Parachuting robot with barometer as altimeter

A barometer allows the robot to sense the air pressure, and use this to calculate the height. The NXT uses this information to deploy it's parachute on the appropriate time to land safely back on the ground.

Techno Class

Industrial Design Department

University of Technology Eindhoven

15th of September 2008

by Niels Molenaar, s031540

n.molenaar @ student.tue.nl

Table of Contents

PARACHUTING ROBOT WITH BAROMETER AS ALTIMETER 1
THE SENSOR 3
Introduction 3
Electronics 3
Software 4
THE PARACHUTE 6
Introduction 6
Theory 6
Hardware 6
The casing 8
CONCLUSION 10
CODE 11
Arduino 11
Lejos 17

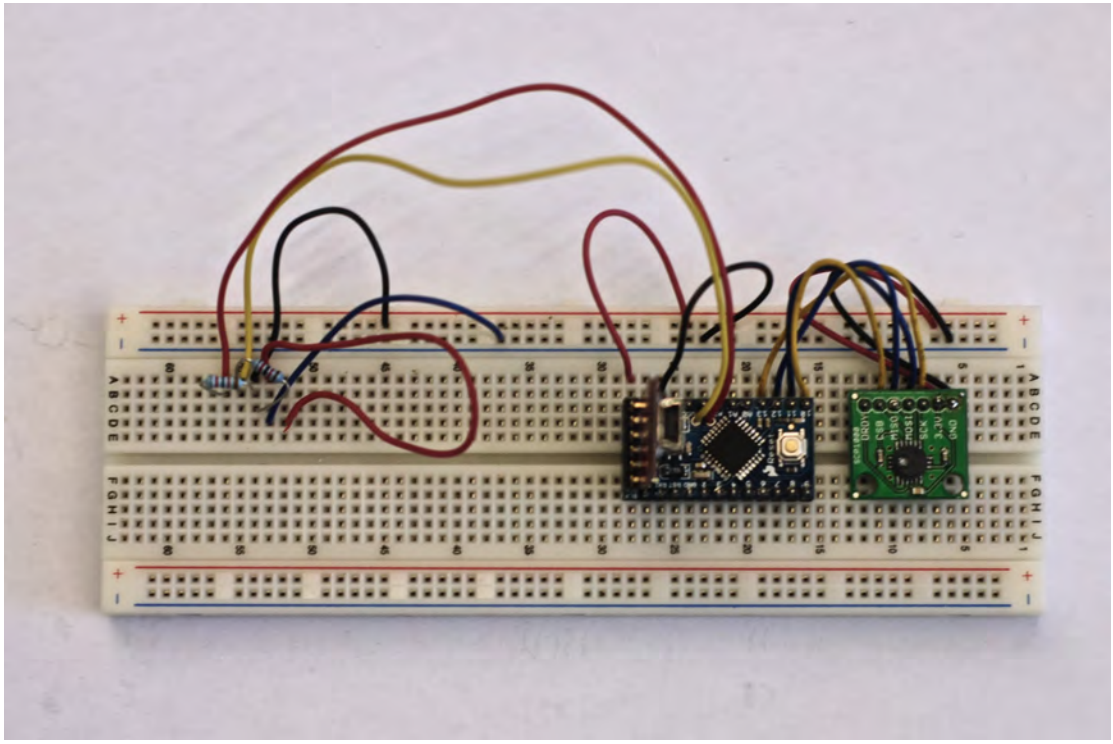
The sensor

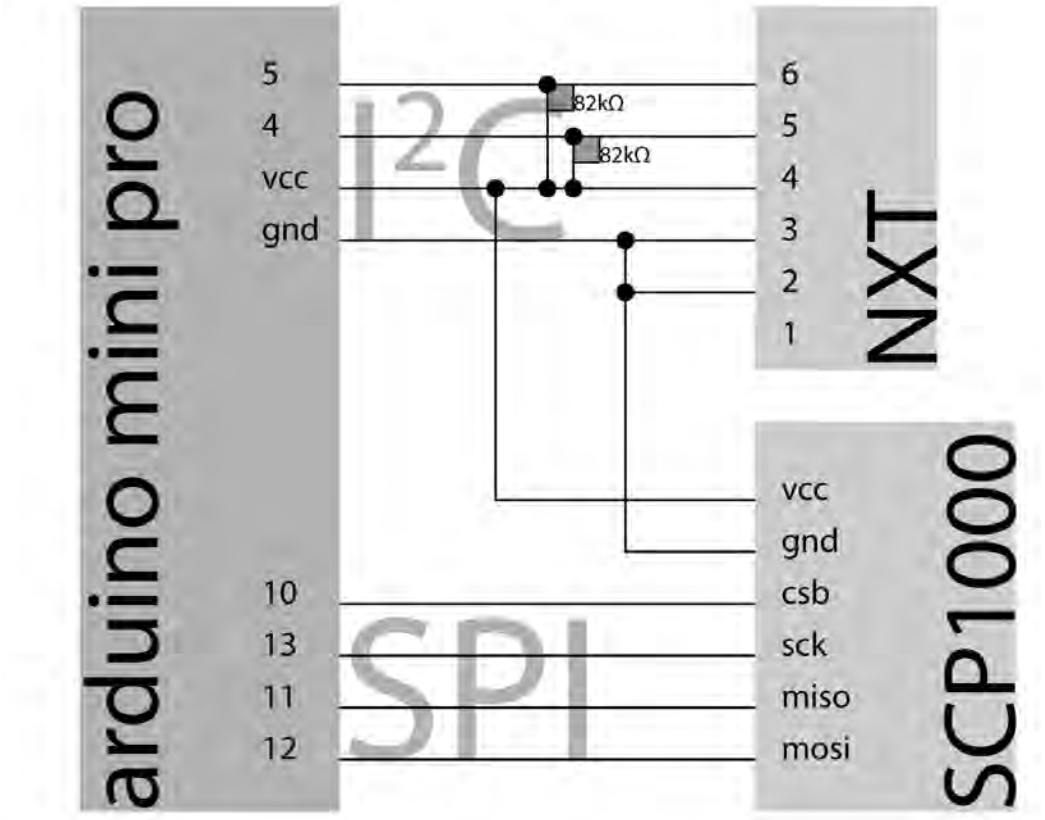
Introduction

The sensor is the part of the robot that translates the needed information of the real world into understandable data for the robot. The sensor consists of an Arduino Mini Pro uC and an SCP1000 barometric sensor.

Electronics

The uC interfaces the sensor with the NXT. This means it uses SPI on one end to retrieve data from the sensor, and on the other hand answers to the NXT with I^2C.





Software

The software consist of two parts. On one hand the uC controller is programmed with the Arduino language. This language has a library called Wire. This library can use the I²C protocol. The uC is set as slave since there is no need for the uC to push data to the NXT. When the NXT needs data, the uC is quered, and the uC answers by sending a byte translation of the amount of microbar.

Then the other part of the software is Lejos. This language is a Java environment suitable for controlling the NXT. This library has a built in class I2CSensor.

First a connection is established between the uC and the NXT by putting both on the same address, and preparing both for working with 6 bytes sent in an array.

When the connection is established, then the communication can begin. When the Sensor is connected, and the NXT turned on, the NXT can query for the air pressure. This is done simply by asking the uC for the array of bytes. Then, to make the robot aware of it's height, the robot needs to know the air pressure on ground level. It measures this by querying quickly 10 times in a row, and then uses the average as the ground level air pressure. This is to make this measurement more accurately. After this is done, the air pressure is sensitive to within 60 microbar ($^1/_3$ of a floor). An example of this is the next photograph. This time I was standing in the elevator and plotted the air pressure. What you can see are stops where people got in or out of the elevator.

SCP1000

sensor writes in register 9 times per second

Arduino mini pro

uC reads sensor register with SPI protocol 8.9 times per second
uC convert microbar in 6 bit byte array
uC answer I²C query with the pre-formatted byte

LEGO NXT

NXT can query for air-pressure on demand
NXT measures air pressure at ground level
NXT queries uC through I2C 8.8 times per second when in falling mode
NXT deploys parachute(runs motor) when pressure differential < 250(6 floors)

When the NXT is put into falling mode by pushing a button, the NXT queries the air pressure 8.8 times per second, and then subtracts this from the pressure measured at ground level. This way the NXT knows the difference in air pressure, and hereby knows it's height. This time, the NXT queries as much as possible and doesn't average the measured value. The amount of queries it needs to keep track of the falling height doesn't allow for such time consuming algorithms. The sensor is accurate enough, which means it might open slightly prematurely, but when it drops below the threshold, quickly the value will catch up.

The parachute

Introduction

The parachute is an example of possible use of the sensor. The barometer used as altimeter is suitable for going up into the sky, and falling back down to earth. The parachute is an interesting implementation since it can be used to implement safeties when parachuting. When the parachute isn't extracted on time, the system can safely deploy the parachute. On the other hand it might be an interesting addition to model rocket building, and bringing those safely back to earth.

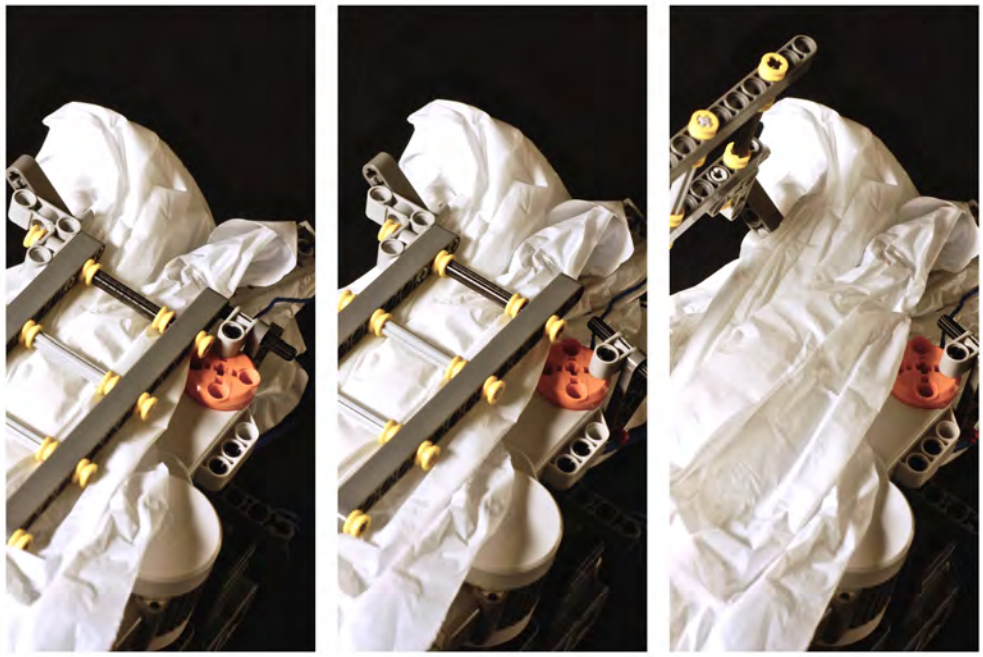
Theory

The NXT and it components weigh 450 grams. Everything landing below 10ft/sec in considered safe. To accomplish this a square parachute made from a trash bag is big enough. The trash bag, when folded open, is a square meter, which should decelerate the NXT to 8,25 ft/sec or 9 km/h (http://www.onlinetesting.net/cgi-bin/descent3.3.cgi). When there wouldn't be a parachute or when it fails, I want to know the terminal velocity to figure out the damage gravity could do. With a robot of 450 grams and a cross section of 50 square cm, the terminal velocity is estimated at 159 ft/sec or 174 km/h (http://www.calctool.org/CALC/eng/aerospace/terminal). The parachute is therefore an important part of the system.

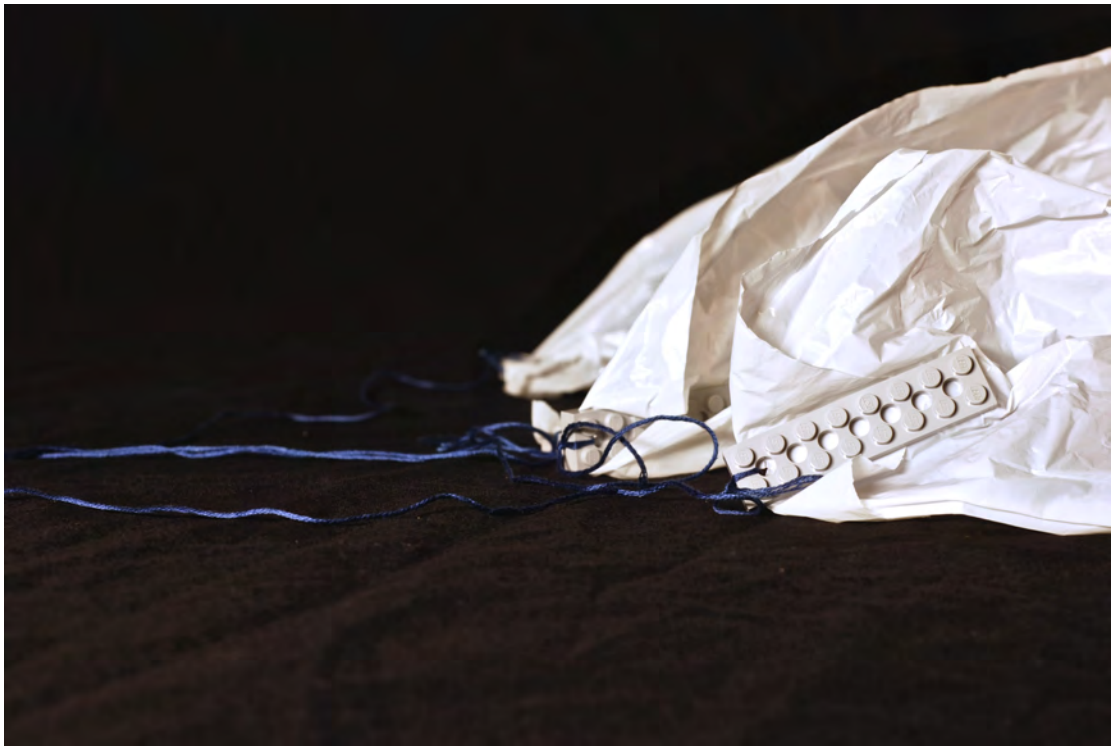
Hardware

The parachute deployment system uses standard LEGO parts. This is to keep down costs, and make it easy for other people to try the system. It is a box with a lid that is kept on by a bolt on an engine. When the engine turns, it pulls out the bolt, and the big parachute is pulled out by the little stabilisation

parachute. This release machanism is made in such a way that the direction of the force is perpendicular to the LEGO connections. LEGO isn't strong enough to hold itself together when a small parachute pulls on it. However, by using this method, the LEGO can hold.

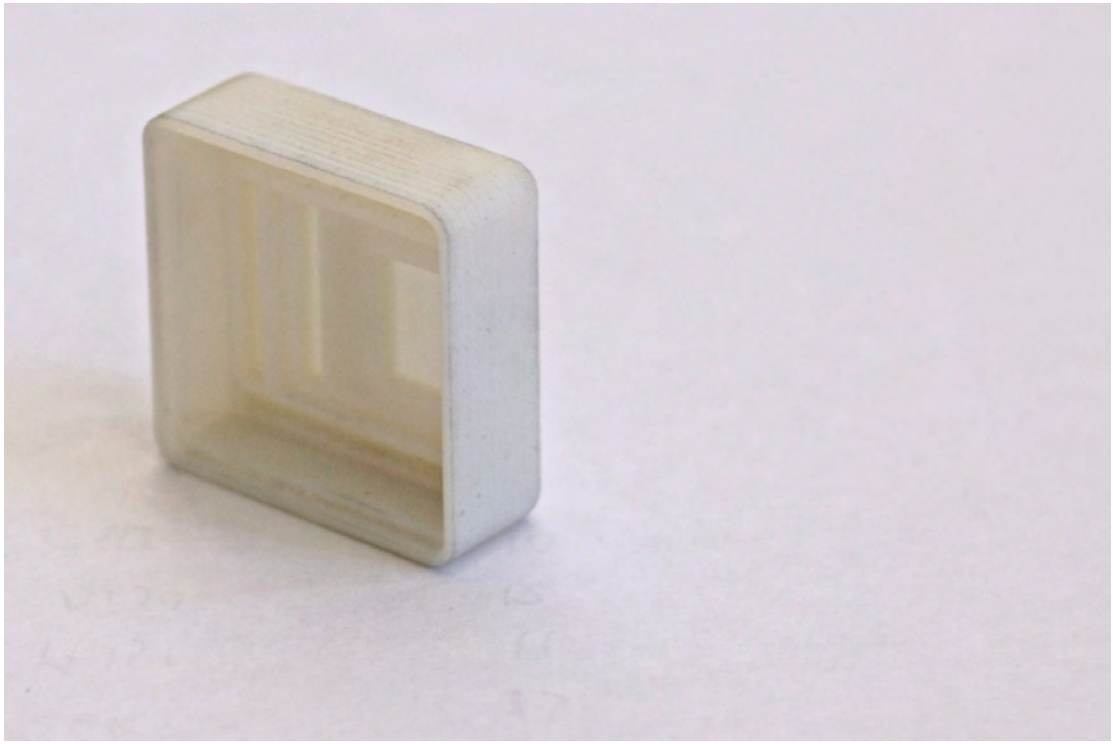


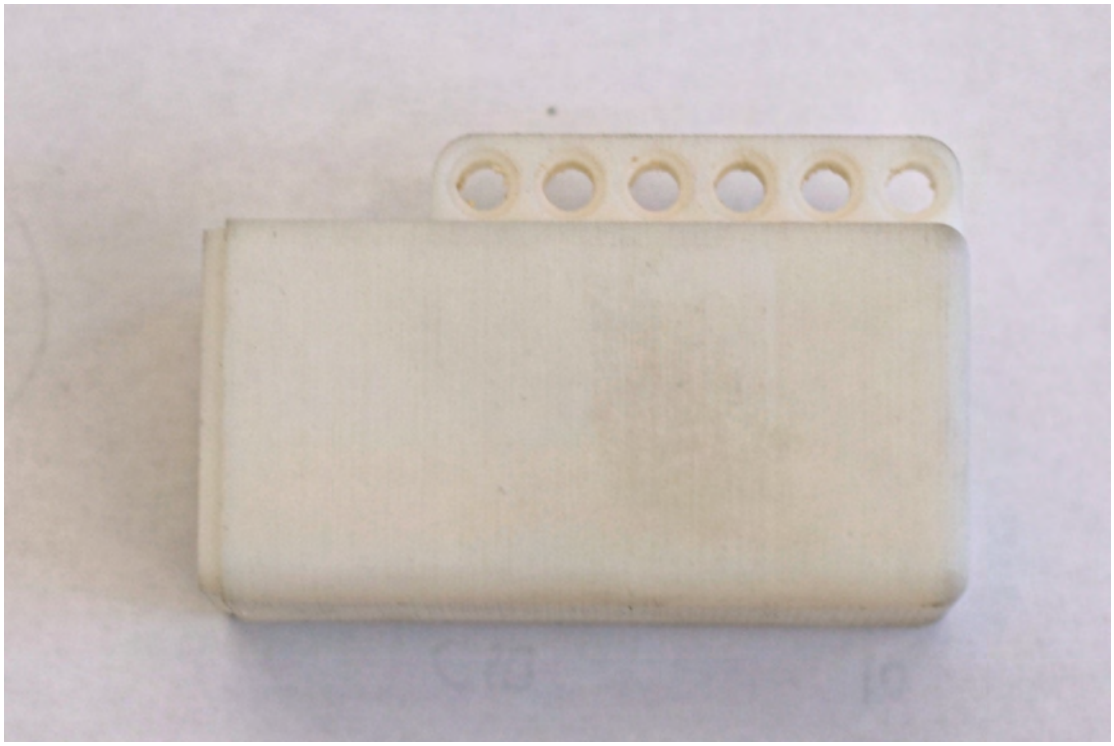
The parachute is made from trash bags with a small stabilization hole in the middle. To make the robot totally reliant on LEGO, the connection between the NXT and parachute is, besides the rope, completely build with LEGO.

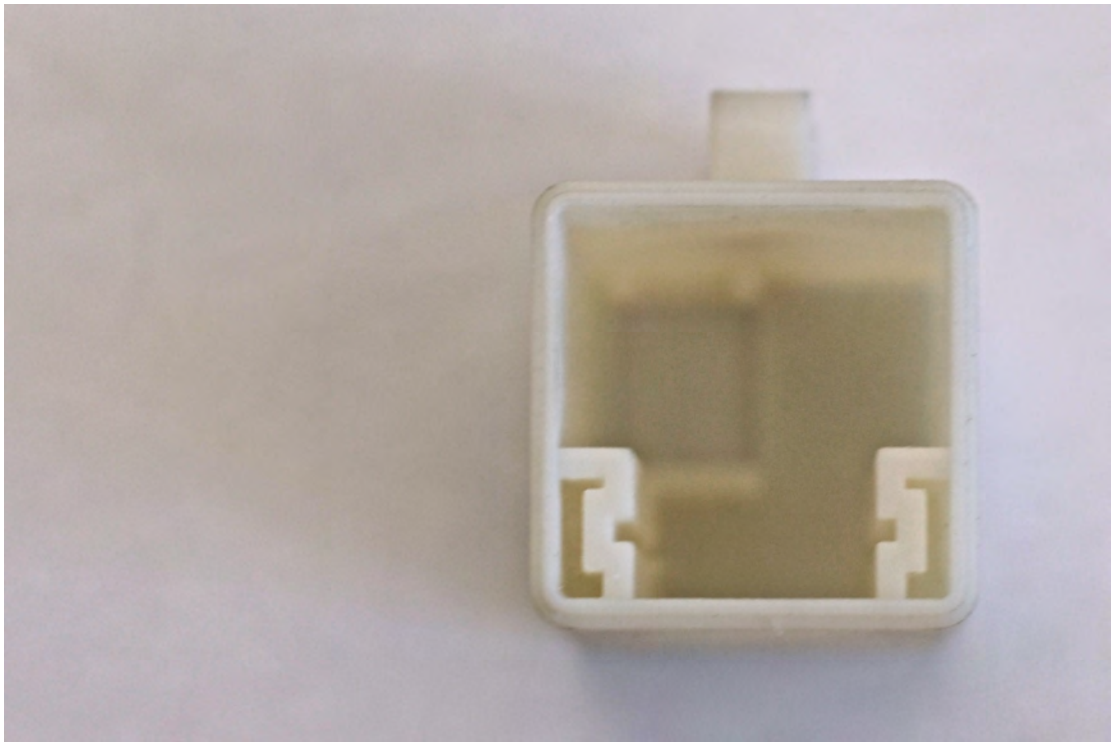


The casing

The casing is made with rapid prototyping, and so the the sensor fits snug within the casing. This way I can slide in the pressure sensor with Arduino, and then user rubber straps to make it all fit together without too much movement, and being able to withstand small impacts.







Conclusion

The act of a parachuting robot seems doable when taking in account that it takes a really long time before a parachute get opened by airflow, and delivers enough drag to slow down the robot. This is what went wrong with my attempt. As you can see on the movie (http://www.surfmedia.nl/app/video/GvmnqERkAIQTLLZlS3oIO4sV/play?format_id=6UQWdhl6VXTmZo8xFZQAEXuL&mode=object). The sensor did work, and the parachute got released. But then there wasn't enough time and height to open the parachute and it landed in a tree. The point then is, even though scary, that higher is better. It seems more dangerous since the robot can plummet to earth and fall into a lot of small pieces.

The software is simple enough to use by people used to programming in Lejos, but without I^2C knowledge. By just rapid prototyping the casing, and uploading the software to the desired components, one can create a rocket and measure the maximum height, or create safety components for parachutists, to ensure their parachute opens.

Code

Arduino

```
#include <string.h>
#include <Wire.h>   //Voor i2c communicatie

#define senderLength 6

// define spi bus pins
#define SLAVESELECT 10
#define SPICLOCK 13
#define DATAOUT 11          //MOSI
#define DATAIN 12    //MISO
#define UBLB(a,b)  ( ( (a) << 8) | (b) )
#define UBLB19(a,b) ( ( (a) << 16 ) | (b) )

//Addresses
#define REVID 0x00 //ASIC Revision Number
#define OPSTATUS 0x04   //Operation Status
#define STATUS 0x07     //ASIC Status
#define START 0x0A      //Constant Readings
#define PRESSURE 0x1F   //Pressure 3 MSB
#define PRESSURE_LSB 0x20 //Pressure 16 LSB
#define TEMP 0x21       //16 bit temp

char rev_in_byte;
int temp_in;
unsigned long pressure_lsb;
unsigned long pressure_msb;
unsigned long temp_pressure;
```

```
unsigned long pressure;
byte numbers[6];
unsigned long temp;


void setup()
{
  byte clr;
  pinMode(DATAOUT, OUTPUT);
  pinMode(DATAIN, INPUT);
  pinMode(SPICLOCK,OUTPUT);
  pinMode(SLAVESELECT,OUTPUT);
  digitalWrite(SLAVESELECT,HIGH); //disable device


  SPCR = B01010011; //MPIE=0, SPE=1 (on), DORD=0 (MSB first), MSTR=1 (master), CPOL=0 (clock idle when low), CPHA=0 (samples MOSI on rising edge), SPR1=0 & SPR0=0 (500kHz)
  clr=SPSR;
  clr=SPDR;
  delay(10);
  Serial.begin(9600);
  delay(500);


  /*Serial.println("Initialize High Speed Constant Reading Mode");
  write_register(0x03,0x09);*/
  Serial.println("Initialize High Resolution Constant Reading Mode");
  write_register(0x03,0x0A);
  /*Serial.println("Initialize Ultra Low Power Constant Reading Mode");
  write_register(0x03,0x0B);*/
```

```
  Wire.begin(27);               // join i2c bus with address 127
  Wire.onRequest(requestEvent);   // request event
}

void loop()
{
  /*Serial.print("PRESSURE [");
  Serial.print(pressure, DEC);
  Serial.println("]");

  temp_in = read_register16(TEMP);
  temp_in = temp_in / 20;
  //temp_in = ((1.8) *temp_in) + 32;
  Serial.print("TEMP C [");
  Serial.print(temp_in , DEC);
  Serial.println("]");*/
  converter();
  delay(112);
}

char spi_transfer(volatile char data)
{
  SPDR = data;                          // Start the transmission
  while (!(SPSR & (1<<SPIF)))     // Wait for the end of the transmission
  {
  };
  return SPDR;                          // return the received byte
}
```

```
char read_register(char register_name)
{
   char in_byte;
   register_name <<= 2;
   register_name &= B11111100; //Read command

   digitalWrite(SLAVESELECT,LOW); //Select SPI Device
   spi_transfer(register_name); //Write byte to device
   in_byte = spi_transfer(0x00); //Send nothing, but we should get back the
register value
   digitalWrite(SLAVESELECT,HIGH);
   delay(10);
   return(in_byte);

}

float read_register16(char register_name)
{
   byte in_byte1;
   byte in_byte2;
   float in_word;

   register_name <<= 2;
   register_name &= B11111100; //Read command

   digitalWrite(SLAVESELECT,LOW); //Select SPI Device
   spi_transfer(register_name); //Write byte to device
```

```
  in_byte1 = spi_transfer(0x00);
  in_byte2 = spi_transfer(0x00);
  digitalWrite(SLAVESELECT,HIGH);
  in_word = UBLB(in_byte1,in_byte2);
  return(in_word);
}

void converter()
{
 rev_in_byte = read_register(REVID);

 pressure_msb = read_register(PRESSURE);
 pressure_msb &= B00000111;
 pressure_lsb = read_register16(PRESSURE_LSB);
 /*pressure_lsb = read_register16(PRESSURE_LSB);
 pressure_lsb &= 0x0000FFFF;*/
 pressure = UBLB19(pressure_msb, pressure_lsb);
 pressure /= 4;

 numbers[0] = (pressure / 100000);
 unsigned long temptemp = 100000;
 temp = numbers[0] * temptemp;
 pressure = pressure - temp;

 numbers[1] = (pressure / 10000);
 temptemp = 10000;
 temp = numbers[1] * temptemp;
 pressure = pressure - temp;
```

```
  numbers[2] = (pressure / 1000);
  temptemp = 1000;
  temp = numbers[2] * temptemp;
  pressure = pressure - temp;

  numbers[3] = (pressure / 100);
  temptemp = 100;
  temp = numbers[3] * temptemp;
  pressure = pressure - temp;

  numbers[4] = (pressure / 10);
  temptemp = 10;
  temp = numbers[4] * temptemp;
  pressure = pressure - temp;

  numbers[5] = (pressure / 1);
  temptemp = 1;
  temp = numbers[5] * temptemp;
  pressure = pressure - temp;

  //Serial.println(numbers[5], DEC);
}

void requestEvent()
{
    Wire.send(numbers, senderLength);
    Serial.print("Request geweest");
```

```
}

void write_register(char register_name, char register_value)
{
  register_name <<= 2;
  register_name |= Booooooio; //Write command

  digitalWrite(SLAVESELECT,LOW); //Select SPI device
  spi_transfer(register_name); //Send register location
  spi_transfer(register_value); //Send value to record into register
  digitalWrite(SLAVESELECT,HIGH);
}
```

Lejos

```
import lejos.nxt.*;


public class i2CCommunication {


      // common.h
      // Common setup for NXT and Arduino
      // sT 28.01.2008
      static int pressure;
      static int normal;
      static boolean aboveLevel;


      public static void main(String[] args) {
            //TouchSensor touch = new
TouchSensor(SensorPort.S1);
            //SensorPort.S2.i2cEnable(LEGO_MODE);
            I2CSensor Arduino = new I2CSensor(SensorPort.S2);
            byte[] buf = new byte[6];

            for(int i = 0; i <6; i++){
                  buf[i] = 5;
            }
            Arduino.setAddress(27);
```

```
            LCD.clear() ;
            LCD.drawString("PleaseWait", 0, 0);
            try {
                  Thread.sleep(500);
            } catch (Exception e) {
            }
            LCD.clear() ;
            LCD.drawString("Ready", 0, 0);
            LCD.drawString("Left:CurrentPressure", 0, 2);
            LCD.drawString("Go:NormalizePressure", 0, 3);
            LCD.drawString("Right:JumpMode", 0, 4);
            LCD.drawString("Its a great day", 0, 6);
            LCD.drawString("TO FLY........", 0, 7);

            while(!Button.ESCAPE.isPressed()){
                  //LCD.refresh();
                  if(Button.ENTER.isPressed()){
                        LCD.clear() ;
                        LCD.drawString("PleaseWait", 0, 0);
                        try {
                              Thread.sleep(250);
                        } catch (Exception e) {
                        }
                        normal = 0;
                        for ( int i = 0; i < 10; i ++) {
                              Arduino.getData(0, buf, 6);
                              pressure = buf[0] * 100000 + buf[1]
* 10000 + buf[2] * 1000 + buf[3] * 100 + buf[4] * 10 + buf[5]
* 1;
                              normal = normal + pressure;
                              try {
                                    Thread.sleep(113);
                              } catch (Exception e) {
                              }
                        }
                        normal= normal / 10;
                        LCD.drawString("HeightNormalized", 0, 0);
                        LCD.drawInt(normal, 0, 1);
                        LCD.drawString("Press right to", 0, 3);
                        LCD.drawString("JUMP...", 0, 4);
                  }
                  if(Button.RIGHT.isPressed()){
                        /*LCD.clear() ;
                        for ( int i = 0; i < 100; i ++) {
                              Arduino.getData(0, buf, 6);
                              pressure = buf[0] * 100000 + buf[1]
* 10000 + buf[2] * 1000 + buf[3] * 100 + buf[4] * 10 + buf[5]
```

```
* 1;
                              LCD.setPixel (1 ,i , ((pressure -
99500) / 17));
                              try {
                                    Thread.sleep(1500);
                              } catch (Exception e) {
                              }
                        }*/
                        aboveLevel = true;
                        Motor.A.stop();
                        while (aboveLevel) {
                              LCD.clear();
                              LCD.drawString("ReadyToFall", 0,
0);
                              Arduino.getData(0, buf, 6);
                              pressure = buf[0] * 100000 + buf[1]
* 10000 + buf[2] * 1000 + buf[3] * 100 + buf[4] * 10 + buf[5]
* 1;
                              pressure = pressure - normal;
                              LCD.drawInt(pressure, 0, 1);
                              if (pressure > -450) {
                                    Motor.A.backward();
                                    try {
                                          Thread.sleep(1000);
                                    } catch (Exception e) {
                                    }
                                    Motor.A.flt();
                                    aboveLevel = false;
                              }
                              try {
                                    Thread.sleep(113);
                              } catch (Exception e) {
                              }
                        }
                  }
                  if(Button.LEFT.isPressed()){
                        LCD.clear();
                        Arduino.getData(0, buf, 6);
                        pressure = buf[0] * 100000 + buf[1] *
10000 + buf[2] * 1000 + buf[3] * 100 + buf[4] * 10 + buf[5] *
1;
                        LCD.drawInt(pressure, 0, 0);
                        LCD.drawInt((pressure - normal), 0, 1);
                        /*try {
                              Thread.sleep(200);
                        } catch (Exception e) {
                        }*/
```

```
                    }
                }
            }
        }
```