

While Loops

Slide	Topic
4	<u>Program Flow Control With Loops</u>
5-6	<u>While Loops</u>
7	<u>While Statement</u>
8	<u>Built-in Digital Sensor</u>
9	<u>While Loop on Push Button</u>
10	<u>While and Boolean Operators</u>
11	<u>Boolean Operators Cheat Sheet</u>
12-14	<u>Drive Until Sensor is Pressed</u>

Slide	Topic
15	<u>Changing the Condition</u>
16-17	<u>Square Up Using Bump Sensors</u>
18	<u>Large Touch Sensors Mounted on Back of Robot</u>
19	<u>Square Up Method #2</u>
20	<u>Square Up Method #2 Solution</u>

- What if we want to repeat the same “item/action” over and over (and over and over)?
- For example, checking to see if a touch sensor has been pressed.
- We can do this using a loop, which controls the flow of the program by repeating a block of code.

While Loops

We accomplish this loop with a **while** statement.

while statements keep a block of code running (repeating/looping) so that sensor values can be continually checked and a decision made.

The while statement checks to see if something is true or false (via Boolean operators).

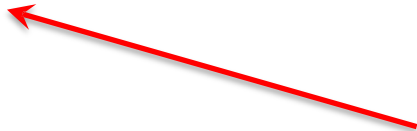
```
while (condition)
```

```
{
```

```
    Code to execute while  
    the condition is true
```

```
}
```

Notice there is
no terminating
semicolon after
the while
statement



While Loops

The `while` loop checks to see if a **Boolean test** is **true** or **false**...

- If the **test** is **true**, then the `while` loop **continues** to execute the **block of code** that *immediately* follows it.
- If the **test** is **false**, then the `while` loop **finishes**, and the line of code *after* the **block of code** is executed.

```
int main()
{
    // Code before loop
    while (Boolean test) ← Block Header
    {                               (no
        // Code to repeat ...       semicolon!)
    }
    // Code after loop

    return 0;
}
```

Begin →

End →

While Statement

Type of sensor:
analog, digital

Port number:
analog (0-5),
digital (0-9)

Notice no
terminating
statement

while (**digital** (**port#**) **==** **0**)

{

motor (**0,75**);

motor (**3,75**);

}

Code to execute while the condition is true

Boolean logic;

> Greater than

>= Greater than or
equal

< Less than

<= Less than or
equal

== Equal to

!= Not equal to

The Wombat has a built-in physical button on the right side of the controller

push_button()

The Wombat also has built-in touch screen buttons on the bottom of the robot screen (a, b, c and more if needed)

a_button() **b_button()** **c_button()**

- returns a value of 1 if the button is currently pressed
- returns a value of 0 if the button is not being pressed at that time

While Loop on Push Button



Example:

```
1 #include <kipr/wombat.h>
2
3 int main()
4 {
5     // Has push button been touched?
6     while(push_button() == 0)
7     {
8         printf("Press the Push Button!\n");
9     }
10
11     printf("Ahh! Something touched my Push
12 Button!\n");
13     return 0;
}
```

push_button

The **Boolean test** in a **while** loop is asking a question:

Is this statement **true** or **false**?

- The **Boolean test** (question) often compares two values to one another using a **Boolean operator**, such as:

== Equal to (NOTE: two equal signs, not one which is an assignment!)

!= Not equal to

< Less than

> Greater than

≤ Less than or equal to

≥ Greater than or equal to

Boolean Operators Cheat Sheet

Boolean	English Question	True Example	False Example
A == B	Is A equal to B?	5 == 5	5 == 4
A != B	Is A not equal to B?	5 != 4	5 != 5
A < B	Is A less than B?	4 < 5	5 < 4
A > B	Is A greater than B?	5 > 4	4 > 5
A <= B	Is A less than or equal to B?	4 <= 5 5 <= 5	6 <= 5
A >= B	Is A greater than or equal to B?	5 >= 4 5 >= 5	5 >= 6

Drive Until Sensor is Pressed

Description: Write a program that drives the DemoBot forward until a touch sensor is pressed, and then stops.

Analysis: What is the program supposed to do?

Pseudocode

1. Drive forward.
2. Loop: Is not touched?
3. Stop motors.
4. End the program.

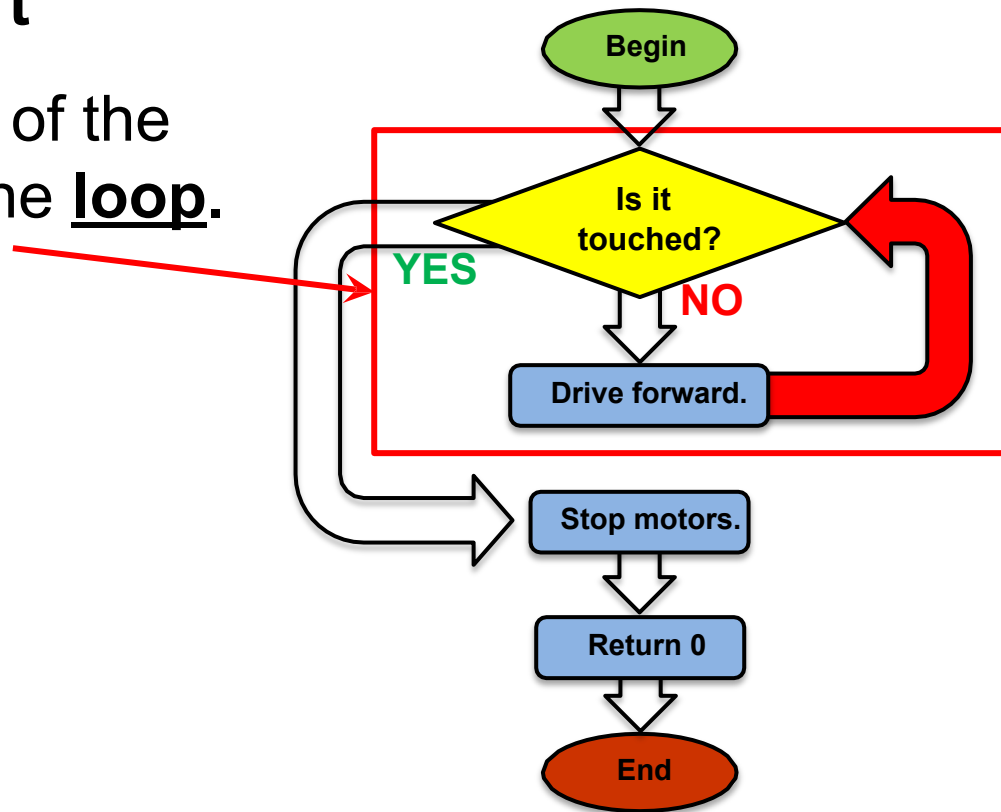
Comments

- // 1. Drive forward.
// 2. Loop: Is not touched?
// 3. Stop motors.
// 4. End the program.

Analysis:

Flowchart

This part of the code is the loop.



Drive Until Sensor is Pressed

Solution:

Pseudocode

1. Loop: Is it Touched?
 Drive Forward
2. Stop Motors
3. End the Program

Source Code

```
1 #include <kipr/wombat.h>
2
3 int main()
4 {
5     printf("Drive until bump\n");
6     while (digital(0) == 0)
7     {
8         motor(0, 75);
9         motor(3, 75);
10    }
11    ao();
12
13    return 0;
14 }
15
```

Changing the Condition

1. Change the expected (test condition) value from 0 to 1
2. Objective:
Predict/describe what you think the robot will do
3. Run the program

Source Code

```
1 #include <kipr/wombat.h>
2
3 int main()
4 {
5     printf("Drive until bump\n");
6     while (digital(0) == 1)
7     {
8         motor(0, 75);
9         motor(3, 75);
10    }
11    ao();
12
13    return 0;
14 }
15
```

- Sometimes, it is useful to have a robot “square up” to then drive straight 90 degrees from a “wall”.
- This can be done in a number of ways. One common one is to use two bump (digital) sensors mounted at two “corners” of the back of the robot.
- What follows are two possible “algorithms/methods”

Square Up Using Bump Sensors

Description: Use the pair of touch sensors on the back of the DemoBot to square up on a wall or PVC structure.

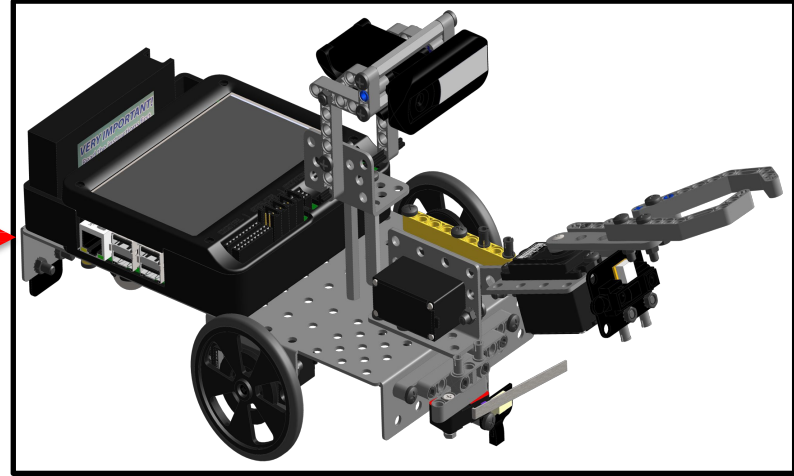
Background diagnostic work: You will need to plug your digital button sensors into digital ports. A good strategy might be to use the same port number as your motor port. E.g. right motor plugged into port 0, right button sensor plugged into digital 0.

Key Coding Concepts: Each of the digital sensors will need to be “married” to a wheel in code. One way to handle this is to nest two (if-else pairs) inside of a while loop. Essentially, one of these pairs will control the left wheel and one will control the right wheel.

Method #1: The robot will move backward until it senses either back bump sensor is pushed. Upon a sensor being pushed, its corresponding wheel will freeze, the other wheel will continue to move backward until its sensor is pushed. At the point, the robot will exit the loop.

Bonus: Upon completing a square up, your robot will move forward 1000 ticks.

Large Touch Sensors Mounted on Back of Robot



Square Up Method #2

Description: Write a program that drives backward to orient your robot perpendicular to a “PVC wall”.

Analysis: What is the program supposed to do?

Pseudocode

1. Loop: Both sensors touched?
2. If **only** right sensor touched?
3. Else If **only** left sensor touched?
4. Else drive backward
5. End the program when both touched

Comments

```
// 1. Loop: Are both sensors pressed?  
// 2. If right sensor is touched turn CCW  
// 3. Else-If left sensor is touched turn  
CW  
// 4. Else drive backward  
// 5. End the program.  
  
// note: CCW means counter clockwise; CW  
means clockwise
```

Square Up Method #2 Solution

Source Code

```
1 #include <kipr/wombat.h>
2
3 int main()
4 {
5     printf("Back Up to Square Up :-)\n");
6     while ((digital(3) == 0) || (digital(0) == 0)) // Left or Right is not pressed
7     {
8         if ((digital(3) == 0) && (digital(0) == 1)) //Right is pressed (not Left)
9         {
10            motor(3, -90);
11            motor(0, 10); // turn CCW backwards with right motor at zero
12        }
13        else if ((digital(3) == 1) && (digital(0) == 0))
14        {
15            motor(3, 10);
16            motor(0, -90); // turn CW backwards with left motor at zero
17        }
18        else
19        {
20            motor(3, -75);
21            motor(0, -75); // just keep going backwards
22        }
23    }
24    ao();
25    return 0;
26 }
```

Assumes that motor 0 and digital 0 are on the right side and motor 3 and digital 3 are on the left side.

Square Up Method #2 Solution - Reflectance

Source Code

```
1 #include <kipr/wombat.h>
2
3 int main()
4 {
5     printf("Back Up to Square Up :-)\n");
6     while ((analog(0) < 1300) || (analog(5) < 1300)) // Left or Right not on black
7     {
8         if ((analog(0) < 1300) && (analog(5) >= 1300)) //Right not on black
9         {
10            motor(3, -50);
11            motor(0, 50); // turn CCW
12            msleep(50);
13        }
14        else if ((analog(0) >= 1300) && (analog(5) < 1300)) //Left not on black
15        {
16            motor(3, 50);
17            motor(0, -50); // turn CW
18            msleep(50);
19        }
20        else
21        {
22            motor(3, 75);
23            motor(0, 75); // just keep going forwards
24        }
25        motor(0, 0);
26        motor(3, 0);
27        msleep(50);
28    }
29    return 0;
30 }
```

Assumes that motor 0 and digital 0 are on the right side and motor 3 and digital 5 are on the left side.