Robotics with the XBC Controller

Session 2

Instructor: David Culp

Email: culpd@cfbisd.edu

Learning Goals

- The student will learn the basics of motor control with the XBC, digital sensors, if-then and while loops, Boolean expressions and be able to combine these elements into a mobile robot that reacts to its environment.

Basic motor control

- The most basic motor control function in IC is the “motor” function.
  - Defined as motor(<*motor_#*>, <*speed*>)
- motor_# = motor port 0-3
- speed = -100 to 100

Example of use

- **To move forward for 3 seconds:**
- Type in, save, and run the following program. Make certain the left motor is plugged into port #0 and the right motor is plugged into port #2.

```
void main()
{
    motor(0,100);
    motor(2,100);
    sleep(3.0);
    ao();
}
```

My robot doesn’t go straight!

- If your robot does not go **forward** your motor wires are **likely** plugged in backwards.
- If your robot goes backwards:
  - Both motor wires are plugged in backwards.
- If your robot turns left:
  - The left motor wire is plugged in backwards.
- If your robot turns right:
  - The right motor wire is plugged in backwards.
- **If you robot veers to the right or left, one motor is weak, or an axle is pinched.**

What is happening…

- void main()
  - Remember, all C programs start at the main function.
- {
  - Opens the block of **statements** for “main” function.
- motor(0,100);
  - Turn on motor port 0 at full power **[100%]**
  - Notice the ending ;
- motor(2,100);
  - Turn on motor port 2 at full power
  - Notice the ending ;

What is happening continued…

- sleep(3.0);
  - Pause for 3 seconds while motors continue to rotate.
  - Notice the ending ;
- ao();
  - Turn off all motors
- }
  - Close the main function **[end block of statements]**
- **Note: All statements end with ';'**

Turning

- In order to turn we move one motor backwards and the other forwards.
- Turn the motor backwards in the direction you want to turn. i.e. if you want to turn left run the left motor backwards.

Example of turning

```
void main()
{
    motor(0,100);
    motor(2,100);
    sleep(3.0);

    motor(0,-100);
    motor(2,100);
    sleep(1.5);
    ao();
}
```

Your turn…

- Write a program that will cause your robot to do the following:
  - Move forward for 2 seconds
  - Turn left for 0.5 seconds
  - Move backwards for 2 seconds
  - Turn right for 0.75 seconds

```
void main()
{
   motor(0,100);
   motor(2,100);
   sleep(2.0);

   motor(0,-100);
   motor(2,100);
   sleep(0.5);

   motor(0,-100);
   motor(2,-100);
   sleep(2.0);

   motor(0,100);
   motor(2,-100);
   sleep(0.75);

   ao();
}
```

Other motor control functions

- ao();
  - Turn all motors off
- fd(<*motor_#*>);
  - Turn on motor_# in a “forward” direction **at full power.**
- bk(<*motor_#*>);
  - Turn on motor_# in a “backwards” direction **at full power.**
- off(<*motor_#*>);
  - Turn off motor_#

Boolean expressions

- Boolean expressions evaluate to either TRUE or FALSE.
  - 2<5 = TRUE
  - 3>5 = FALSE
- 0 = FALSE
- 1 = TRUE
- All expressions with a relational operator (i.e <) are boolean expresions.
- AND, OR, NOR, XOR, NOT are other boolean operators.
- Also known as relational, conditional, or comparison expressions

Boolean expressions continued.

- < less than
- <= less than or equal
- > greater than
- >= greater than or equal
- == equal
- != not equal
- && and
- || or
- ! not

Explanation of AND, OR and NOT

- In an AND expression **BOTH** statements **MUST** be true.
  - (2<3) && (17<30) = TRUE
  - (4>2) && (14<10) = FALSE
- In an OR expression **EITHER** statement can be true.
  - (2<3) || (17<30) = TRUE
  - (4>2) || (14<10) = TRUE
  - (4<2) || (14<10) = FALSE
- NOT is a unary operator which **negates or reverses** the current statement.
  - !0 = TRUE
  - !1 = FALSE
  - !(2<3) = FALSE

Advanced AND explanation

- The expression is evaluated 'Left to Right`. **If any part of the expression returns ZERO the evaluation ends**.

```
k=0;
i=3;
j=2;
if ( i-i && j++) k=1
```

What will j and k equal???

Advanced OR explanation

- OR also evaluates 'Left to Right' **and will stop when an expression returns true.**

```
k=0;
i=3;
j=2;
if ( i+i || j++) k=1
```

What will j and k equal?

If-then statements

- if (<*expression*>)
  <*statement-1*>
  else <*statement-2*>
- (<*expression*>)
  - A Boolean or conditional expression
- <*statement-1*>
  - program statements to execute if (<*expression*>) evaluates to TRUE
- else <*statement-2*>
  - Optional statements to execute if (<*expression*>) evaluates to FALSE.

Pseudo code example

```
if ( test for something)
{
  Do this if true…
}

If not true jump to here…..
```

While loops

- while (<*expression*>)
  <*statement*>
- (<*expression*>)
  - A boolean expression to test
- <*statement*>
  - C program statements to execute if (<*expression*>) evaluates to TRUE
- Multiple statements can be contained in braces { …}

Pseudo code while example

```
while( test for something)
{
  do this……
}
```

When while test = FALSE jump to here…..

An example program

```
void main()
{
    while(a_button() == 0)
      {// Open while loop braces

    }// Close while loop brace

    printf("A button pressed!\n");
    sleep(0.5);
    printf(“Program End”);
}
```

What are those weird // things?

- The // denotes comments in code.
- Comments are NOT executed or downloaded to the XBC.
- Comments are used to make the code more readable.
- Use comments liberally throughout your code.

Comments continued

- Block comments are used to comment large sections.
- /* opens a block comment
- */ closes a block comment
- Example:

```
/* This comment
Takes up more than one line
*/
```

Explanation

- void main()
  - Start the “main” function
- {
  - Open brace for the main function
- while(a_button() == 0)

{// Open while loop braces

}// Close while loop brace

  - Check the status of the “a” button
  - If it is NOT pressed then loop back up and check again.
  - Could also be written as while(!a_button())

More explanation

- a_button() checks the status of the a button on the game boy.
  - Returns a 1 if pressed
  - Returns a 0 if not pressed.
- printf("A button pressed!\n");
  - Print “A button pressed” if the while loop tests as FALSE.
- sleep(0.5);
  - A brief pause
- printf(“Program End”);
  - Tell reader that the program stopped.
- }
  - Close the main function

Another example

```
void main()
{
    while(1)
      {// Open while loop braces
        if (a_button() == 1)
          { // open if statement brace
            printf("A button pressed!\n");
            sleep(0.5);
        }// close if statement brace
    }// Close while loop brace
}
```

Explanation…

- while(1)
  {// Open while loop braces
  - 1 is ALWAYS TRUE therefore this while loop will never exit.
- if (a_button() == 1)

```
  { // open if statement brace
    printf("A button pressed!\n");
    sleep(0.5);
}// close if statement brace
```

- Test the "a" button. If it is pressed then execute the statements between the braces { }
- }// Close while loop brace
  - Return to the top of the while loop
- } - close the main function

An assignment

- Write a program that will do the following:
  - Your robots wheels move in a reverse direction if the “a” button is pressed.
  - Otherwise (else) your robots wheels move in a forward direction.

Possible solution

```
void main()
{
    while(1)
      {

        if(a_button())
          {
            bk(0);
            bk(1);
        }
        else
          {
            fd(0);
            fd(1);
        }
    }
}
```

Digital sensors

- Digital sensors have only TWO possible states:
  - On or off
  - 1 or 0
- Touch sensor the most common example. [The A button is a *built in* touch sensor.]

Reading digital sensors

- digital(<*port#*>);
  - Port# = ports 8-15
  - Returns a 0 or a 1

An example of using digital() **to print its state**

```
void main() // assumes a touch sensor attached to port #8
{
    while(1)
      {
        display_clear();
        if(digital(8))
          {
            printf("Digtal port 8 = 1");
        }
        else
          {
            printf("Digtal port 8 = 0");
        }
        sleep(.25);
    }
}
```

Preparing bumper-bot

- V1 kits – Plug the left switch into port 8 and the right switch into port 9.
- All others - Plug the front bumper into port 8 and the rear bumper into port 9.

V1 Kit Project

- Write a program that will cause your robot to roam around the room and react to its environment with the front touch sensors.
- If the left touch sensor is triggered the robot backs up and then turns right and continues.
- If the right touch sensor is triggered the robot backs up and then turns left and continues.

Project - All other kits

- Cause “bumper-bot” to play ping-pong
- If the front bumper is pressed the robot goes in reverse.
- If the rear bumper is pressed the robot goes forward.

Normally open Vs. Normally closed switches

- Be aware that the switches on the V1 kit are mounted in such a way that they are NORMALLY CLOSED (NC).
  - They should return a 1 or TRUE when the robot has NOT encountered an obstacle.
  - To test if it HAS hit an object:
    - if(digital(8) == 0)

Normally open Vs. Normally closed switches continued

- All other kits are NORMALLY OPEN (NO)
  - They should return a 0 or FALSE when the robot has NOT encountered an obstacle.
  - To test if it HAS hit an object:
    - if(digital(8) == 1)

Using comments to set robot parameters.

- Remember, comments do nothing other than document the code and robot.
- Comments are especially useful in documenting the configuration of the robot.

```
/*
Program Name: bumperbot.ic
Date Created: August 9th, 2006
Author: David Culp
email: culpd@cfbisd.edu

Purpose:

This program will cause a differential drive robot with a touch
switch in the front and the back to "ping pong“: When the front
switch is touched the robot begins moving backwards.  When the
rear switch is triggered the robot begins moving forwards.

Robot configuration:

Left DC motor - port 2
Right DC motor - port 0
Front touch switch - digital port 8 NORMALLY OPEN SWITCH
Rear touch switch -  digital port 9 NORMALLY OPEN SWITCH

*/

void main()
{
```

```
void main()
{
    fd(0); //start robot going forward
    fd(2);

    while(1) // do this forever
      {
        if(digital(8) == 1) // if the front bumper is pressed
          {
            bk(0);// set both motors in reverse
            bk(2);
        }//end if
        if(digital(9) == 1)// if back bumper is pressed
          {
            fd(0); //set both motors going forward
            fd(2);
        }//end if
    }//end while
} // end main
```