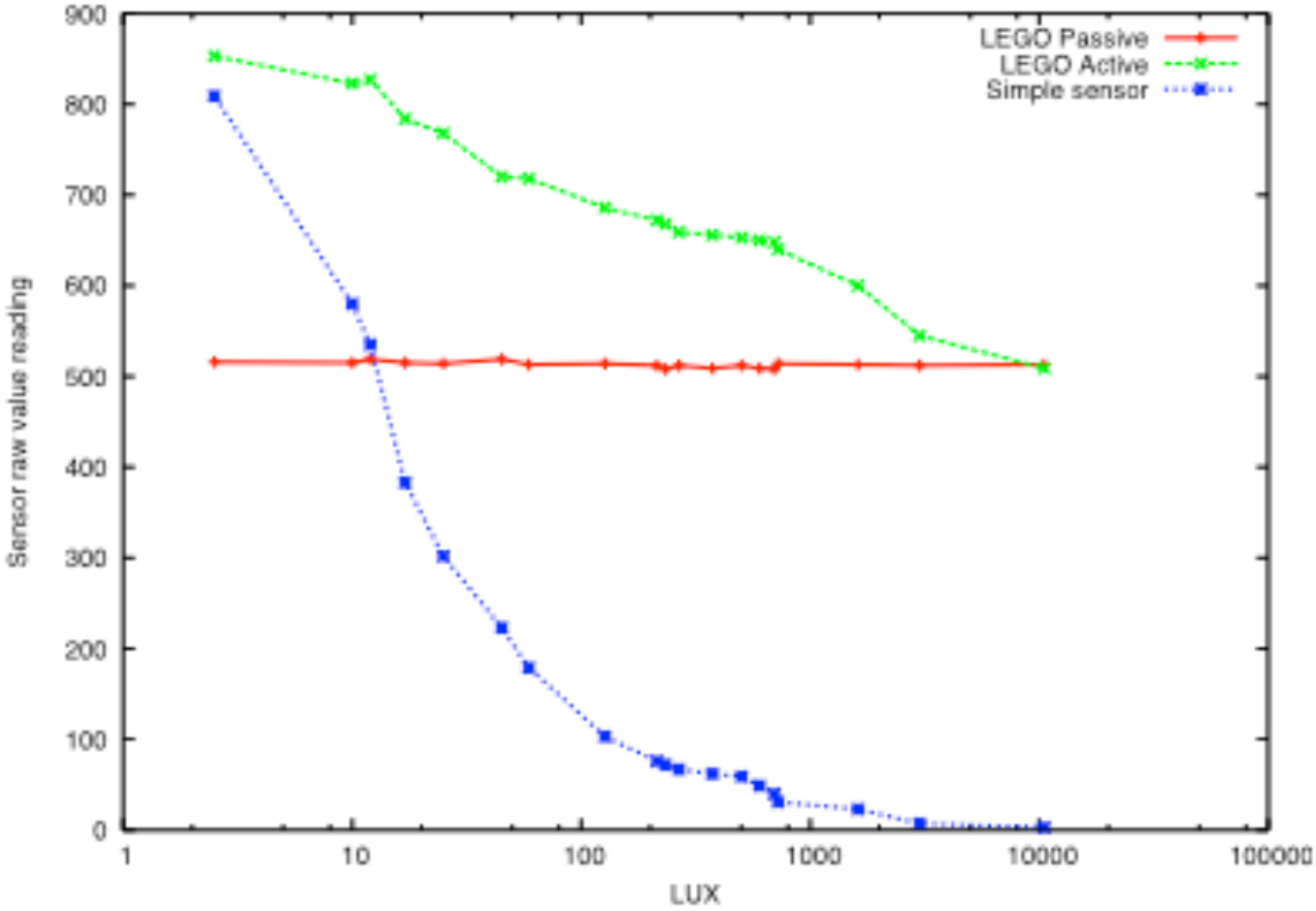


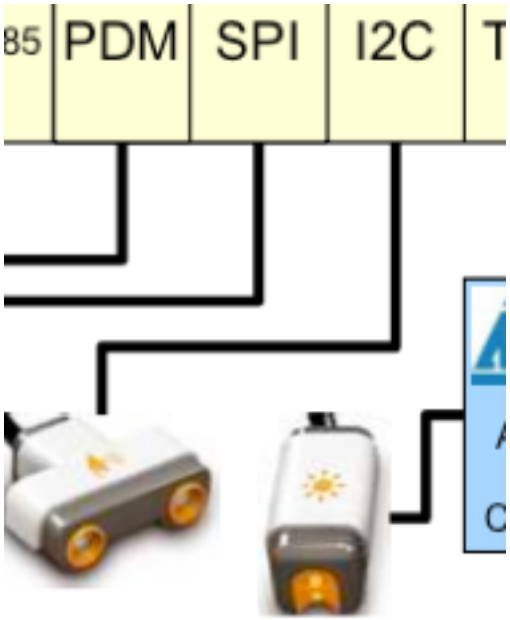
Figure 4: Raw values from the sensors under different light conditions.

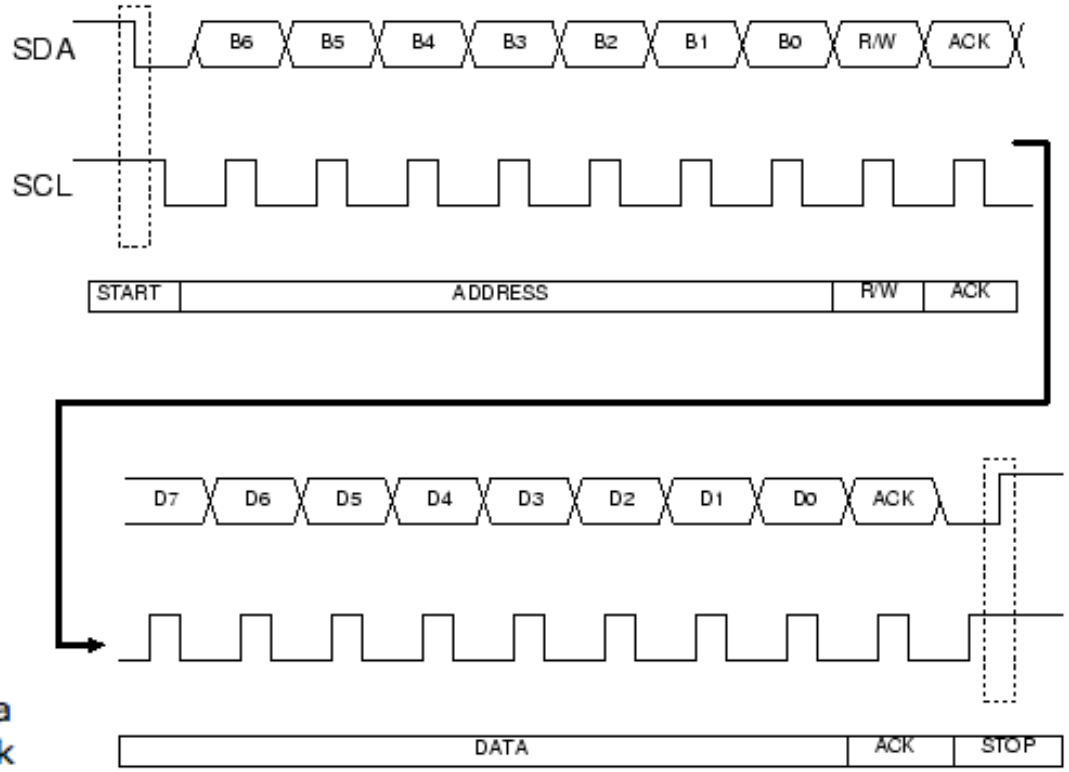
Sensor	White	Yellow	Orange	Green	Magenta	Brown	Blue	Red	Black
Passive	518	517	518	518	518	517	518	518	518
Active	686	688	686	729	698	713	742	729	760
Simple	209	229	252	286	288	335	379	520	760

Table 5: Sensor values when pointed at different colors.

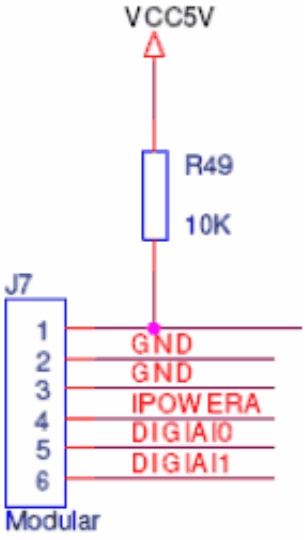
Sensor	White	Yellow	Orange	Green	Magenta	Brown	Blue	Red	Black
LEGO	699	702	698	743	710	726	758	729	795
Simple	301	308	318	428	329	388	468	308	598

Table 6: Sensor color measurements without influence from ambient light.





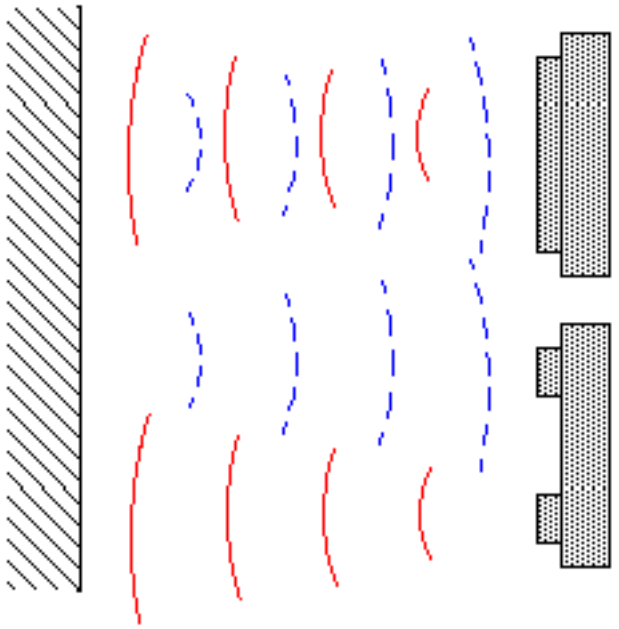
SDA : Serial Data
SCL : Serial Clock

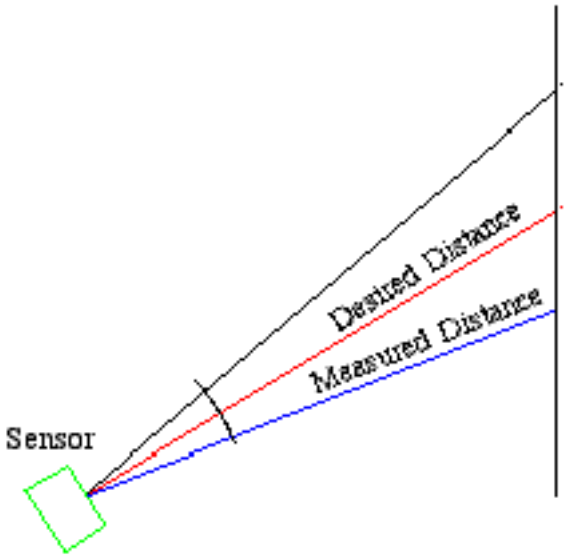


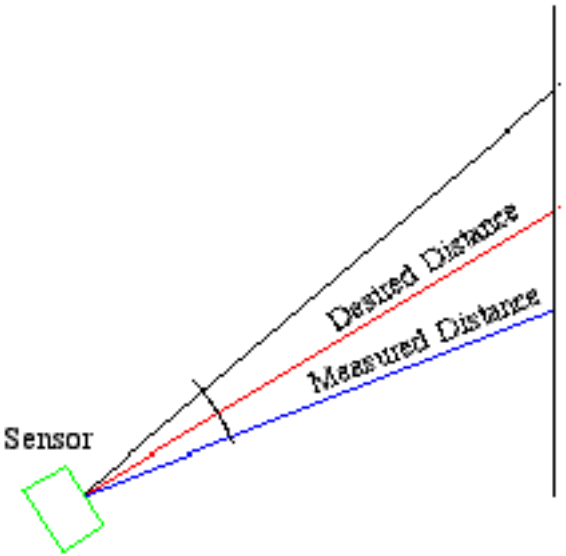
- Pin 1, ANA — Analog input and possible current output signal
- Pin 2, GND — Ground signal
- Pin 3, GND — Ground signal
- Pin 4, IPOWERA — 4.3 Volt output supply
- Pin 5, DIGIAI0 — Digital I/O pin connected to the ARM7 processor
- Pin 6, DIGIAI1 — Digital I/O pin connected to the ARM7 processor

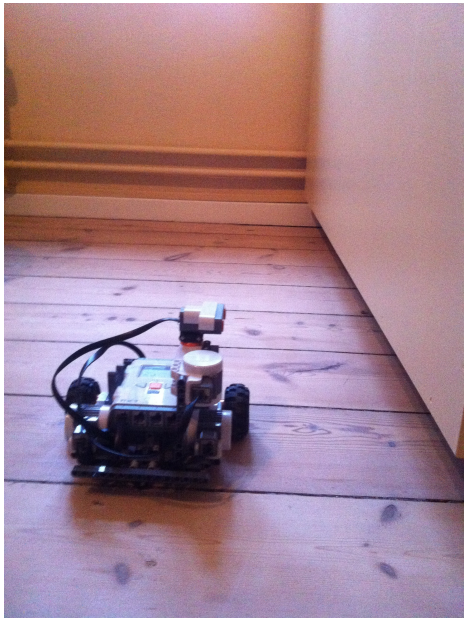
SCL : Serial Clock

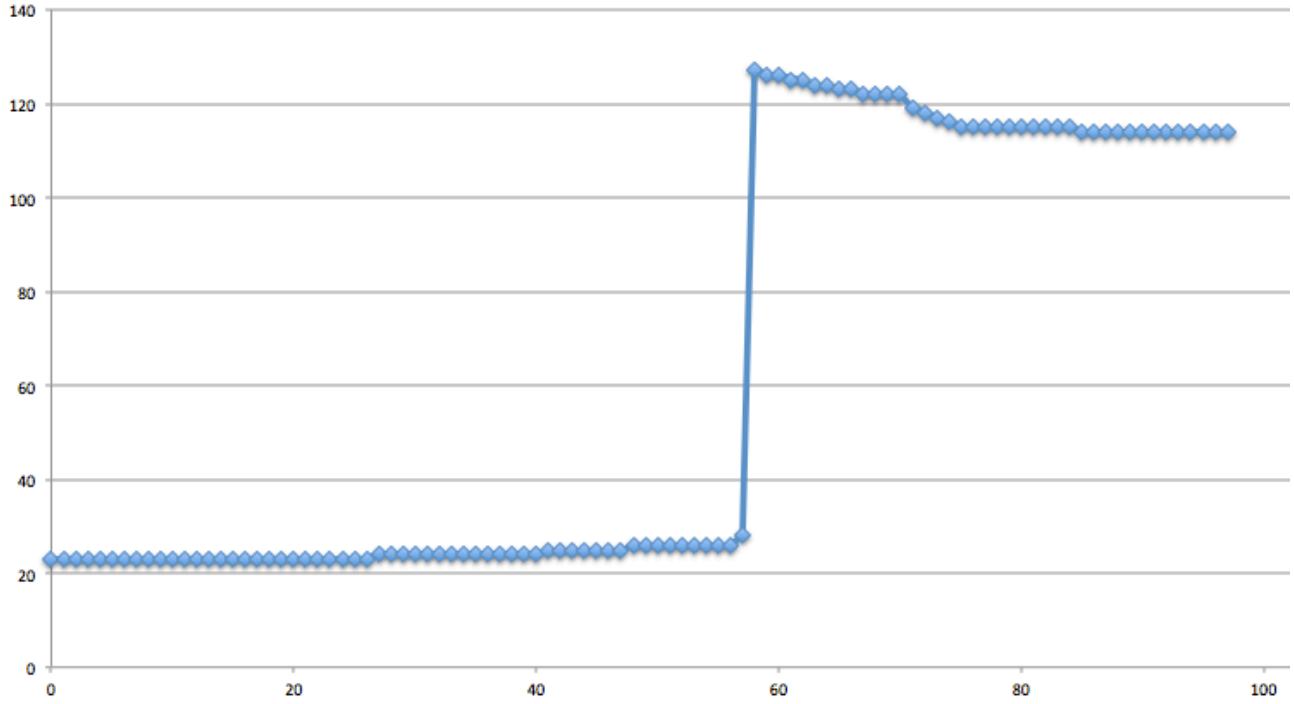
SDA : Serial Data

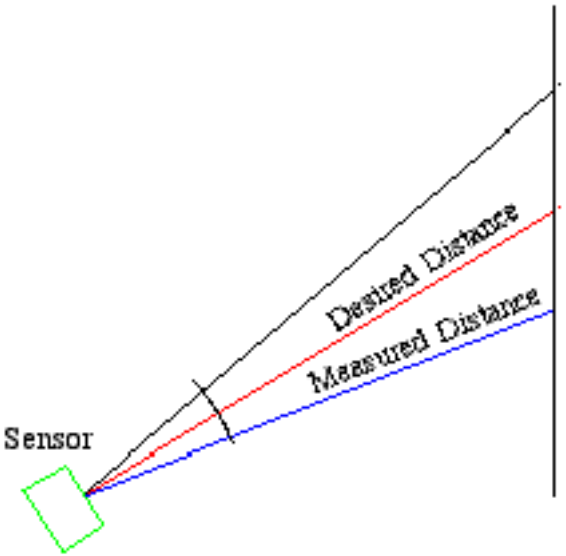


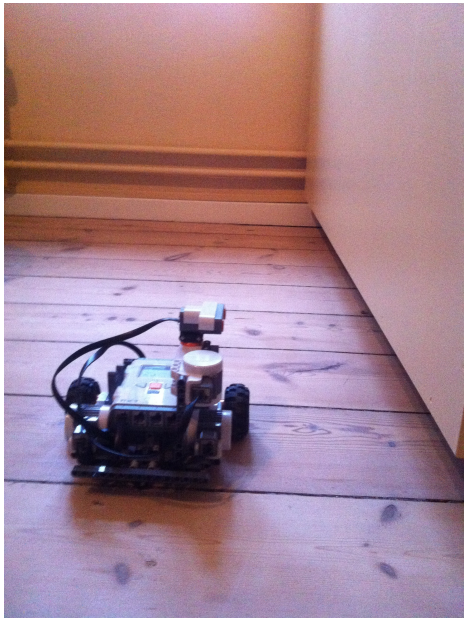

Desired Distance
Measured Distance
Sensor

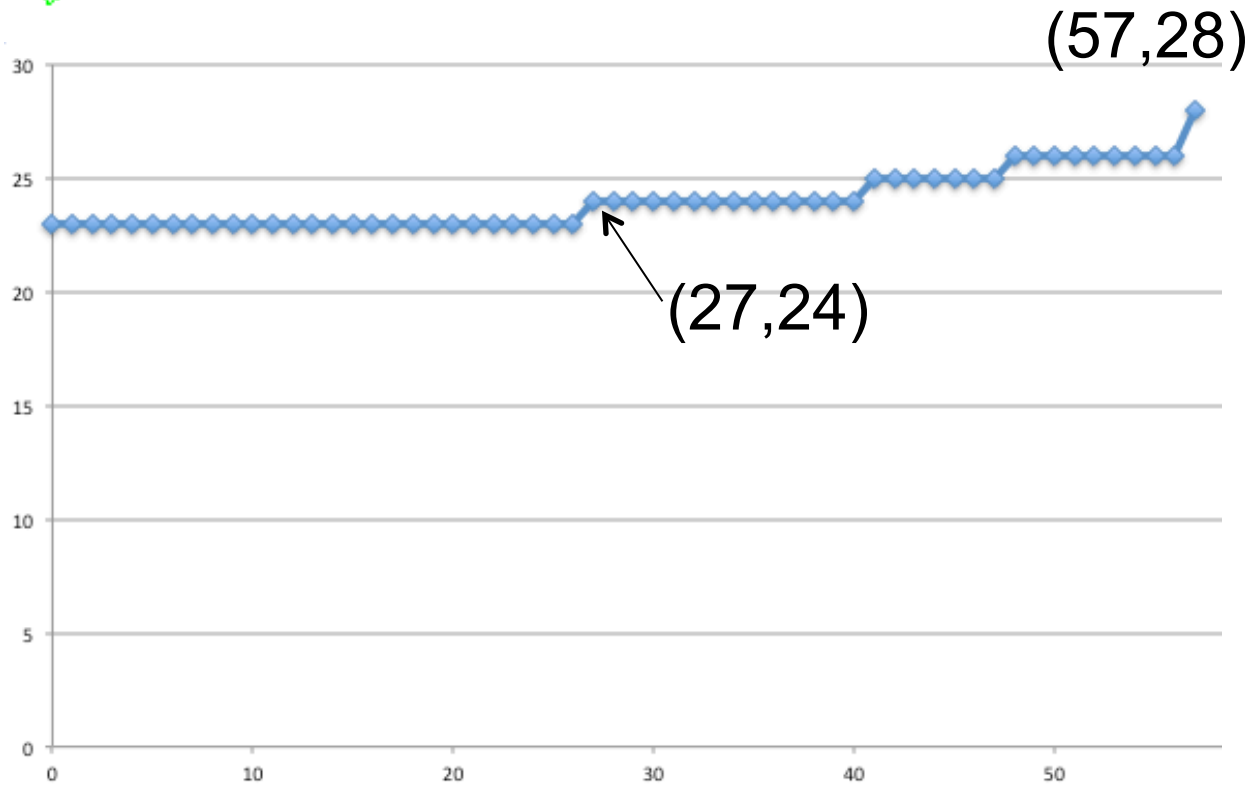

Sensor
Desired Distance
Measured Distance

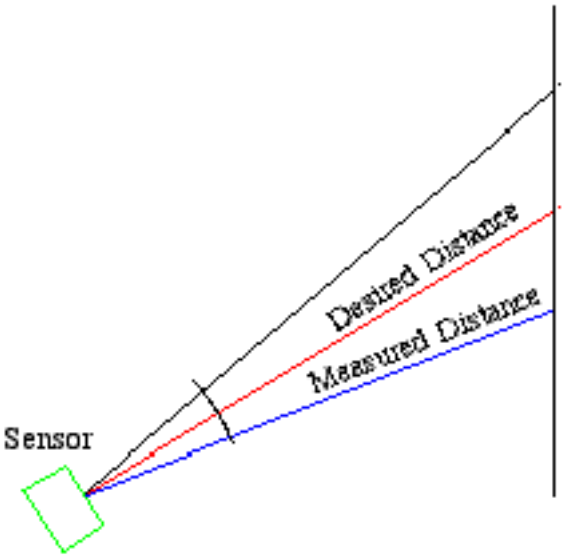


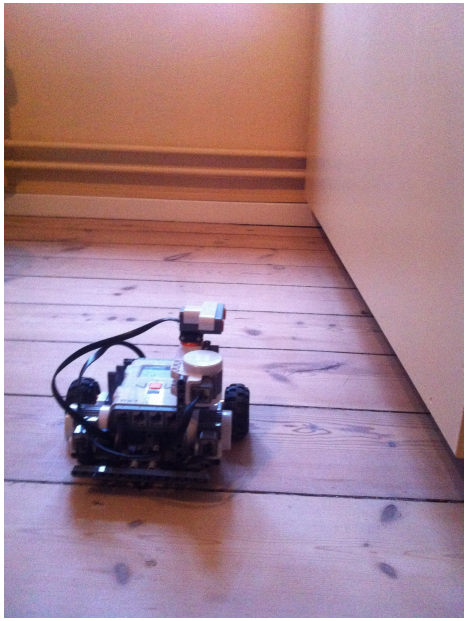

140
120
100
80
60
40
20
0
0
20
40
60
80
100


Sensor
Desired Distance
Measured Distance

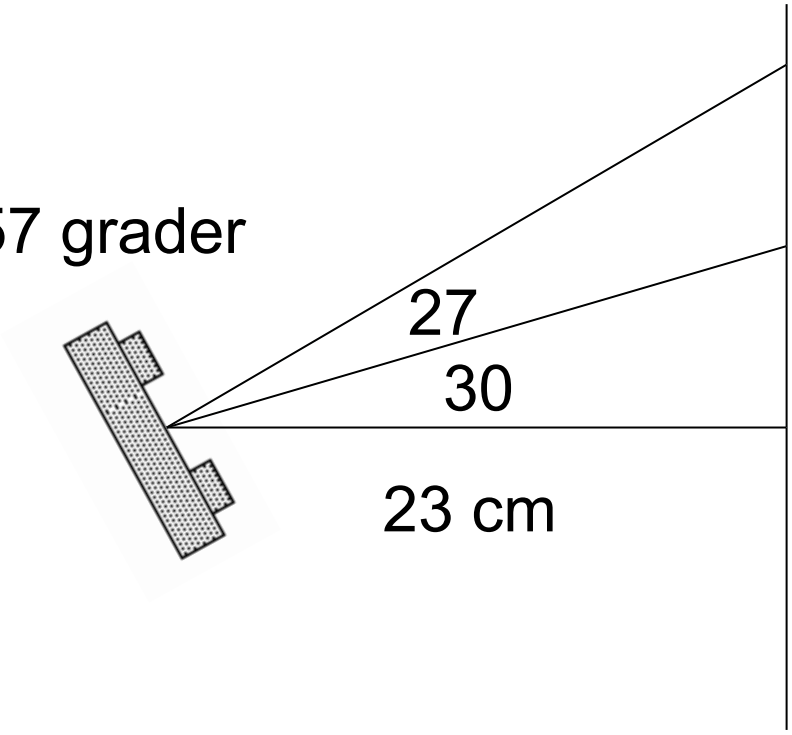



(57,28)
(27,24)
30
25
20
15
10
5
0
0
10
20
30
40
50



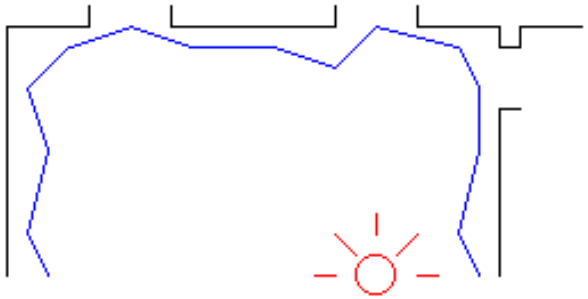


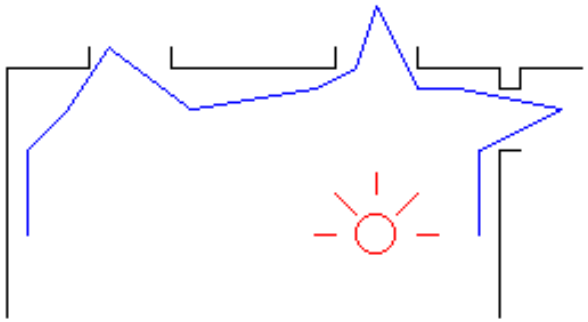
Ialt 57 grader

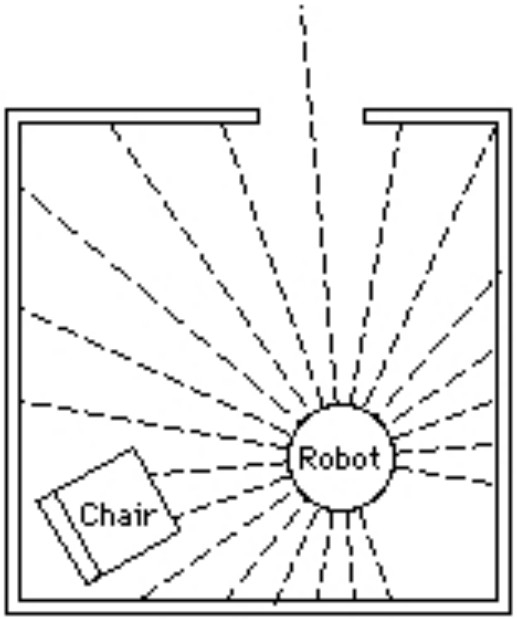


Measured Distance =

23/cos(30) ~ 27 cm






Robot
Chair

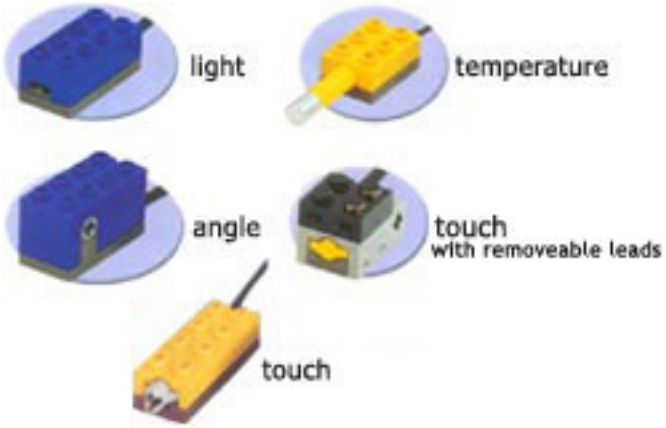

Doorway
Length of Echo
Chair
Scan moving from left to right extr

Embedded Java

The leJOS API classes provide access to the hardware. Especially the sensor ports and sensors:

TouchSensor

LightSensor



UltrasonicSensor

SensorPort

lejos.nxt

Class TouchSensor

java.lang.Object
  └ lejos.nxt.TouchSensor

All Implemented Interfaces:
SensorConstants

```
public class TouchSensor
extends Object
implements SensorConstants
```

Abstraction for a NXT touch sensor. Also works with RCX touch sensors.

Constructor Summary

TouchSensor(ADSensorPort port) Create a touch sensor object attached to the specified port.

Method Summary

boolean	**isPressed**() Check if the sensor is pressed.

```java
package lejos.nxt;

/**
 * Abstraction for a NXT touch sensor.
 * Also works with RCX touch sensors.
 *
 */
public class TouchSensor implements SensorConstants {
    ADSensorPort port;

    /**
     * Create a touch sensor object attached to the specified port.
     * @param port port, e.g. Port.S1
     */
    public TouchSensor(ADSensorPort port)
    {
       this.port = port;
       port.setTypeAndMode(TYPE_SWITCH, MODE_BOOLEAN);
    }
    /**
     * Check if the sensor is pressed.
     * @return <code>true</code> if sensor is pressed, <code>false</code> otherwise.
     */
    public boolean isPressed()
    {
        return (port.readRawValue() < 600);
    }

}
```

Interpret raw sensor value as pressed/released

```java
/**
 * An abstraction for a port that supports Analog/Digital sensors.
 *
 * @author Lawrie Griffiths.
 *
 * <br/><br/>WARNING: THIS CLASS IS SHARED BETWEEN THE classes AND pccomms PROJECTS.
 * DO NOT EDIT THE VERSION IN pccomms AS IT WILL BE OVERWRITTEN WHEN THE PROJECT IS BUILT.
 */
public interface ADSensorPort extends BasicSensorPort {

    public boolean readBooleanValue();

    public int readRawValue();

    public int readValue();
}
```

lejos.nxt

Class LightSensor

java.lang.Object
  └ lejos.nxt.LightSensor

All Implemented Interfaces:
SensorConstants, LampLightDetector, LightDetector

```
public class LightSensor
extends Object
implements LampLightDetector, SensorConstants
```

This class is used to obtain readings from a LEGO NXT light sensor. The light sensor can be calibrated to low and high values.

Constructor Summary

LightSensor(ADSensorPort port)
Create a light sensor object attached to the specified port.

readValue

```
public int readValue()
```

Get the light reading

Returns:
the light level

```
private int _zero = 1023;
private int _hundred = 0;
```

```
public int getLightValue()
{
        if(_hundred == _zero) return 0;
        return 100*(port.readRawValue() - _zero)/(_hundred - _zero);
}

/**
 * Get the light reading
 *
 * @return the light level
 */
public int readValue() {
        // TODO: Deprecate some of these read methods.
        return getLightValue();
}
```

$$light\ value = 100 \times (raw\ value - 1023)/(0 - 1023)$$

lejos.nxt

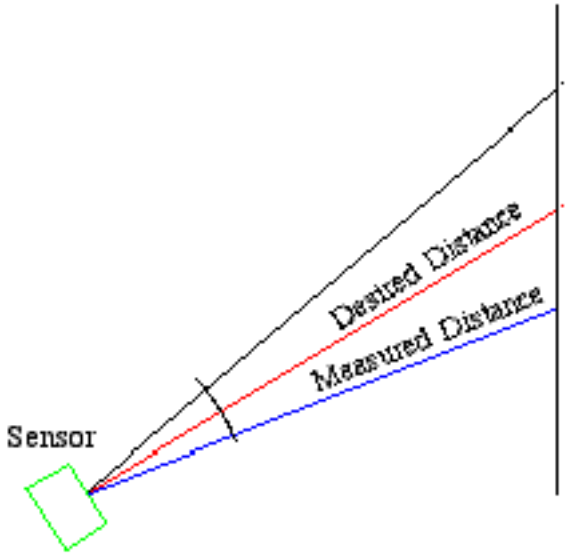
Class UltrasonicSensor

java.lang.Object
  └ lejos.nxt.I2CSensor
    └ **lejos.nxt.UltrasonicSensor**

All Implemented Interfaces:
SensorConstants

```
public class UltrasonicSensor
extends I2CSensor
```

Abstraction for a NXT Ultrasonic Sensor.



getDistance

```
public int getDistance()
```

Return distance to an object. To ensure that the data returned is valid this method may have to wait a short while for the distance data to become available.

Returns:
distance or 255 if no object in range

Method Summary

int	**capture**() Set capture mode Set the sensor into capture mode.
int	**continuous**() Switch to continuous mode.
int	**getActualMode**() Returns the current operating mode of the sensor.
int	**getCalibrationData**(byte[] data) Return 3 bytes of calibration data.
int	**getContinuousInterval**() Return the interval used in continuous mode.
int	**getData**(int register, byte[] buf, int off, int len) Executes an I2C read transaction and waits for the result.
int	**getDistance**() Return distance to an object.
int	**getDistances**(int[] dist) Return an array of distances.

```
public class UltrasonicSensor extends I2CSensor implements RangeFinder
{
        /* Device modes */
        public static final byte MODE_OFF = 0x00;
        public static final byte MODE_PING = 0x01;
        public static final byte MODE_CONTINUOUS = 0x02;
        public static final byte MODE_CAPTURE = 0x03;
        public static final byte MODE_RESET = 0x04;
        /* Device control locations */
        private static final byte REG_FACTORY_DATA = 0x11;
        private static final byte REG_UNITS = 0x14;
        private static final byte REG_CONTINUOUS_INTERVAL = 0x40;
        private static final byte REG_MODE = 0x41;
        private static final byte REG_DISTANCE = 0x42;
        private static final byte REG_CALIBRATION = 0x4a;
        /* Device timing */
        private static final int DELAY_CMD = 5;
        private static final int DELAY_DATA_PING = 50;
        private static final int DELAY_DATA_OTHER = 30;

        private long nextCmdTime = 0;
        private long dataAvailableTime = 0;
        private byte mode = MODE_CONTINUOUS;
        private byte[] byteBuff = new byte[8];
```

```
/**
 * Return distance to an object. To ensure that the data returned is valid
 * this method may have to wait a short while for the distance data to
 * become available.
 *
 * @return distance or 255 if no object in range or an error occurred
 */
public int getDistance()
{
        int delay;
        switch (mode)
        {
                case MODE_OFF:
                        throw new IllegalStateException("sensor is off");
                case MODE_PING:
                        delay = DELAY_DATA_PING;
                        break;
                default:
                        delay = DELAY_DATA_OTHER;
        }

        waitUntil(dataAvailableTime);
        int ret = getData(REG_DISTANCE, byteBuff, 1);
        if (ret < 0)
                return 255;

        // Make a note of when new data should be available.
        dataAvailableTime = System.currentTimeMillis() + delay;

        return byteBuff[0] & 0xFF;
}
```

lejos.nxt

Class I2CSensor

java.lang.Object
  └ **lejos.nxt.I2CSensor**

Method Summary

protected String	**fetchString**(byte reg, int len) Read a string from the device.
int	**getAddress**() Return the the I2C address of the sensor.
int	**getData**(int register, byte[] buf, int len) Executes an I2C read transaction and waits for the result.

lejos.nxt

Interface I2CPort

All Superinterfaces:

BasicSensorPort, SensorConstants

All Known Implementing Classes:

RemoteSensorPort, SensorPort

```
public interface I2CPort
extends BasicSensorPort
```

Abstraction for a port that supports I2C sensors.

Method Summary

void	**i2cDisable**() Disable the device.
void	**i2cEnable**(int mode) Enable the low level device
int	**i2cStatus**() Check to see the status of the port/device
int	**i2cTransaction**(int deviceAddress, byte[] writeBuf, int writeOffset, int writeLen, byte[] readBuf, int readOffset, int readLen) High level i2c interface.

lejos.nxt

Class SensorPort

java.lang.Object
  └ **lejos.nxt.SensorPort**

All Implemented Interfaces:
ADSensorPort, BasicSensorPort, I2CPort, LegacySensorPort, ListenerCaller, SensorConstants

---

```
public class SensorPort
extends Object
implements LegacySensorPort, I2CPort, ListenerCaller
```

Abstraction for a NXT input port.

Field Summary

static SensorPort[]	**PORTS** Array containing all three ports [0..3].
static SensorPort	**S1** Port labeled 1 on NXT.
static SensorPort	**S2** Port labeled 2 on NXT.
static SensorPort	**S3** Port labeled 3 on NXT.
static SensorPort	**S4** Port labeled 4 on NXT.

```
public class SensorPort implements LegacySensorPort, I2CPort, ListenerCaller
{



  /**
   * Port labeled 1 on NXT.
   */
  public static final SensorPort S1 = new SensorPort(0);
  /**
   * Port labeled 2 on NXT.
   */
  public static final SensorPort S2 = new SensorPort(1);
  /**
   * Port labeled 3 on NXT.
   */
  public static final SensorPort S3 = new SensorPort(2);
  /**
   * Port labeled 4 on NXT.
   */
  public static final SensorPort S4 = new SensorPort(3);
```

```
/**
 * <i>Low-level API</i> for reading sensor values.
 * Currently always returns the raw ADC value.
 * @param aPortId Port ID (0..4).
 * @param aRequestType ignored.
 */
private static native int readSensorValue(int aPortId);


/**
 * Low-level method to start an I2C transaction.
 * @param aPortId The port number for this device
 * @param address The I2C address of the device
 * @param writeBuffer The buffer for write operations
 * @param writeOffset Index of first byte to write
 * @param writeLen Number of bytes to write
 * @param readLen Number of bytes to read
 * @return < 0 if there is an error
 */
private static native int i2cStartById(int aPortId, int address,
        byte[] writeBuffer, int writeOffset, int writeLen, int readLen);

/**
 * Complete and I2C operation and retrieve any data read.
 * @param aPortId The Port number for the device
 * @param readBuffer The buffer to be used for read operations
 * @param offset Index of first byte to read
 * @param numBytes Number of bytes to read
 * @return < 0 if the is an error, or number of bytes transferred
 */
private static native int i2cCompleteById(int aPortId, byte[] readBuffer, int offset, int readLen);
```

AD sensors

Temperature
Sound
Light

I2C sensors

Ultrasonic
Compass
Accelerometer
RFID
GPS

AD sensors

Temperature
Sound
Light

readRawValue

```
public final int readRawValue()
```

Reads the raw value of the sensor.

Specified by:

`readRawValue` in interface `ADSensorPort`

Returns:

the raw sensor value

I2C sensors

Ultrasonic
Compass
Accelerometer
RFID
GPS

getAllAccel

```
public boolean getAllAccel(int[] dst,
                           int off)
```

Reads all 3 acceleration values into the given array. Elements off+0, off+1, and off+2 are filled with X, Y, and Z axis.

Parameters:

`dst` - destination array.

`off` - offset

Returns:

true on success, false on error

lejos.nxt

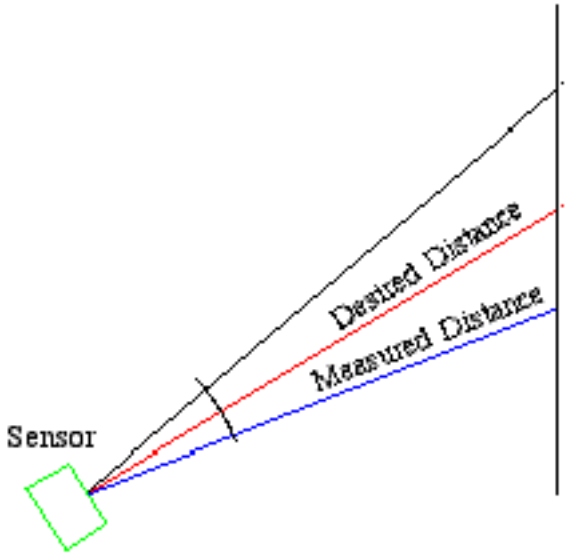
Class UltrasonicSensor

java.lang.Object
  └ lejos.nxt.I2CSensor
    └ **lejos.nxt.UltrasonicSensor**

All Implemented Interfaces:
SensorConstants

```
public class UltrasonicSensor
extends I2CSensor
```

Abstraction for a NXT Ultrasonic Sensor.



getDistance

```
public int getDistance()
```

Return distance to an object. To ensure that the data returned is valid this method may have to wait a short while for the distance data to become available.

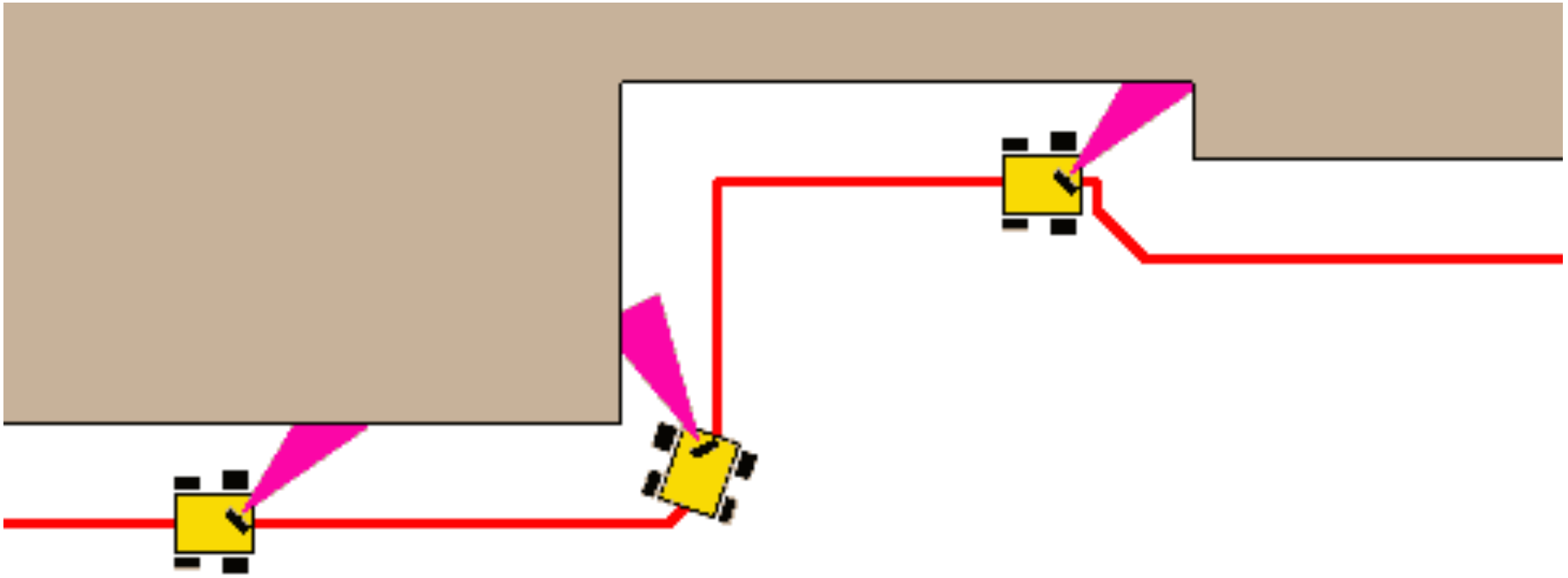
Returns:
distance or 255 if no object in range

Wall Follower

Philippe Hurbain has build and programmed a wall follower based on the LEGO Mindstorms RCX and a home build distance sensor, [5]. He used NQC (not quite c) to program the controller for the wall follower.

Exercise 6

Try to use his program and sensor placement to write a similar program in Java and make the LEGO 9797 car follow a wall. Compare the NQC control algorithm with the different suggestions on page 179, 5.1.3 exercises, [2].



- [5], Philippe Hurbain, WallFollower