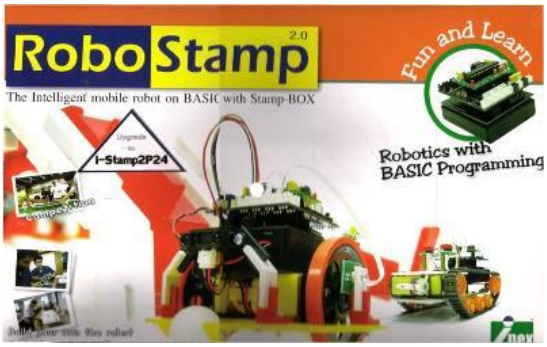
RoboStamp – Basic Software
(A tutorial by Technotutorz)

The Robostamp robotic kit is one of the robots used as standard in the Technotutorz workshops. Two versions can be built up: the izeBot and the RoboTank. The Robostamp can be used for collision detection, distance measurement, line tracking and remote control operation. The izeBot is ideal for a 1-day workshop as the students spend little time building up the robot before they can start with basic programming.

The Robostamp uses the i-Stamp2P24 microcontroller, which is programmed via the BASIC Stamp Editor from Parallax. Parallax developed PBasic, a simple, easy to learn language that is also well suited for this architecture, and highly optimized for embedded control. It includes many of the instructions featured in other forms of BASIC (GOTO, FOR...NEXT, IF...THEN…ELSE) as well as some specialised instructions (SERIN, PWM, BUTTON, COUNT and DTMFOUT).



The standard parts used in this tutorial are:

1. The izeBot with two switches and a distance sensor.
2. Basic Stamp Editor
3. Download the code for the switch and distance sensors.

The RoboStamp is available from Robokits on request and all the required code and user manuals are supplied with the kit.

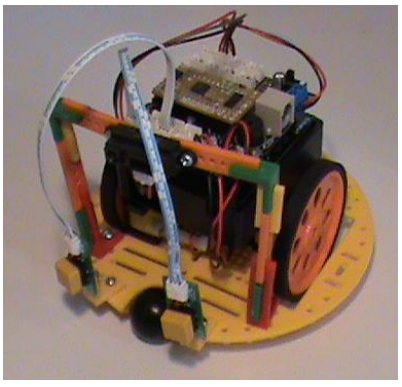
The following tutorial explains step by step instructions to help you program the izeBot. This tutorial covers the preparation, motor control, collision detection and distance sensing.

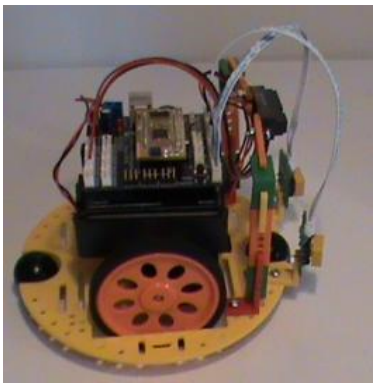
1 Preparation

1.1 Build up a standard izeBot using the instructions provided.

For this tutorial, the motors and sensors are connected as follows:

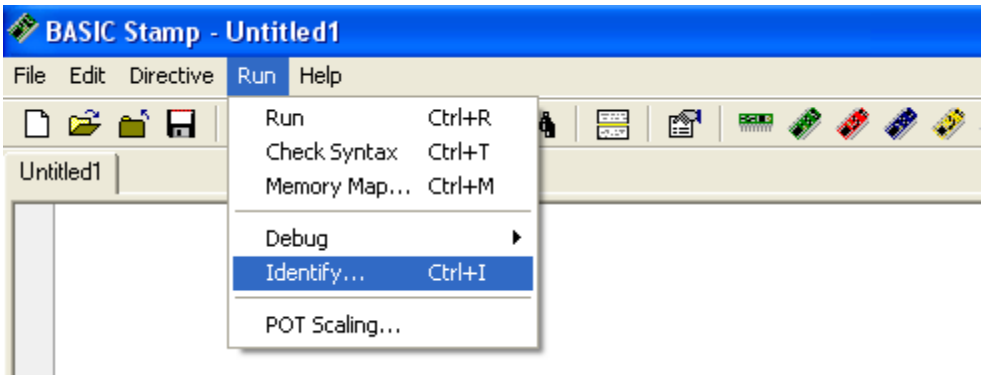
a. Left motor –> Connect to Motor B Direct input
b. Right motor –> Connect to Motor A Direct input
c. Left switch –> P1
d. Right switch –> P2
e. Distance sensor: Analog 3



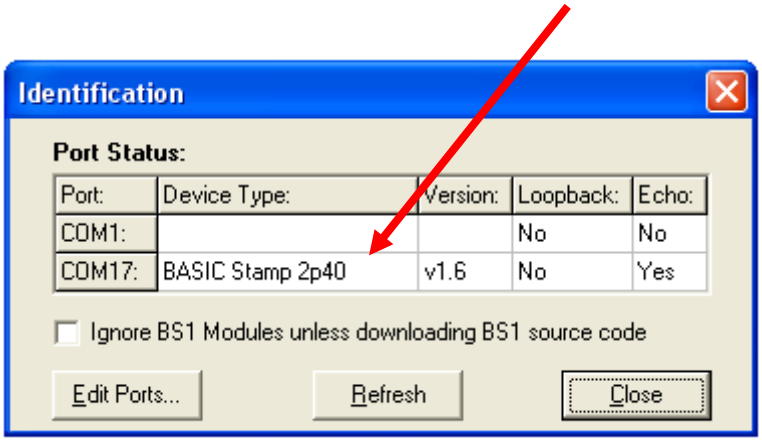


1.2 Open Basic Stamp Editor.

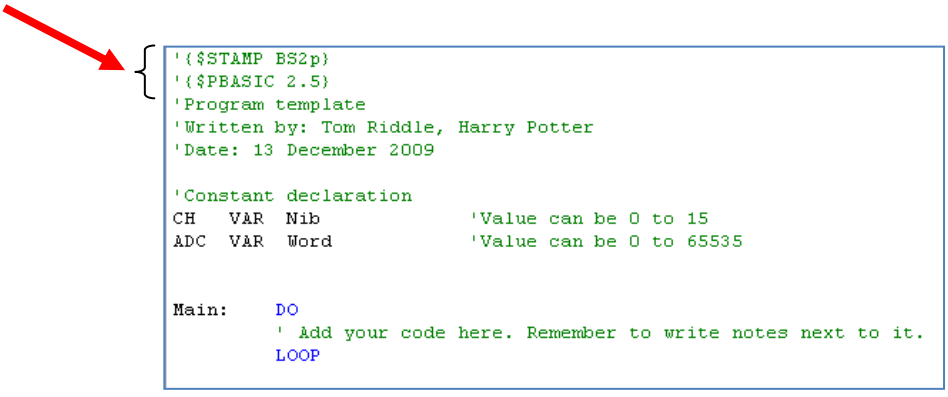
Turn on the EziBot and go to Run –> Indentify (Shortcut Ctrl I). See below.



The following window will appear. This shows you the type of Stamp attached.



The above value determines the $STAMP directive that is added at the top of the editor window. On this case BASIC Stamp 2p40 is equivalent to BS2p. The $PBASIC directive is underneath. Open the file BasicTemplate_RoboStamp.bsp and find these directives at the top of the editor window.



```
'{$STAMP BS2p}
'{$PBASIC 2.5}
'Program template
'Written by: Tom Riddle, Harry Potter
'Date: 13 December 2009

'Constant declaration
CH   VAR  Nib              'Value can be 0 to 15
ADC  VAR  Word             'Value can be 0 to 65535


Main:    DO
         ' Add your code here. Remember to write notes next to it.
         LOOP
```

2 Motor movement with the izeBOT

2.1 Operation

The izeBOT uses two motors that can be controlled independently. The motors are standard DC motors and the direction of the motor's turn depends on the electric current.

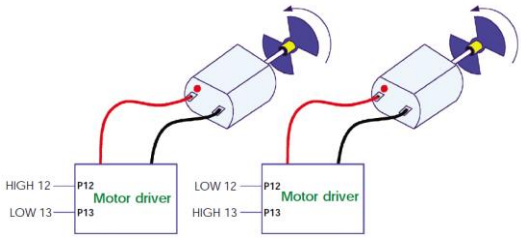


Figure 1. The diagram shows the motors changing direction depending which pin is high.

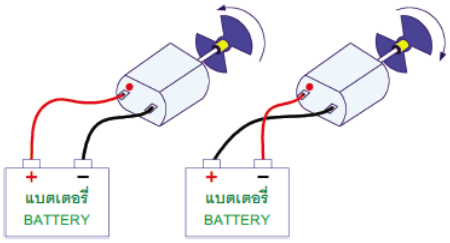


Figure 2. This figure is similar to figure 1 but showing the voltages.

The motor leads are connected via the motor drivers to Pin 12, 13, 14 and 15 of the microcontroller. Pin 12 and 13 control one motor and Pin 14 and 15 the other. The table below explains the connections. Complete the table.

Movement	Motor A			Motor B		
	P15	P14	Direction	P13	P12	Direction
Forward	High	Low	▲▲	High	Low	▲▲
Backward		High		Low		▼▼
Turn Left	Low		⊘		Low	
Turn Right	High					⊘
Rotate Left			▼▼		Low	▲▲
Rotate Right		Low		Low		▼▼
Stop	Low	Low				⊘

Figure 1. This table explains the settings when Motor A is the left hand motor and Motor B the right hand motor.

Code in PBasic 2.5:

To go forward: HIGH 13 : LOW 12 : HIGH 15 : LOW 14

Create a subroutine: (That can be called many times)

Label name: → Forward: HIGH 13 : LOW 12 : HIGH 15 : LOW 14 : RETURN

End of sub routine:

The subroutine is called from the main program. It will execute and when it gets to RETURN it will go back to the main program, one line after the previous command.

Create subroutines for the other movements:
(Complete the commands.)

Forward:	HIGH 13 : LOW 12 : HIGH 15 : LOW 14 : RETURN
Backward:	
Turn_Left:	
Turn_Right:	
Rotate_Left:	
Rotate_Right:	
Motor_Stop:	

Call in program as:
Gosub Forward

The standard program structure is shown below: (See the Basic Stamp Manual for more information)
The code in green – Comments, extra information and program documentation
The code in black – Normally for user defined variables, functions
The code in blue – Reserved words on PBasic

This template is a basic starting point for all programs.

$STAMP directive and $PBASIC directive required for all programs. This can be added by selective the correct directive at *Menu -> Directive -> Stamp and Menu -> Directive -> PBasic or short-cut keys shown below.*

Project information. This is to ensure you know what the program is about, who wrote it and when.

Not used yet

Functions/ Subroutines

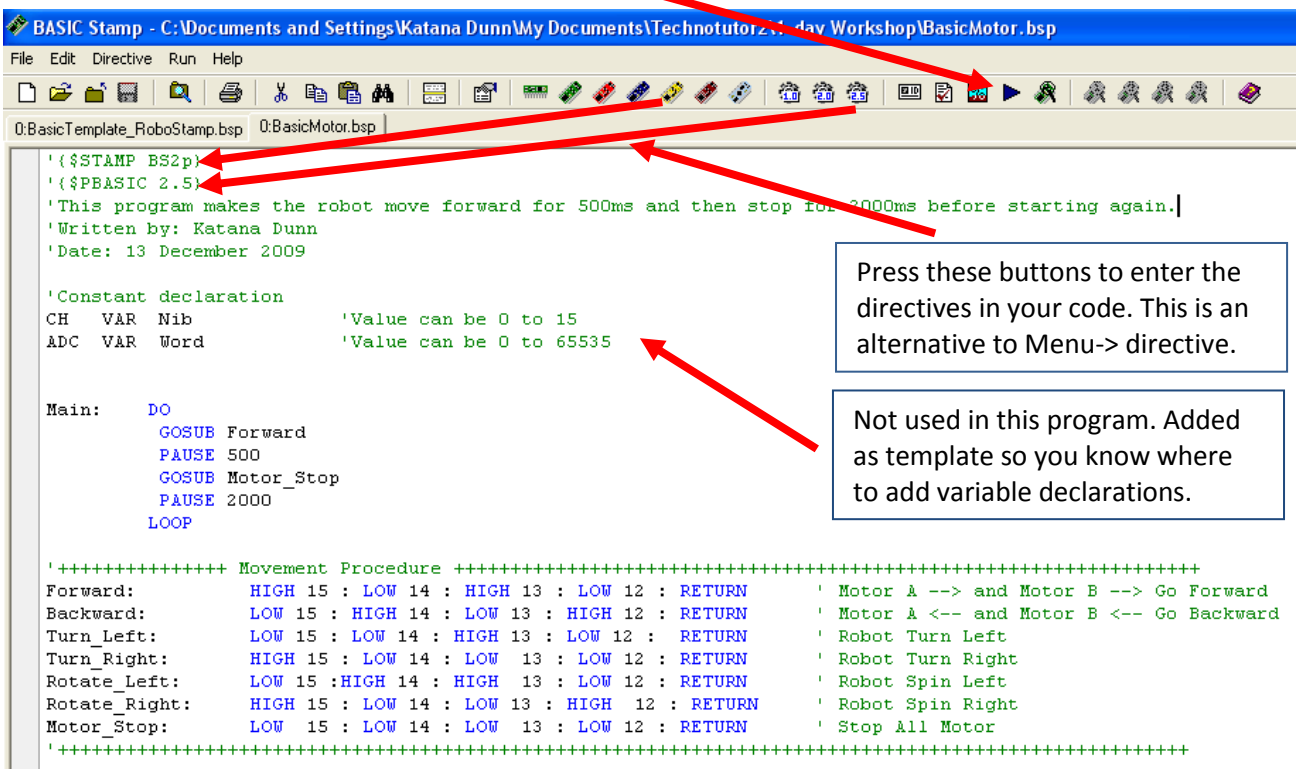
```
'{$STAMP BS2p}
'{$PBASIC 2.5}
'Program template
'Written by: Tom Riddle, Harry Potter
'Date: 13 December 2009

'Constant declaration
CH    VAR  Nib              'Value can be 0 to 15
ADC   VAR  Word             'Value can be 0 to 65535


Main:     DO
          ' Add your code here. Remember to write notes next to it.
          LOOP

'+++++++++++++++ Movement Procedure ++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Forward:          HIGH 15 : LOW 14 : HIGH 13 : LOW 12 : RETURN      ' Motor A --> and Motor B --> Go Forward
Backward:         LOW 15 : HIGH 14 : LOW 13 : HIGH 12 : RETURN      ' Motor A <-- and Motor B <-- Go Backward
Turn_Left:        LOW 15 : LOW 14 : HIGH 13 : LOW 12 :  RETURN      ' Robot Turn Left
Turn_Right:       HIGH 15 : LOW 14 : LOW  13 : LOW 12 : RETURN      ' Robot Turn Right
Rotate_Left:      LOW 15 :HIGH 14 : HIGH  13 : LOW 12 : RETURN      ' Robot Spin Left
Rotate_Right:     HIGH 15 : LOW 14 : LOW 13 : HIGH  12 : RETURN     ' Robot Spin Right
Motor_Stop:       LOW  15 : LOW 14 : LOW  13 : LOW 12 : RETURN      ' Stop All Motor
'+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

Figure 2 . BasicTemplate_RoboStamp.bsp

To execute the code repetitively, create your code between DO and LOOP. Add the following code.

Code	Explanation
GOSUB Forward	Call the Forward Subroutine
PAUSE 500	Wait for 500ms (while still moving Forward)
GOSUB Motor_Stop	Turn motor off

Save the file and run the program



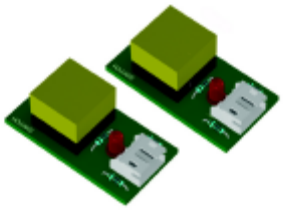
2.2 Challenges

This basic program can be used to create your own code. Program the robot to do the following:

a) Move around an obstacle
b) Move in a square shape
c) Move in a triangle shape

Because we are not using feedback to control the motors, accuracy will be an issue. This is a great way to get a feel for the motor movement and how the pause statement can effectively be used.

3 Collision sensors

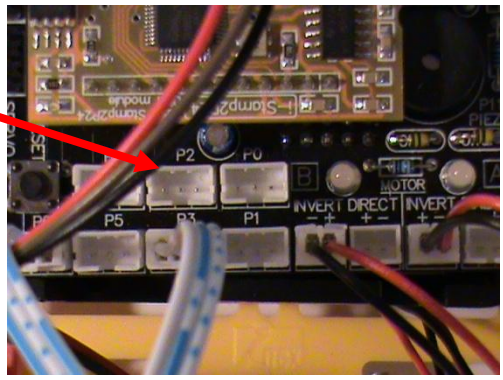


3.1 Operation

Switches are used as collision sensors.

It is very easy to read the input from the switches as shown below. The switch input is used to detect and avoid obstacles at logic '0'. It can also be used as an on/off switch.

The switches can be connected to any digital port from P0 to P6.



Example: If switch is connected to P1 read from IN1 in your program.

Use this basic code to read your values *(Use the basic template above and add your code*)**:**

```
DO
  IF IN1 = 0 THEN          'Check switch 1 before run program
    GOSUB Forward
    PAUSE 2000
    GOSUB Backward
    PAUSE 1000
    GOSUB Motor_Off
  ENDIF
LOOP                       'Loop program
```

```
'{$STAMP BS2p}
'{$PBASIC 2.5}
'This program waits for the switch on P1 to be pressed. It then go forward for 2sec, backward for 1sec and stop for two sec.
'Written by: Katana Dunn
'Date: 13 December 2009

'Constant declaration
CH   VAR  Nib              'Value can be 0 to 15
ADC  VAR  Word             'Value can be 0 to 65535


Main:    DO
           IF IN1 = 0 THEN           'Check switch before run program. If IN1 = 0(if switch connected to P1 is pressed)
              GOSUB Forward                                          'do this.
              PAUSE 2000
              GOSUB Backward
              PAUSE 1000
              GOSUB Motor_Stop
              PAUSE 1000
          ENDIF
         LOOP    'Loop program

'+++++++++++++++ Movement Procedure ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Forward:          HIGH 15 : LOW 14 : HIGH 13 : LOW 12 : RETURN      ' Motor A --> and Motor B --> Go Forward
Backward:         LOW 15 : HIGH 14 : LOW 13 : HIGH 12 : RETURN      ' Motor A <-- and Motor B <-- Go Backward
Turn_Left:        LOW 15 : LOW 14 : HIGH 13 : LOW 12 :  RETURN      ' Robot Turn Left
Turn_Right:       HIGH 15 : LOW 14 : LOW  13 : LOW 12 : RETURN      ' Robot Turn Right
Rotate_Left:      LOW 15 :HIGH 14 : HIGH  13 : LOW 12 : RETURN      ' Robot Spin Left
Rotate_Right:     HIGH 15 : LOW 14 : LOW 13 : HIGH  12 : RETURN     ' Robot Spin Right
Motor_Stop:       LOW  15 : LOW 14 : LOW  13 : LOW 12 : RETURN      ' Stop All Motor
'++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

3.2 Challenges:

Check the PBasic Syntax guide which is part of the BASIC Stamp editor for more information.

Try all the programs yourself before looking at the answers.

1. Create a program that makes the robot move only after a switch is pressed (start switch). It then moves around an obstacle until returning to the starting point.

```
Init: IF IN1 = 0 THEN          'Check switch before run program
        PAUSE 500
        GOTO Main
      ENDIF
      GOTO Init                ' Loop test again
```

2. Create a program that reacts when one of the switches is pressed. It has to react in the following way:
   (Hint: Keep the start switch to ensure robot only starts moving when you want it to.)

   a. If the left switch is pressed, move backwards for 500ms, turn right and continue.

```
Main:    DO
           GOSUB Forward
           IF IN1 = 0  THEN           'Left switch pressed
              GOSUB Backward
              PAUSE 500
              GOSUB Turn_Right
              PAUSE 500
           ENDIF
```

   b. If the right switch is pressed, move backwards for 500ms, turn left and continue.
   (Add to previous code)

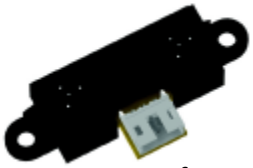
```
Main:    DO
           GOSUB Forward
           IF IN1 = 0  THEN           'Left switch pressed
              GOSUB Backward
              PAUSE 500
              GOSUB Turn_Right
              PAUSE 500
           ENDIF
           IF IN2 = 0  THEN           'Right switch pressed
              GOSUB Backward
              PAUSE 500
              GOSUB Turn_Left
              PAUSE 500
           ENDIF
         LOOP    'Loop program
```

3. Create a program that waits for two switches to be pressed at the same time before starting. *(Add your code here):*
(Hint: if IN1=0 AND IN2=0 THEN)

Answer:

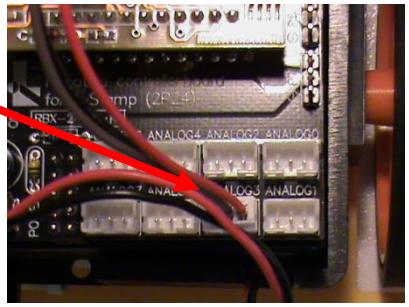
```
DO
  IF IN1 = 0  AND IN2 = 0 THEN          'Check both switches before run program. If IN1 = 0 as
     GOSUB Forward                      'well as IN2 = 0(If switch connected TO P1 and P2 is pressed) c
     PAUSE 2000
     GOSUB Motor_Stop
     PAUSE 100
     GOSUB Backward
     PAUSE 1000
     GOSUB Motor_Stop
     PAUSE 1000
  ENDIF
LOOP    'Loop program
```

4 Distance sensors



4.1 Operation

The Infrared detector module, found in the Robostamp kit, can measure distances ranging from 4 to 30cm using infrared light. This means that the robot can avoid obstacles without making contact. The sensor cable is connected to the analog inputs on the controller board. Example: If sensor connected to Analog 3, read from channel 3 in your RD_ADC.



We use code, supplied with the Basic StampEditor code, to read the distance sensor values:

```
'Basic program to read the values from the distance sensor and display in the debug window.

'Variable decarations
ADC     VAR     Word
R       VAR     Word
I       VAR     Byte
X       VAR     Word

'Main program
        PAUSE 1000
        HIGH 10                         ' Initialise ADC
        DO
          X = 0
          FOR I = 1 TO 5                ' Get Mean Off ADC 5 Time:Avaraging
              GOSUB RD_ADC
              X = (ADC+X)
          NEXT
         X = X/5
         R = (2914 /(X+5))-1            ' Convert Voltage to Range (CM)
         ' Show Range On Debug Terminal
         DEBUG "ADC = ",DEC X,TAB, "Range = ",DEC R ,"  CM",  CR
         PAUSE 500
        LOOP                            ' Read Again

RD_ADC:  LOW 10: PAUSE 1: HIGH 10       ' Send Acknowledge
        SEROUT 10,240,[3]               ' Send Select Chip, analog 3
        SERIN 10,240,250,Error,[ADC.BYTE0,ADC.BYTE1]    ' Read ADC
        RETURN
Error:  DEBUG "Error Reading",CR
        RETURN
```

Channel. This value will be: 1 for analog channel 1, 2 for analog channel 2 etc.

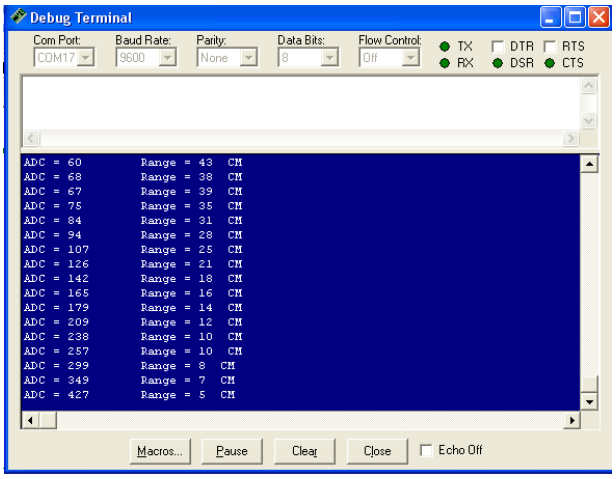
This sub function reads the ADC value for the sensor from the specified channel.

Figure 3. Use the ReadDistanceSensors.bsp to read the sensor values

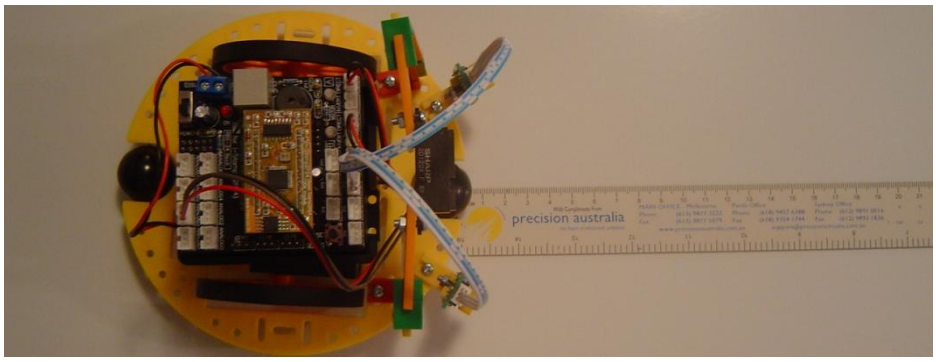
To make sure we understand the calibration of these sensors we measure the values the sensor measures for obstacles different distances from it.

Follow these steps.

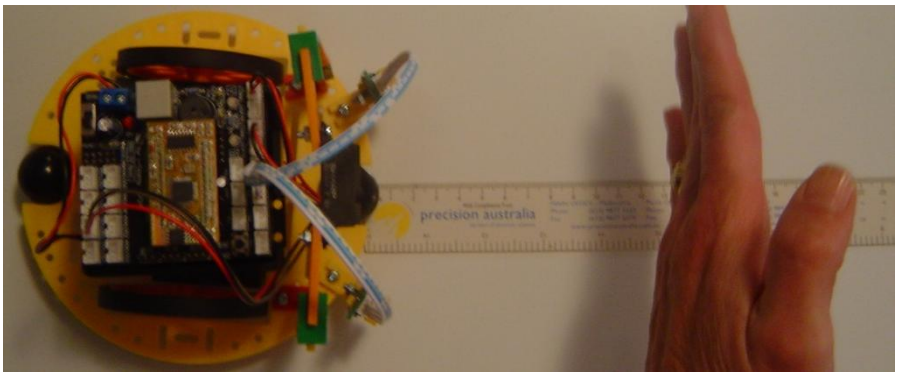
1. Run the program *ReadDistanceSensors.bsp* and keep the programming cable in. The debug window will appear. Observe the ADC values and the calculated range values.



2. Use a ruler in front of your robot, as shown.



3. Put your hand (or a book) in front and move it closer and further away from the distance



sensor.

4. **Complete the following table.**
   The first column is the value read from the ruler. Use values from 1cm to 40cm away (Put your hand or another object in the position as shown) and read the ADC and range value from the debug window.

Object Distance(cm)	ADC Value	Range

Object Distance(cm)	ADC Value	Range

The values can be used to make your code clever. You can use it to determine when the robot reads a certain value that it will do something. Use the same if statement that you used in the switch challenge.

4.2 Challenges:

1. Create code so that the robot turns without touching a wall. Keep your start switch from before.

```
'This program stays between four walls using the distance sensor
'If the sensor "sees" a wall, it turns
'Written by: Katana Dunn
'Date: 17 December 2009

'Variable and constant declaration
CH   CON  3              'Value can be 0 to 15. Use 3 for analog 3
ADC  VAR  Word           'Value can be 0 to 65535

Init: IF IN1 = 0 THEN         'Check switch before run program
        PAUSE 500
        GOTO Main
        ENDIF
      GOTO Init               ' Loop test again
Main: DO
        GOSUB Forward
        GOSUB RD_ADC
        IF ADC > 300  THEN      'If ADC more than 300 (8 cm in my case) do this:
           GOSUB Backward
           PAUSE 500
           GOSUB Turn_Right
           PAUSE 500
        ENDIF
      LOOP    'Loop program

'+++++ Analog to Digital Converter Procedure (Analog inputs only) ++++++++++
RD_ADC:  LOW 10: PAUSE 2: HIGH 10                          ' Send Acknowledge
         SEROUT 10,240,[CH]                                ' Send Select Chip
         SERIN 10,240,250,Error,[ADC.BYTE0,ADC.BYTE1]      ' Read ADC, store in variable ADC
         RETURN
Error:   DEBUG "Error Reading",CR
         RETURN

'+++++++++++++++ Movement Procedure ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
Forward:          HIGH 15 : LOW 14 : HIGH 13 : LOW 12 : RETURN      ' Motor A --> and Motor B --> Go Forward
Backward:         LOW 15 : HIGH 14 : LOW 13 : HIGH 12 : RETURN      ' Motor A <-- and Motor B <-- Go Backward
Turn_Left:        LOW 15 : LOW 14 : HIGH 13 : LOW 12 :  RETURN      ' Robot Turn Left
Turn_Right:       HIGH 15 : LOW 14 : LOW  13 : LOW 12 : RETURN      ' Robot Turn Right
Rotate_Left:      LOW 15 :HIGH 14 : HIGH  13 : LOW 12 : RETURN      ' Robot Spin Left
Rotate_Right:     HIGH 15 : LOW 14 : LOW 13 : HIGH  12 : RETURN     ' Robot Spin Right
Motor_Stop:       LOW  15 : LOW 14 : LOW  13 : LOW 12 : RETURN      ' Stop All Motor
'+++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
```

2. Modify the code to make the robot spin/dance when it detects an object. Be creative.