Intro to Sensors

- 4 types of sensors available in the standard NXT kit

Programming Sensors

- Sensors return a varying range of values
  - Touch Sensors return a value of 0 or 1
  - Light and Sound Sensors return a value between 0 and 100
  - Sonar Sensors return a value in cm, up to 255cm.
- One function in ROBOTC returns this value for you to use in your program
  - **SensorValue[*sensorName*];**
  - **SensorValue(*sensorName*);**
- Another function works in the same manner for encoder counts.
  - **nMotorEncoder[*motorName*];**

Touch Sensor

- Digital Sensor
  - Returns either 0 or 1
- Useful for…
  - Detecting touches
  - Acting as a limit switch
  - User interfaces to robot

While Loops

- A while loop is a structure within ROBOTC which allows a portion of code to be run over and over, as long as the specified Boolean condition remains “true”.

```
task main()
{
 while(nMotorEncoder[motorC]<360)
 {
   motor[motorC]=100;
   motor[motorB]=100;
 }
}
```

The condition is true as long as the rotation sensor detects less than 360 degrees of rotation.

While the condition is true, both motors will receive 100% power.

While Loops

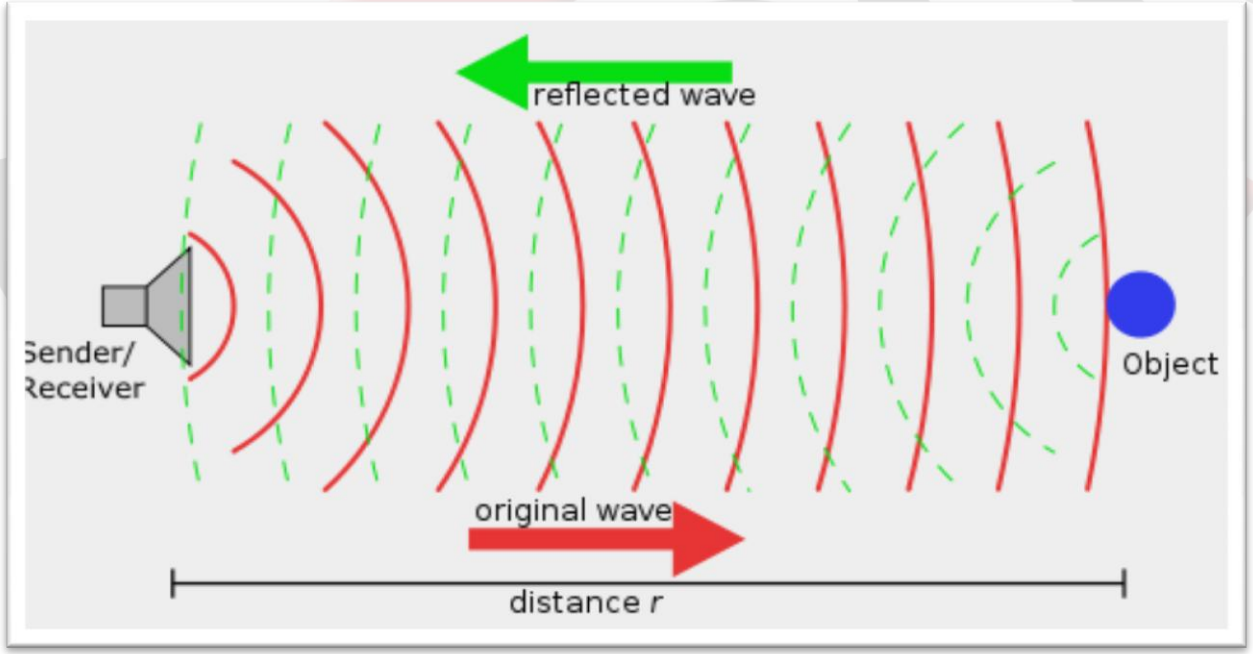
- Things to avoid with “while loops”
  - Having a condition that could never be true
    - Example: while(SensorValue[touch1] < 0)
  - Using a semicolon
    - Example: while(SensorValue[touch1] == 0);
    - This code will make an “idle” loop
      - i.e. a loop with no code
  - Not using curly braces
    - While the code will still compile, it will be very difficult to track what is in the loop and what isn’t.

Sonar Sensor

- I²C Sensor
  - Returns a value between 0 and 255
  - Value returned is number of cm
- Useful for…
  - Detecting flat objects
  - Measuring distances

Sonar Sensors

- A sonar sensor works by sending out an ultrasonic pulse and measure how much time it takes for the echo to be returned – it relies on the speed of sound as a reference point.

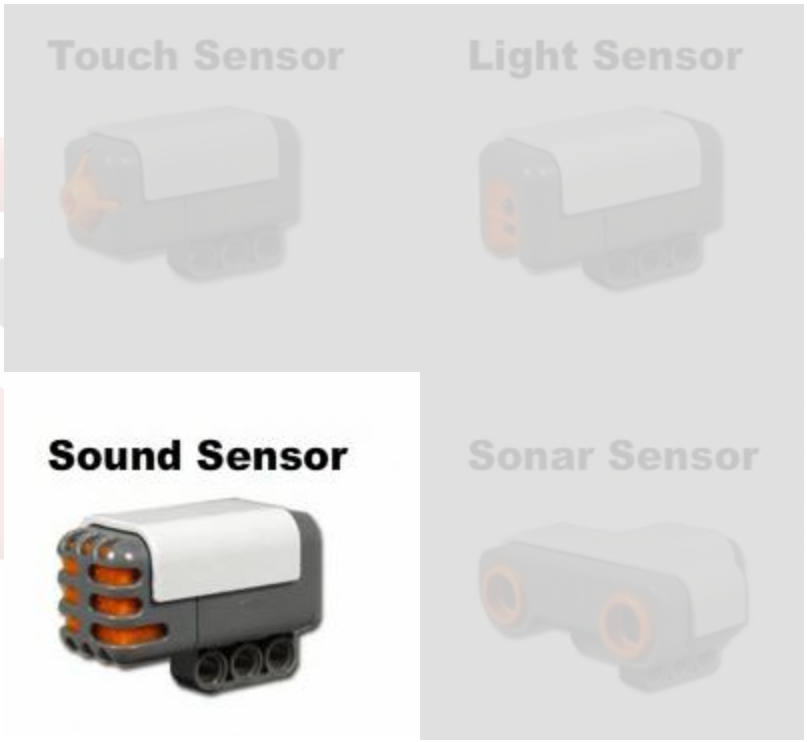


Sonar Sensors

- Sonar sensors are not very effective on round objects
  - The "echo" can't return back to the sonar sensor very well
- Multiple sonar sensors in the same area can cause "cross-talk"
  - They could interfere and produce random results
- How could you convert the cm to inches?
  - Divide by 2.54!
  - (SensorValue[sonar4] / 2.54)

Sound Sensor

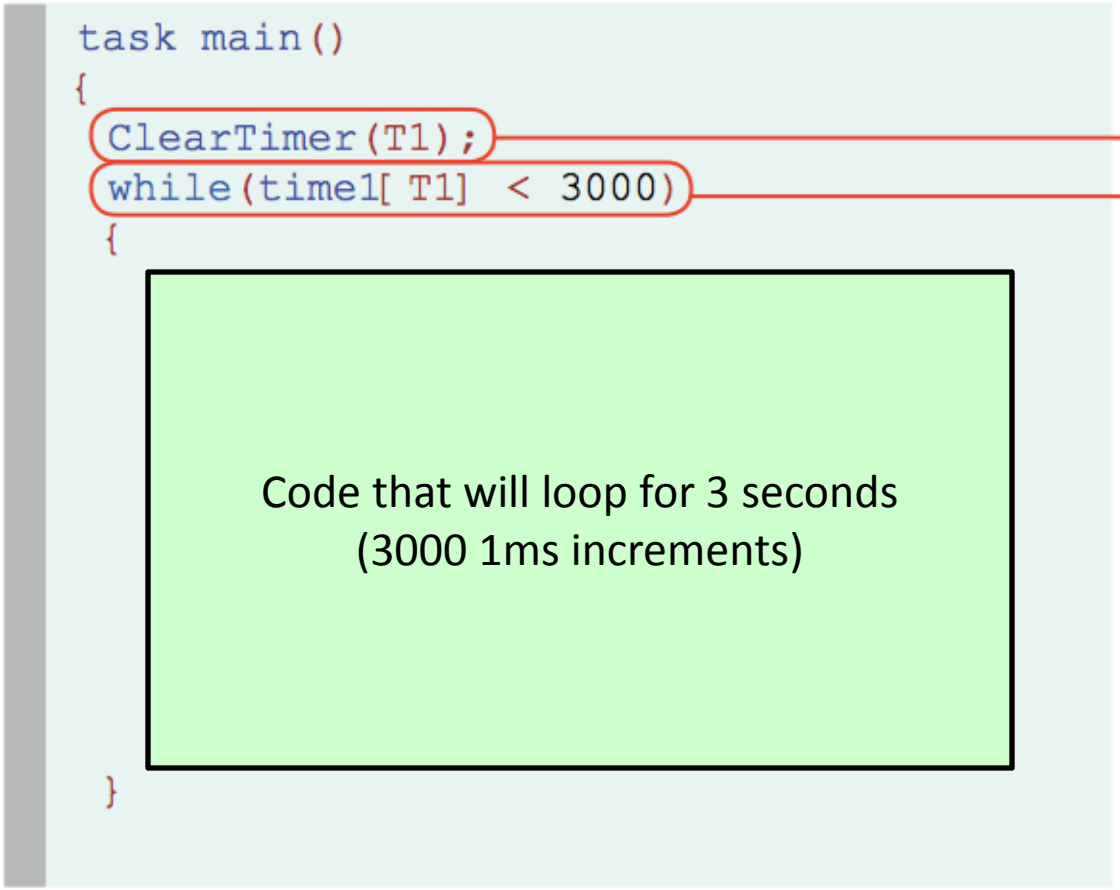
- Analog Sensor
  - Returns a value between 0 and 100
- Useful for…
  - Detecting Volume of Sounds
- Not really useful for TETRIX
  - The sound of the DC motors will drown out the Sound Sensor



Timers

- Timers are very useful for performing a more complex behavior for a certain period of time.
  - Wait statements (wait1Msec) do not let the robot execute commands while waiting period
- Timers allow you to track the amount of elapsed time while having other code run in your program.

Timers



Clear the Timer
Clearing the timer resets and starts the timer. You can choose to reset any of the timers, from T1 to T4.

Timer in the (condition)
This loop will run "while the timer's value is less than 3 seconds", i.e. **less than 3 seconds have passed since the reset.** The line tracking behavior inside the {body} will continue for 3

Robo Rmatter INCORPORATED

Carnegie Mellon Robotics Academy

Timers

- First, you must reset and start a timer by using the ClearTimer() command. Here's how the command is set up:
  - ClearTimer(Timer_number);
  - ROBOTC has 4 built in timers: T1, T2, T3, and T4.
- Then, you can retrieve the value of the timer by using…
  - time1[T1] – Returns the number of 1ms increments that have elapsed.
  - time10[T1] – Returns the number of 10ms increments that have elapsed.
  - time100[T1] – Returns the number of 10ms increments that have elapsed.

Light Sensor

- Analog Sensor
  - Returns a value between 0 and 100
- Useful for…
  - Detecting reflect light
  - Detecting ambient light
  - Detecting changes in surfaces (dark vs. light)
  - Following Lines

Light Sensors

- Thresholds are the most important thing!
  - Every environment that you will be in will cause a different threshold value
- Distance away from an object is the second most important thing!
  - If the light sensor is 1cm away from an object, the threshold will be very different if the light sensor becomes 3cm away.
- Light Sensors can be used without the red LED
  - Set your sensor type to “Light Inactive”
  - This will make the light sensor a passive light sensor, good for detecting room brightness.

If/Else Statements

- An if-else Statement is one way you allow a computer to make a decision.
  - With this command, the program will check the (condition) and then execute one of two pieces of code, depending on whether the (condition) is true or false.
- If/Else statements typically need a while loop as well!
  - Otherwise, the If/Else statement will only be checked once and the program will continue onwards.

If/Else Statements

- If the “while(true)” loop was missing, this code would only execute the “if” or “else” section once.

```
task main()
{
  while(true)
  {
    if(SensorValue[light2] < 45)
    {
      motor[motorB] = -50;
      motor[motorC] = 50;
    }
    else
    {
      motor[motorB] = 50;
      motor[motorC] = -50;
    }
  }
}
```

If/Else Statements

- If statements do not require an “else” statement.
  - If the “if” statement is false, it will just be skipped over.
- You can also make a multiple decision “if/else” statements.
  - Example on the right →

```
task main()
{
  int myValue = 50;

  if(myValue < 30)
  {
    //Scenario #1
  }
  else
  {
    if(myValue < 40)
    {
      //Scenario #2
    }
    else
    {
      if(myValue > 45)
      {
        //Scenario #3
      }
      else
      {
        //Scenario #4
      }
    }
  }
}
```

“else if” Shorthand

```
task main()
{
  int myValue = 50;

  if(myValue < 30)
  {
    //Scenario #1
  }
  else
  {
    if(myValue < 40)
    {
      //Scenario #2
    }
    else
    {
      if(myValue > 45)
      {
        //Scenario #3
      }
      else
      {
        //Scenario #4
      }
    }
  }
}
```

Make your code more readable by using the “else if(condition)” command

```
task main()
{
  int myValue = 50;

  if(myValue < 30)
  {
    //Scenario #1
  }
  else if(myValue < 40)
  {
    //Scenario #2
  }
  else if(myValue > 45)
  {
    //Scenario #3
  }
  else
  {
    //Scenario #4
  }
}
```

If/else Shorthand

- You can also have a single line “if” statement with out the need for curly braces “{“

```
task main()
{
  int myVariable = 34;

  if(myVariable > 40)
    wait1Msec(500);
  else
    wait1Msec(2000);
}
```

```
task main()
{
  int myVariable = 34;

  if(myVariable > 40) wait1Msec(500);
  else wait1Msec(2000);
}
```

For Loops

- “For Loops” are important for repeating a block of a code a certain number of times.
- For loops require a specific structure:

```
for (initial; condition; increment)
{
  body
}
```

For Loops

- For Loop Syntax:
  - Initial – The variable that will be used to count the number of iterations through the loop
    - Example: int i = 0;
  - Condition – The conditional statement to decide how many iterations to loop through
    - Example: i < 10;
  - Increment – The statement to modify the counter variable through each iteration of the loop.
    - Example: i++
    - No semicolon at the end of this increment portion

For Loops

```
for (initial; condition; increment)
{
  body
}
```

Drive and Turn – Loops 10 Times

```
for (int i = 0; i < 10; i++)
{
  motor[motorB] = 50;
  motor[motorC] = 50;
  wait1Msec(1000);
  motor[motorB] = -50;
  motor[motorC] = 50;
  wait1Msec(500);
}
```

For Loop Walkthough

1. Create the for loop initial variable
2. Check the condition of the for loop
   - If true, continue onward
   - If false, skip the “for” loop
3. Run code inside of “for” loop
4. Run the iteration code and increment “i” by one.

- Loop steps 2-4 until 2 returns false.

```
for (int i = 0; i < 10; i++)
{
  motor[motorB] = 50;
  motor[motorC] = 50;
  wait1Msec(1000);
  motor[motorB] = -50;
  motor[motorC] = 50;
  wait1Msec(500);
}
```

RoboMatter INCORPORATED

Carnegie Mellon Robotics Academy

Switch Case

- A “switch” case is used as a selection control mechanism to make a decision based on a variable/value.
- The idea is similar to a number of “if, else if, else if, else if, else” statements but in an easier to read fashion.
- You can only specify single values for each case – you can’t say…
  - “Case 50 to 100”
  - “Case Less than 30”

Switch Case Example

- Each case is specified with a value
  - Example: “case 5:”
- All of the code under that case is what will run if “myVar” is the value of that case.
  - In this example, the motors will turn on at 50% power
- The “break;” statement is required at the end of each case
  - Otherwise the cases will “fall” through and start running all of the code, regardless of the “case” heading.
  - In this example, the motors would turn on at 50% power, then 75% power and then 0% power.

```
int myVar = 12;

switch (myVar)
{
  case 5:
  motor[motorB] = 30;
  motor[motorC] = 30;
  break;

  case 12:
  motor[motorB] = 50;
  motor[motorC] = 50;
  break;

  case 8:
  motor[motorB] = 75;
  motor[motorC] = 75;
  break;

  default:
  motor[motorB] = 0;
  motor[motorC] = 0;
}
```

Switch Cases

- Switch cases do not have much application in Robotics, but they do in state machine architectures and application programming.
- A series of “else if” statements are usually more effective and clearer for new programmers to understand, especially with sensor values.
- Switch/case statements are a frequent source of bugs among even experienced programmers, given that, in practice, the "break" is almost always the desired path, but not the default behavior of the switch/case construct.

Loop Control

- Two control statements are available to you when using “while” and “for” loops:
  - **continue;** - skips any remaining code below the continue statement and proceeds with the next iteration in the loop.
  - **break;** - breaks out of the current looping structure and proceeds execution of the rest of the program.

Loop Control

- The continue statement will cause the robot to keep moving forward until the touch sensor it pressed.
- The break statement will end the infinite while loop if the touch sensor is pressed

```
for(int i = 0; i < 10; i++)
{
  motor[motorB] = 50;
  motor[motorC] = 50;
  wait1Msec(1000);

  if(SensorValue[touch1] == 0)
    continue;

  motor[motorB] = -50;
  motor[motorC] = 50;
  wait1Msec(500);
}
```

```
while(true)
{
  motor[motorB] = 50;
  motor[motorC] = 50;

  if(SensorValue[touch1] == 1)
    break;
}
```