Additional Technology Activities

NOAA's new ROV, the *Deep Disscoverer*. Its namesake was a highly accomplished retired NOAA vessel, the NOAA Ship *Discoverer*. The *Discoverer* was the largest vessel ever built by the U.S. for oceanographic research at 303'. It operated in the Arctic, Antarctic, Pacific and Atlantic basins. Its legacy is vast and wide and its data continue to be used today by scientists. The expectation for *Deep Discoverer* is that it will go places others have not, and provide a legacy of data that will provide value to the science and management communities for many years to come by doing what it does best—making discoveries. Image courtesy NOAA.

During a period of relative calm, the ROV pilot and pilot trainee recover the SuperPhantom ROV to the deck of the *R/V F.G. Walton Smith*.
http://oceanexplorer.noaa.gov/explorations/13pulleyridge/logs/august19/media/rov_recovery_1559_hires.jpg

Underwater Robots

This series of three activities introduces basic systems used in many underwater robots to gather information for ocean exploration. The activities in this series are:

1. Getting Control with Microcontrollers;
2. Making Things Happen with Servos; and
3. Exploring with Sensors.

The purpose of these activities is to introduce students and educators to materials, methods, and technologies that they can use for a wide variety of inquiries, activities, and projects that integrate skills in science, technology, engineering, and mathematics.

For additional information see the Underwater Robots section of *The Okeanos Explorer Education Materials Collection, Volume 2: How Do We Explore?* starting on page 121 (*http://oceanexplorer.noaa.gov/okeanos/edu/collection/media/hdwe-URintro.pdf*).

The control room of the *Okeanos Explorer* is as well coordinated as any live television studio. Image courtesy of NOAA *Okeanos Explorer* Program, 2013 Northeast U.S. Canyons Expedition.
http://oceanexplorer.noaa.gov/okeanos/explorations/ex1304/logs/aug9/media/controlrm-hires.jpg

Activity 1: Getting Control with Microcontrollers

This activity is the first in a series of three activities that introduce basic systems used in many underwater robots to gather information for ocean exploration.

Figure 1. A microcontroller

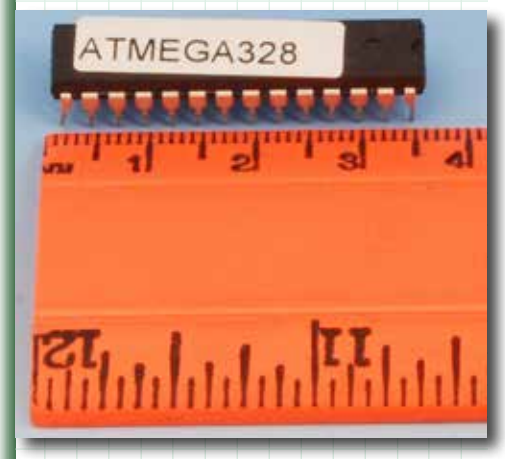


Figure 2. A microcontroller placed onto an Arduino board.

Figure 3. Lego® NXT brick

What is a robot? Most definitions involve the concept of a mechanical device performing human or near-human tasks, and/or behaving in a human-like manner. The key ideas are that a robot has a purpose, and mimics certain human or animal functions. The earliest robots were operated entirely by humans who were connected by a cable or wireless radio control system. Today, even the simplest robots contain an electronic device called a microcontroller. This device is not only for robots; microcontrollers are found in home appliances, automobiles, marine engines, televisions, media players, interactive games, toys and many other products. A microcontroller is a tiny computer that includes:

- A central processing unit (CPU) that receives input, processes data, and provides output;
- Memory that stores operating instructions, input data, and results of data processing;
- Input and output ports; and
- One or more timers.

All of these functions are contained in a plastic package (a "chip") that is about the size of a piece of macaroni (Figure 1). Microcontrollers need additional hardware to connect them to other devices, and this hardware is often placed on a circuit board along with the microcontroller (Figure 2).

Many different microcontrollers are used in robots. Two of the best-known and most popular are the Arduino board and the Lego® NXT brick (Figures 2 and 3). Some of their characteristics are compared in Table 1. Igoe (2011) reviews these along with six other microcontroller systems, and offers general guidance on how to choose an entry-level microcontroller. Both of these systems are easy to work with, and absolute beginners can get results in less than 30 minutes. This does not mean, however, that these systems are too simple for "serious" robotics projects. Arduino and NXT microcontrollers have been used in projects ranging from satellites to underwater remotely operated vehicles.

Summary

This activity is intended to familiarize students with beginning steps for using two popular types of microcontroller. Students are not expected to be proficient programmers following this activity; only to know how to follow instructions for creating programs and how to upload such programs to the microcontroller they are using.

Materials

For a Lego® NXT brick system:

- NXT brick microcontroller
- Lego® Color Sensor
- USB cable
- Computer with Lego® Mindstorms NXT software installed
- Engineering Design Journals, one for each student; educator may specify specific format (size, binding, etc.)
- Copy of *Introduction to Lego® NXT Programming Student Guide* (page 8), one copy for each student group

Table 1
Some Characteristics of Lego® NXT and Arduino Microcontrollers

	Lego®	Arduino
Cost	NXT brick alone – $145 Software – $80 Base Set – $280 (includes software; NXT brick; cables; 3 servos; light, touch, ultrasonic sensors; building parts)	Microprocessor Board alone – $30 Software – free download Starter Kit – $60 (includes USB cable, prototyping board, power supply adapter, light sensor, electronic components)
Inputs/Outputs	3 motor ports, 4 sensor ports, USB port	14 digital input/output ports, 6 analog input ports, 6 analog output ports, USB port, power supply jack
Programming Language	Mindstorms visual language based on LabVIEW; Robot C software available ($80)	C-based Arduino language; expandable using C/C++ language
Accessories	Compatible with Lego® construction kits; many projects documented in books and on the Internet; wide variety of accessories and curricula available	Wide variety of add-on kits available; many projects documented in books and on the Internet; open-source construction allows inexpensive electronic components and sensors to be used
Knowledge Base & Support	Many tutorials and examples on the Internet; FIRST Lego League is a team-based robotics competition for 9 – 16 year-olds	Many tutorials and examples on the Internet; world-wide community of users who share ideas and information
Comments	The graphical interface makes it easy to learn programming logic and Lego® construction methods are familiar to many students. The price is high for basic microcontroller projects, and the closed-construction obscures the actual components. Adapting this system to other components can be difficult.	Inexpensive, easily adaptable to many different projects. C/C++ language syntax can be confusing for beginners, but researchers at the MIT Media Lab have recently developed a graphical interface for Arduino programming that provides an alternative to the text-based C/C++ language (Baafi, 2011).

For an Arduino system:

- Arduino Board (many versions are available; the Duemilanove is one of the simplest and easiest to learn)
- Light emitting diode (LED) (e.g., Radio Shack 276-0017; exact specifications are not critical)
- USB cable
- Computer with Arduinos software installed (download from *www.arduino.cc/en/Main/Software*; installation instructions are linked from the same page)
- Engineering Design Journals, one for each student; educator may specify specific format (size, binding, etc.)
- Copy of *Introduction to Arduino Programming Student Guide* (page 12), one copy for each student group

Introduction

Microcontrollers are tiny computers that include:

- A central processing unit (CPU) that receives input, processes data, and provides output;
- Memory that stores operating instructions, input data, and results of data processing;
- Input and output ports; and
- One or more timers.

All of these functions are contained in a plastic package (a "chip") that is about the size of a piece of macaroni (Figure 1). Microcontrollers need additional hardware to connect them to others devices, and this hardware is often placed on a circuit board along with the microcontroller (Figure 2).

Microcontrollers are involved in many technological systems aboard the *Okeanos Explorer*, including

- communications equipment for telepresence;
- meterological sensors;
- motor control and navigation equipment for underwater robots;
- video imaging systems;
- CTD data processing; and
- multibeam sonar mapping.

Microcontrollers are also common in many home appliances, automobiles, marine engines, televisions, media players, interactive games, toys and many other familiar products.

In this activity, students program a microcontroller to carry out a set of instructions. Other activities in this series use microcontrollers to operate sensors that collect environmental information, and to operate servos that provide motion outputs. The objective of this activity is to program a microcontroller so that it will flash a light in a specific pattern. This will provide experience with the programming process, as well as with connecting components in a microcontroller-based system.

Procedure

a. **Instructor Preparation** – Make copies of *Introduction to Lego® NXT Programming Student Guide*, or *Introduction to Arduino System Programming Student Guide*; one copy for each student group.

b. **Pre-activity** – Ask students to explain what a microcontroller is and what it does. Students should realize (possibly with your help) that microcontrollers are small computers that control something. Be sure students understand that control, in this context, means using information about a system to take actions that cause this system (or another system) to change. The systems involved may include combinations of machines, processes, and/or people that are designed to work together to accomplish certain tasks.

Students should also know that microcontrollers can receive inputs, process data from input, and provide outputs that cause certain actions to take place, such as turning an appliance on or off, starting or stopping a motor, or printing a message. To do these things, a microcontroller typically includes:

- Memory that stores operating instructions, input data, and results of data processing;

- Input and output ports; and
- One or more timers.

Make sure students understand that inputs and outputs are electrical currents, also called signals, that convey information. Input signals often convey information about specific environmental conditions. This information is processed, which means that the information is analyzed according to instructions programmed into the microcontroller. Depending upon the results of the processing, the microcontroller may send one or more output signals to other devices. Students should recognize the similarity between the input-process-output sequence and the operation of the human brain which causes other body systems to react to information received from sensory inputs.

c. **Activity** – Provide each student group with materials and a copy of the *Student Guide* that are appropriate to the microcontroller system that they will be using. Review the overall procedure which first involves learning to program their microcontroller with a simple set of instructions, then developing a solution to the "Microcontroller Design Challenge." Emphasize that the Engineering Design Process (see page 11) should be used, and that each step in the process should be documented in each student's Engineering Design Journal. You may also want to introduce students to the common ground rules for brainstorming (also on page 11) to assist with this aspect of the Engineering Design Process. Be sure students understand that it is completely acceptable to try an idea that may not work, and that some technological problems are best solved through experimentation.

d. **Post-activity** – Have each student group present their solution to the SOS Challenge, then lead a discussion about their experiences with microcontrollers and the Engineering Design Process. Solutions to the SOS Challenge usually involve:

- creating a loop to generate three short flashes;
- creating another loop to generate three flashes that are three times longer than the short flashes; and
- creating a third loop that alternates three short flashes with three long flashes.

Slightly more sophisticated solutions will insert additional time delays to separate letters, and a longer delay between SOS groups.

During this discussion, reinforce the following concepts:

- For many design problems, the Engineering Design Process may include more steps than are needed to develop solutions for the SOS Challenge. In particular, many problems require designers to select materials from which technological solutions will be created. This selection requires considering the physical properties of potential materials, as well as their cost.

- Systems are a combination of components (including machines, processes, and/or people) that are designed to work together to accomplish certain tasks. Ask students to identify components in the microcontroller systems that they used for this activity. In addition to the microcontroller chip, students should identify several other components, including the light emitting device, the computer and software they used to program the microcontroller, and possibly themselves.

- Constraints are requirements that must be met by satisfactory design solutions. Most problems include constraints that are not necessarily specified in the problem statement, such as environmental conditions (e.g., conditions in the deep ocean), cost, and size.

- Trade-offs are often necessary in design solutions to meet the requirements imposed by constraints.

- Optimization is identifying a solution that meets all of the constraints with the best combination of trade-offs.

- Models are used to help visualize possible solutions, and may be two-dimensional, three-dimensional, or mathematical.

- Analysis is a systematic examination of information needed to proceed through each step of the Engineering Design Process.

Tell students that the Arduino microcontroller system was originally developed to be used by designers and artists so that they could use electronics to create prototypes of things they design. Since it was first introduced, this system has been used in many other ways by many other groups of people, and is an excellent example of:

- Technology transfer that occurs when a new user applies an innovation developed for one purpose to accomplish a different purpose;
- Systems that allow things to be done that could not be accomplished without the help of technology; and
- The close links between technology and creativity.

Ask students to describe some of the ways they think that microcontrollers might be part of technological systems that they use every day. Students should realize that specific uses are functions of need and setting, and that the versatility of microcontrollers helps increase the rate at which this technology is of developed and diffused. Ask students to list some other technological systems with which microcontrollers might interact, and how they might use a microcontroller system for a new purpose. Briefly remind students about ocean exploration strategies and tools used aboard the *Okeanos Explorer*, and discuss how microcontrollers contribute to advancement in science and mathematics.

Discuss whether the use of microcontrollers might have unintended consequences for human well-being or environmental quality. Could microcontrollers cause people to lose their jobs? Some students may be aware that some electronic components and manufacturing processes use toxic materials that can become pollutants. In 2003 the European Union adopted the "Directive on the restriction of the use of certain hazardous substances in electrical and electronic equipment," usually referred to as the Restriction of Hazardous Substances Directive or RoHS. This directive restricts the use of six hazardous materials in various types of electronic and electrical equipment:

- Lead;
- Mercury;
- Cadmium;
- Hexavalent chromium;
- Polybrominated biphenyls; and
- Polybrominated diphenyl ether.

Many electronic components are now marked as "RoHS Compliant," and this helps avoid a potential trade-off between environmental protection and using electronic technologies. Students should realize that technology is not inherently good nor bad, but decisions about the use of products and systems can result in desirable or undesirable consequences.

Resources

Baafi, E. 2011. Drag-n-Drop Arduino Programming. Make 25:52-56; describes Modkit, a free, web-based editor that uses graphical programming blocks to program Arduino boards

Banzi, M., D. Cuartielles, T. Igoe, G. Martino, and D. Mellis. Arduino [Internet]. Available from: *http://arduino.cc/en/*; Web site of the Arduino Team

Banzi, M. 2008. Getting Started with Arduino. O'Reilly Media, Inc. Sebastopol, CA. 118 pp.

Harrison, T. 2010. Links for Beginners in Electronics [Internet] Available from *http://www.toddfun.com/2010/02/09/beginners-in-electronics/*

Igoe, T. 2011. Getting Started with Microcontrollers. Make 25:42-45.

Kelly, J. R. 2010. Lego Mindstorms Nxt-g Programming Guide. Apress. New York. 336 pp.

Palmisano, J. S. Society of Robots [Internet]. Available from *http://www.societyofrobots.com/*; tutorials and basic information for beginning robotics enthusiasts

Introduction to Lego® NXT Programming Student Guide

Lego® NXT brick

Figure 4.

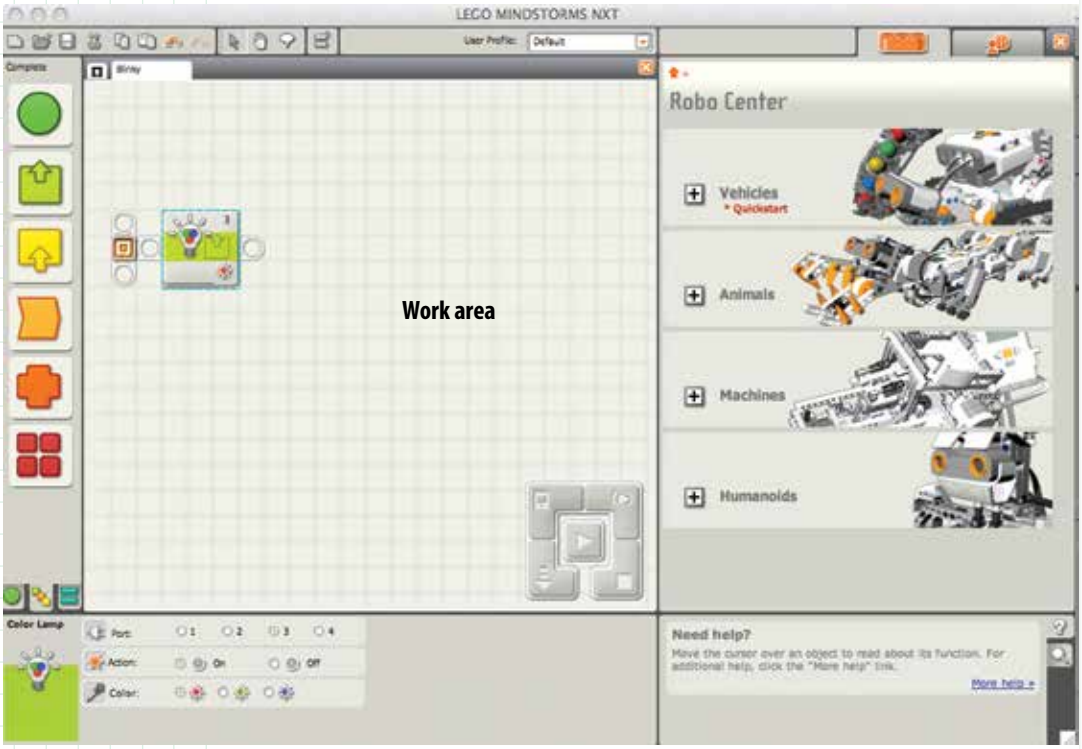
The Lego® NXT brick (see photo at right) has a liquid crystal (LCD) display, four control buttons, three output ports, four input ports, and a USB port. The brick is powered by six size AA batteries.

1. Launch the MINDSTORMS NXT software.

2. In the box beside "Create a new program" enter "Blinky" or whatever name you choose, then click on the "Go >>" button.

3. The programming palette on the left contains all of the building blocks for creating programs for the NXT microcontroller. For this program, click on the middle tab at the bottom (with the three colored squares), then scroll your mouse over on the light green icon (second from the top), and click on the Color Lamp programming block.

4. The area with the light gray grid is the Work Area where programming is done. Drag the Color Lamp block so that it is on top of the square outlined with a blue dashed line, and click to drop the block in this location. Now you have a program with two steps: Start, and Do-Something-With-the-Color-Lamp (Figure 4).



5. Many programming blocks can do several things. The Configuration panel on the lower left side of the window is where you specify what a programming block is supposed to do. For the Color Lamp block, the buttons next to "Port:" tell the microcontroller where the Color Lamp is connected. The buttons next to "Action" tell the microcontroller what to do with the Color Lamp. The buttons next to "Color"

tell the microcontroller the color to which the action applies. For now, be sure these buttons are set to Port 3, On, and Red.

6. Connect the Color Sensor to Port 3 of the NXT brick with a connector cable.

7. Turn on the NXT brick, and connect it to your computer's USB port with a USB cable.

8. In the NXT software, click the "Download and run" button on the Controller. When the program is finished compiling and downloading, the Color Lamp should be shining a red light.

9. Notice the two grid boxes to the left and right of the Color Lamp block that resemble pull tabs on soda cans. These are part of the sequence beam, which controls the flow of your program. Blocks must be connected to the sequence beam to be downloaded to the NXT microcontroller. You can find out more about the sequence beam and any other object in the Work Area by placing your cursor on top of the object and reading information in the Help box on the lower right.

10. Now let's make the Color Lamp turn on for one second, then turn off for one second, then turn back on, and repeat this cycle. In the programming palette, scroll your mouse over the green circle at the top, and click on the Wait programming block (it looks like an hourglass).

11. Drag the Wait block so that it is to the right of the Color Lamp block, and click to drop the block in this location. In the Configuration panel, select "Time" in the window to the right of "Control:". Be sure the "Seconds" window to the right of "Until:" is set to "1".

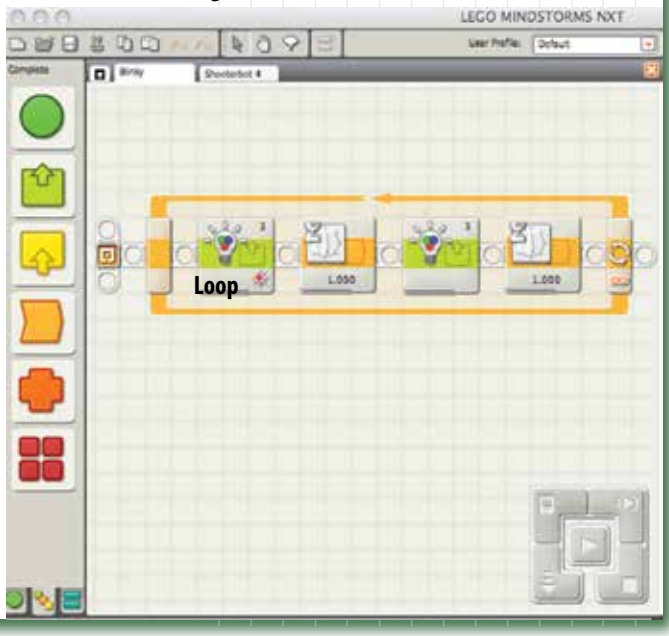
12. Scroll your mouse over the light green icon in the programming palette, and click on the Color Lamp programming block. Drag and drop this block to the right of the Wait block. In the Configuration panel, set Port to 3, Action to off, and Color to red.

13. Put another Wait programming block to the right of the Color Lamp block that was added in Step 12. In the Configuration panel, be sure that "Time" is set in the window to the right of "Control:", and that the "Seconds" window to the right of "Until:" is set to "1".

14. In the programming palette, scroll your mouse over the green circle at the top, and click on the Loop programming block (two curved arrows). Drag and drop this block to the right of the Wait block that was added in Step 13. Click on the button in the window to the right of "Control:" This window specifies how long the loop will continue to run. For now, set this window to "Forever" which means that programming instructions inside the loop will repeat until the microcontroller is turned off.

15. Now we have to put all of the programming steps that we want to repeat inside the loop. To do this, click the first step (the first Color Lamp block, then hold down the shift key and click on the other three steps. Click and hold on the selected blocks, and drag your cursor to the middle of the Hold block. The Loop block will expand to enclose all four steps (Figure 5).

Figure 5.



16. Click the "Download and run" button on the Controller. When the program is finished compiling and downloading, the Color Lamp should be blinking on for one second, then off for one second.

17. Experiment with setting in the Configuration panel for some of your program blocks. Try different wait times, and see how they affect the Color Lamp. Try using "Count" instead of "Forever" for the Loop block to make the Color Lamp blink a few times and then stop. Once the program stops, you can make it run again by pressing the orange button on the NXT brick.

18. The Engineering Design Process (see Box 1 on page11) is a series of steps that engineers use to create solutions to problems.

    Your challenge is to use the Engineering Design Process to create a program that will flash SOS in Morse code. In Morse code, S is three short flashes, and O is three long flashes (a long flash lasts three times longer than a short flash). This is a fairly simple challenge, so some steps in the Engineering Design Process may not take very long to complete; but be sure you record your results for each step in your Engineering Design Journal. Be sure to include:
    - Statement of the design problem
    - Relevant research
    - Possible solutions
    - Detailed plan for the selected solution
    - Results of testing and prototyping
    - Evaluation of testing and prototyping results, including modifications
    - Description of final solution

Box 1: Engineering Design Process

The Engineering Design Process is a series of steps that engineers use to create solutions to problems. There are many versions of the Process, but the basic steps include:

- Define the problem
- Gather relevant information
- Brainstorm possible solutions
- Analyze possible solutions and select the most promising
- Test the solution by building a prototype
- Revise and improve the solution
- Repeat previous steps until results are acceptable
- Report the design process and results

These steps involve several key skills:

- Obtaining, evaluating, and communicating information;
- Analyzing and interpreting data;
- Using mathematics, information and computer technology, and computational thinking; and
- Using evidence to discuss the strengths and weaknesses of ideas and designs.

Most problems will include certain constraints that may relate to cost, size, environmental conditions, or other specific requirements. Some constraints may be identified in the statement of the problem, but most problems need additional analysis to be certain that all constraints are understood. Often, constraints will force designers to make trade-offs in their solutions. For example the strongest material may be too expensive, or too heavy to meet cost and size constraints. Identifying the solution that meets all of the constraints with the best combination of trade-offs is called optimization. Models are frequently used to help designers visualize possible solutions, and may be two-dimensional illustrations, three-dimensional physical shapes, or mathematical calculations that predict how well a potential solution will do what is necessary to solve the problem. Each step of the Engineering Design Process involves systematically examining information that is needed to move to the next step. This kind of examination is called analysis.

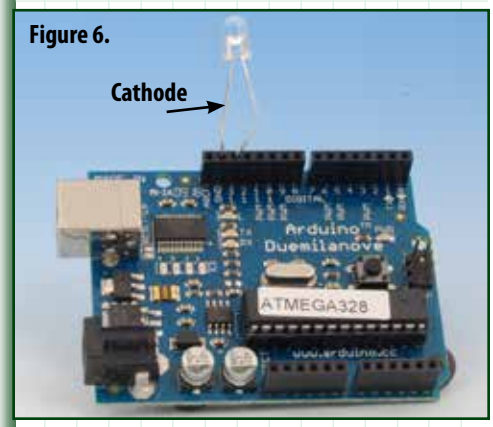
Common Ground Rules for Brainstorming

- The more ideas, the better! Groups should try to generate as many ideas as possible in a set amount of time.
- There are no bad ideas! No ideas should be criticized during a brainstorming session, so that people can offer ideas without worrying about what others may think. Unworkable or less satisfactory ideas will be weeded out later on.
- Weird is welcome! Unusual ideas are fine, and new ways of looking at a problem can provide better solutions.
- Make it better! Sometimes one good idea stimulates others. Ideas that are combinations or modifications of previous ideas are completely acceptable.

In some brainstorming sessions, participants write their ideas on separate "sticky notes." These can then be posted on a wall or marker board and re-arranged, discussed, combined, and modified to select the most promising approaches for solving the problem.

A microcontroller placed onto an Arduino board.

Figure 6.



Introduction to Arduino System Programming Student Guide

The Arduino board has 14 digital input/output pins (you use your programs to specify whether they are input or output) numbered 0 – 13; six analog input pins numbered 0 – 5: six analog output pins numbered 3, 5, 6, 9, 10, and 11; a USB port; and power supply jack (the board can use the USB port for power, but will automatically use a power supply if one is connected to the power supply jack).

1. Notice that one side of your LED is flattened. This is the cathode side, which is connected to the negative side of a power source.

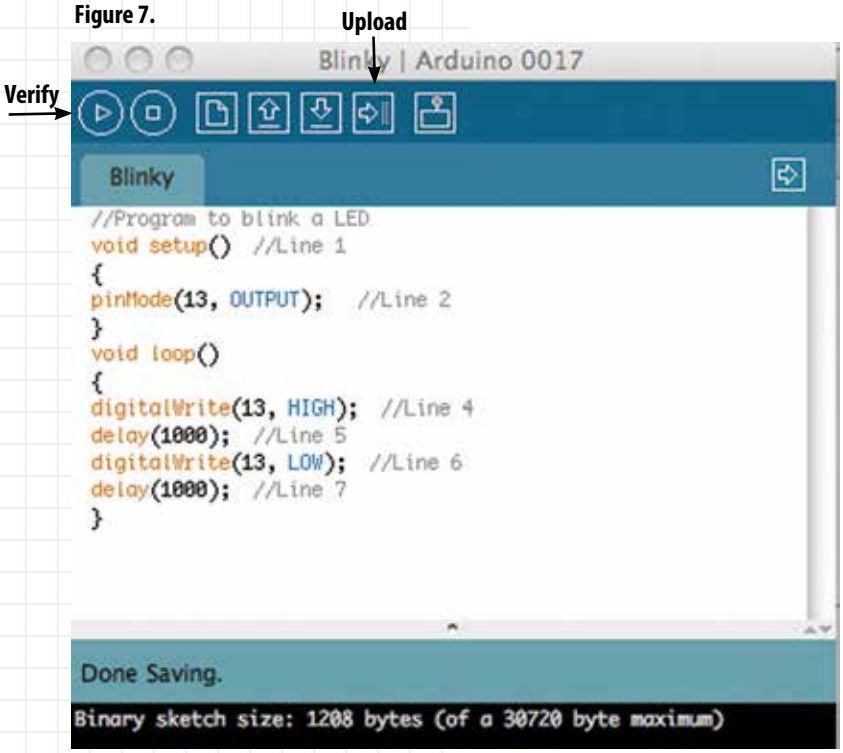
2. Insert the cathode lead from your LED into the socket marked "GND" and the other (anode) lead into socket 13 (Figure 6).

3. Connect the Arduino board to your computer with a USB cable.

4. Launch the Arduino software. Select "New" from the "File" menu. Select the folder in which you want your program to be stored, and name your program "Blinky" or or whatever name you choose.

5. In the Arduino world, programs are called "sketches". Type the following program into the Arduino Sketch Editor (the main window of the Arduino software). (See Figure 7):

Figure 7.




```
//Program to blink a LED
void setup()                    //Line 1
{
pinMode(13, OUTPUT);            //Line 2
}
void loop()                     //Line 3
{
digitalWrite(13, HIGH);         //Line 4
delay(1000);                    //Line 5
digitalWrite(13, LOW);          //Line 6
delay(1000);                    //Line 7
}
```

6. Click on the "Verify" button. If if everything is correct, the message "Done compiling" will appear at the bottom of the window. If an error message appears, carefully check your typing and be sure your program is identical to the one listed above.

7. Click on the "Upload button (second from right) to send your program to the Arduino microcontroller. Two small LEDs built into the board will flash, and in a few seconds, the "Done uploading" message will appear in the bottom window. Your program should now be running, and the LED should be should be blinking on for one second, then off for one second.

8. Experiment with the values in the delay functions, to see what effect they have on the pattern of blinking.

9. The Engineering Design Process (see Box 1 on page11) is a series of steps that engineers use to create solutions to problems.

   Your challenge is to use the Engineering Design Process to create a program that will flash SOS in Morse code. In Morse code, S is three short flashes, and O is three long flashes (a long flash lasts three times longer than a short flash). This is a fairly simple challenge, so some steps in the Engineering Design Process may not take very long to complete; but be sure you record your results for each step in your Engineering Design Journal. Be sure to include:
   - Statement of the design problem
   - Relevant research
   - Possible solutions
   - Detailed plan for the selected solution
   - Results of testing and prototyping
   - Evaluation of testing and prototyping results, including modifications
   - Description of final solution

10. Here's what the lines in the program mean:
    Any text that begins with // is ignored by Arduino. This kind of text is used for notes that say what is happening in various parts of the program. This is very useful for other people who may use your program, and also for you if you haven't looked at the program for a while and are trying to remember what each step does.

    Any text that doesn't begin with // is called "code," and provides specific directions to the microcontroller about what it is supposed to do and when it is supposed to do it. Each line of code must end with a semicolon (;)

    Curly braces {} are used to to enclose blocks of code that do certain things. Arduino expects that two blocks of code will be present in every program: a block named setup(), and a block named loop(). Curly braces must always be used in pairs.

    Line 1 says that the next block of code will be a function named setup(). The word "void" means that the function named setup() will not return any information once it has done what it is supposed to do. Other functions such as a math function like tan will return information; in this case, the tangent of a number.

    Line 2 sets digital pin number 13 to be an output.

    Line 3 says that the next block of code will be a function named loop(). The word void means the same thing as in Line 1.

    Line 4 sets digital pin number 13 to HIGH, which basically means that pin is turned

```
//Program to blink a LED
void setup()                    //Line 1
{



pinMode(13, OUTPUT);            //Line 2
}
void loop()                     //Line 3
{

digitalWrite(13, HIGH);         //Line 4



delay(1000);                    //Line 5


digitalWrite(13, LOW);          //Line 6

delay(1000);                    //Line 7


}
```

on and receives a voltage between three and five volts. The word digitalWrite is an instruction used to set a pin HIGH or LOW (low means that the pin receives one volt or less.

Line 5 tells the microcontroller to wait 1,000 milliseconds (one second) before doing anything else.

Line 6 sets digital pin number 13 to LOW, which means that pin is turned off.

Line 7 tells the microcontroller to wait another 1,000 milliseconds (one second) before doing anything else.

The curly brace following Line 7 is the end of the loop that begun with the curly brace just before Line 4. The microcontroller will continue repeating Steps 4 through 7 until power is turned off or another program is loaded.

Note for Educators

This activity is intended to familiarize students with very basic steps in using two popular types of microcontrollers. Students are not expected to be proficient programmers following this activity; only to know how to follow instructions for creating programs and how to upload such programs to the microcontroller they are using. Additional tutorials and applications can be found in the sources listed under Resources.

Solutions to the SOS challenge usually involve:

- creating a loop to generate three short flashes;
- creating another loop to generate three flashes that are three times longer than the short flashes; and
- creating a third loop that alternates three short flashes with three long flashes.

Slightly more sophisticated solutions will insert additional time delays to separate letters, and a longer delay between SOS groups.

Resources

Baafi, E. 2011. Drag-n-Drop Arduino Programming. Make 25:52-56; describes Modkit, a free, web-based editor that uses graphical programming blocks to program Arduino boards

Banzi, M., D. Cuartielles, T. Igoe, G. Martino, and D. Mellis. Arduino [Internet]. Available from: *http://arduino.cc/en/*; Web site of the Arduino Team

Banzi, M. 2008. Getting Started with Arduino. O'Reilly Media, Inc. Sebastopol, CA. 118 pages.

Harrison, T. 2010. Links for Beginners in Electronics [Internet] Available from *http://www.toddfun.com/2010/02/09/beginners-in-electronics/*

Igoe, T. 2011. Getting Started with Microcontrollers. *Make* 25:42-45.

Kelly, J. R. 2010. Lego Mindstorms NXT-G Programming Guide. Apress. New York. 336 pages.

Palmisano, J. S. Society of Robots [Internet]. Available from *http://www.societyofrobots.com/*; tutorials and basic information for beginning robotics enthusiasts

2. Making Things Happen with Servos

This activity is the second in a series of three activities that introduce basic systems used in many underwater robots to gather information for ocean exploration.

Microcontrollers, sensors, and servos are systems that allow robots to interact with their environment. Sensors provide information about the environment, microcontrollers process the information and send signals that cause the robot to take actions according to instructions that are included in the microcontroller's program. Often, those "action signals" are sent to servos.

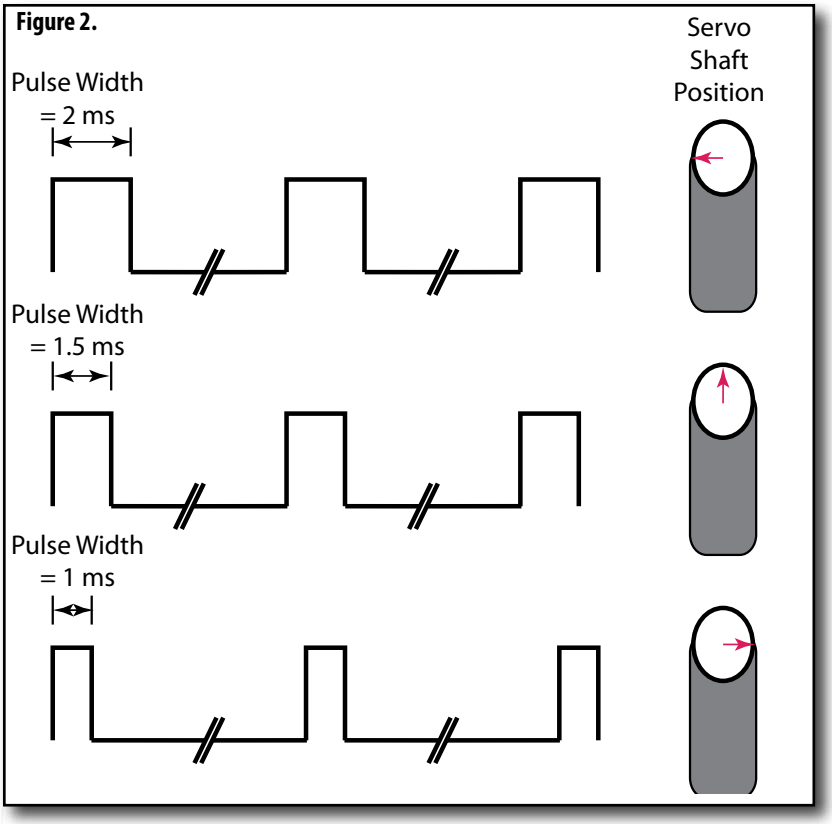
Servos are basically electric motors. But unlike familiar motors that spin continuously when they receive power, the motors in servos are combined with gears and electronic controls that cause them to rotate a certain number of degrees and then stop. This makes servos very useful when precise movements are needed, such as might be desired for a robotic arm. Servos are used in many familiar devices such as copy machines; computer controlled lathes and mills; and radio-controlled model cars, aircraft, and boats. Figure 1 shows a typical servo. The shaft of the servo motor is attached to a mechanical linkage called a servo horn that connects the servo to whatever is supposed to move when the shaft rotates. Servo horns come in many different sizes and shapes.

Figure 1. A typical servo.

Servos are controlled by pulsing digital signals, such as may be produced by a microcontroller. An example of this kind of signal is shown in Figure 2. The duration of each pulse tells the servo how much to rotate. In many servos, pulses 1 millisecond long tell the servo to rotate as far as it can go in one direction (either clockwise or counterclockwise). Pulses 2 milliseconds long instruct the servo to rotate as far as it can go in the opposite direction. Pulses between 1 and 2 milliseconds tell the servo to rotate

to positions in between these extremes. It is important to note that more than one pulse is needed. For a digital signal to cause a servo to rotate, there must be a series of pulses, separated by a certain time interval (usually between 20 and 50 milliseconds). This is an example of pulse-width modulation (PWM), which is a very important technique for controlling a variety of robotic systems.

Figure 2.



Typical servos are not able to rotate continuously through 360°; but it is possible to modify many servos so that they can rotate continuously. Why would anyone want to do that? Because servos are more than just electric motors; they also contain gears and electronic control circuits. A servo that is modified to provide continuous rotation is a high-quality, variable speed motor that can be controlled by simple signals from a microcontroller. Pulse width controls the rotation speed of a modified servo.

This activity uses a microcontroller to operate a servo. The objective of this activity is to program a microcontroller so that it will cause a servo to move in a controlled way. This will provide experience with programming, as well as an introduction to some of the capabilities of servos.

Materials

For a Lego® NXT brick system:

- NXT brick microcontroller
- Lego® Servo Motor
- Wheel and axle to fit Servo Motor hub
- USB cable
- Computer with Lego® Mindstorms NXT software installed

For an Arduino system:

Arduino Board (many versions are available; the Duemilanove is one of the simplest and easiest to learn)
Standard servo (*e.g.*, Futaba S3003; available from hobby stores and the Internet)
3 - Jumper wires to connect the servo cable to the Arduino Board
USB cable
Computer with Arduino software installed (download from *www.arduino.cc/en/Main/Software*; installation instructions are linked from the same page)

Procedure

Note: These instructions assume that you have completed the "Getting Control with Microcontrollers" activity.

For a Lego® NXT brick system:

Lego® Servo Motors have built-in rotation sensors that allow very precise control. The rotation sensors measure motor rotation in units of one degree, or in units of one full rotation.

Figure 3. Lego®Servo Motor with tire attached.

1. Attach a wheel and rubber tire to the Servo Motor (Figure 3). Connect the Servo Motor to Port A of the NXT brick with a connector cable. Turn on the NXT brick, and connect it to your computer's USB port with a USB cable.

2. Launch the MINDSTORMS NXT software.

3. From the "File" menu, select "New" then "Save As," enter "Servo" or whatever name you choose, then click "Save."

4. In the programming palette (on the left), click on the middle tab at the bottom (with the three colored squares), then scroll your mouse over on the green circle (at the top), and click on the Move programming block (gears).

5. Drag the Move block so that it is on top of the square outlined with a blue dashed line, and click to drop the block in this location.

6. The Configuration Menu (lower left) sets the Servo Motor's speed and direction of rotation. If you are controlling two Servo Motors, you can also specify whether they should run at the same speed (so a robot would move in a straight line) or at different speeds (so the robot would turn). If you have Servo Motors on the left and right side of a robot, you can also set the amount of turn and turn direction. Set "Port" to "A," and "Direction" to forward (arrow pointing up). In the windows to the right of "Duration," enter 5 and select "Rotations."

7. Click the "Download and run" button on the Controller. When the program is finished compiling and downloading, the wheel attached to the Servo Motor should rotate five times and then stop. You can press the orange button on the NXT brick if you want to run the program again.

8. Change your program so that the Servo Motor will rotate one-half of one revolution: In the windows to the right of "Duration," enter 180 and select "Degrees." Select the "Brake" button to the right of "Next Action."

9. Click the "Download and run" button on the Controller. When the program runs, the wheel attached to the Servo Motor should rotate one-half turn and then stop. Press the orange button on the NXT brick to run the program again, while you watch the wheel closely to confirm that it rotated one-half turn.

10. Change the "Duration" settings to rotate the wheel 90 degrees, and run the program to confirm that each rotation is one-fourth of a complete revolution.

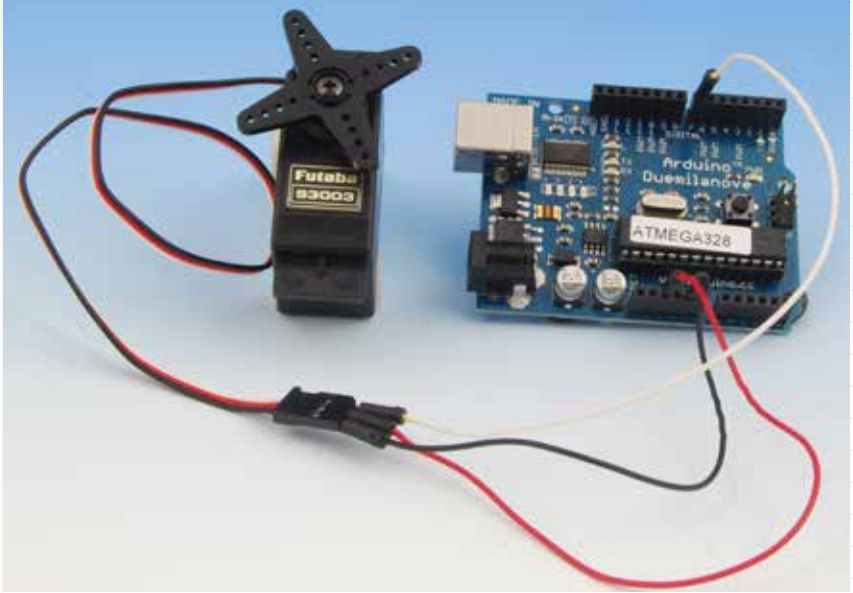
11. ***Challenge:*** Use a servo instead of hydraulics to control the simple actuator or part of the robotic arm constructed in the *Invent a Robot (http://oceanexplorer.noaa.gov/okeanos/edu/collection/media/hdwe-URRobot56.pdf)* lesson. You will have to:
    - Devise a way to mount the servo;
    - Connect the servo horn to the part to be moved;
    - Modify the program to provide the correct amount of movement.

For an Arduino system:

Servos have three wires that must be connected to the control system. The black wire connects to the negative side of the power supply; the red wire connects to the positive side of the power supply; and the white, yellow, or orange wire connects to the source of the digital signal that will control the servo.

1. Use jumper wires to connect the red wire from the servo to the "5V" socket on the Arduino board; the black wire from the servo to the "Gnd" socket on the Arduino board; and the white, yellow, or orange wire to digital pin 7 on the Arduino board (Figure 4).

Figure 4. Servo connected to the Arduino board.



2. Connect the Arduino board to your computer with a USB cable.

3. Launch the Arduino software. Select "New" from the "File" menu. Select the folder in which you want your program to be stored, and name your program "Servo" or or whatever name you choose.

4. Type the following program into the Arduino Sketch Editor (the main window of the Arduino software):

```
//Program to Test Servo Rotation
void setup()                                    //Line 1
{
pinMode(7,OUTPUT);                              //Line 2
}
void loop()                                     //Line 3
{
int counter;                                    //Line 4
for (counter= 0; counter<= 75; counter++)       //Line 5
{
digitalWrite(7,HIGH);                           //Line 6
delayMicroseconds(1000);                        //Line 7
digitalWrite(7,LOW);                            //Line 8
delay(20);                                      //Line 9
}
for (counter= 0; counter<= 75; counter++)       //Line 10
{
digitalWrite(7,HIGH);                           //Line 11
delayMicroseconds(1500);                        //Line 12
digitalWrite(7,LOW);                            //Line 13
delay(20);                                      //Line 14
}
for (counter= 0; counter<= 75; counter++)       //Line 15
{
digitalWrite(7,HIGH);                           //Line 16
delayMicroseconds(2000);                        //Line 17
digitalWrite(7,LOW);                            //Line 18
delay(20);                                      //Line 19
}
}
```

5. Click on the "Verify" button. If if everything is correct, the message "Done compiling" will appear at the bottom of the window. If an error message appears, carefully check your typing and be sure your program is identical to the one listed above.

6. Click on the "Upload" button (second from right) to send your program to the Arduino microcontroller. Two small LEDs built into the board will flash, and in a few seconds, the "Done uploading" message will appear in the bottom window. Your program should now be running, and the servo should be rotating from its full clockwise position, to its middle position, to its full counterclockwise position, then back to its full clockwise position. If the direction of the rotation sequence is opposite to this, it's OK; the important thing is that your servo is rotating between two positions with a stop in between.

7. Experiment with the delay time in Line 7. If the delay time is smaller, the pulses from this part of the program will be shorter. Does the servo rotate farther if the pulses are shorter? If your servo vibrates after it has rotated, it means that is has reached the limits of its rotation. **Do not leave your servo in a vibrating condition for very long, as it could damage the gear train.**

8. Experiment with the delay time in Line 17. If the delay time is larger, the pulses from this part of the program will be longer. Does the servo rotate farther if the pulses are longer? See the warning above about vibration.

9. Once you have determined the limits of rotation for your servo, try changing the delay time in Line 12. Can you position the servo horns exactly midway between the full clockwise and counterclockwise position?

10. Modify the program so that the servo starts at its full counterclockwise position, then travels in three equal steps to its full clockwise position, and then rotates back to the full counterclockwise position.

11. ***Challenge:*** Use a servo instead of hydraulics to control the simple actuator or part of the robotic arm constructed in the *Invent a Robot* (*http://oceanexplorer.noaa.gov/okeanos/edu/collection/media/hdwe-URRobot56.pdf*)lesson. You will have to:
    - Devise a way to mount the servo;
    - Connect the servo horn to the part to be moved;
    - Modify the program to provide the correct amount of movement.

12. Here's what the lines in the program mean:
    Any text that begins with // is ignored by Arduino. This kind of text is used for notes that say what is happening in various parts of the program. This is very useful for other people who may use your program, and also for you if you haven't looked at the program for a while and are trying to remember what each step does.

    Any text that doesn't begin with // is called "code," and provides specific directions to the microcontroller about what it is supposed to do and when it is supposed to do it. Each line of code must end with a semicolon (;), except "void" and "for" statements.

    Curly braces {} are used to to enclose blocks of code that do certain things. Arduino expects that two blocks of code will be present in every program: a block named setup(), and a block named loop(). Curly braces must always be used in pairs.

    Line 1 says that the next block of code will be a function named setup(). The word "void" means that the function named setup() will not return any information once it has done what it is supposed to do. Other functions such as a math function like tan will return information; in this case, the tangent of a number.

    Line 2 sets digital pin number 7 to be an output.

    Line 3 says that the next block of code will be a function named loop(). The word void means the same thing as in Line 1.

    Line 4 says that the program will use a variable named "counter" and that it is an integer.

    Line 5 is a "for" statement which says that the program is supposed to repeat the next block of code enclosed by curly braces, and to keep on repeating it until a certain condition is met. The "for" statement includes three pieces of information in parentheses, separated by semicolons: (1) the starting condition of some variable, in this case "counter" which is given a starting value of 0; (2) the condition that must

be met for the code to repeat again, in this case that "counter" is less than or equal to 75; and (3) how much the variable increases or decreases with each repetition, in this case "++" means that the value of "counter" increases by 1 with each repetition (if "++" were replaced with "--" the value of "counter would be decreased by 1 with each repetition). The result of this statement is that the following block of code (Lines 6 through 9) will repeat 75 times.

Line 6 sets digital pin number 7 to HIGH, which basically means that pin is turned on and receives a voltage between three and five volts. The word digitalWrite is an instruction used to set a pin HIGH or LOW (low means that the pin receives one volt or less).

Line 7 tells the microcontroller to wait 1,000 microseconds (one millisecond) before doing anything else. Since digital pin number 7 is HIGH during this time, the pulse width will be 1 millisecond. This instruction sets the width of the digital pulses.

Line 8 sets digital pin number 7 to LOW, which means that pin is turned off.

Line 9 tells the microcontroller to wait another 20 milliseconds before doing anything else. This sets the interval between pulses.

Lines 10 and 15 are similar to Line 5.

Lines 11 and 16 are similar to Line 6.

Lines 12 and 17 are similar to Line 7, and set the width of the digital pulses that are sent by this part of the program.

Lines 13 and 18 are similar to Line 8.

Lines 14 and 19 are similar to Line 9, and set the interval between pulses that are sent by this part of the program.

Resources

Kurt, T.E. 2010. Servomotors. *Make* 19:140-147.

Palmisano, J. S. Actuators - How to Modify a Servo [Internet]. Available from: *http://www.societyofrobots.com/actuators_modifyservo.shtml*; directions for modifying a servo so that it rotates continuously

Palmisano, J. S. Actuators - Mechanics - Robot Chassis Construction [Internet]. Available from: *http://www.societyofrobots.com/mechanics_chassisconstruction.shtml#wheels*; how to mount a wheel on a servo; and *http://www.societyofrobots.com/mechanics_chassisconstruction.shtml#servos*; how to attach a servo to a robot chassis

3. Exploring with Sensors

This activity is the third in a series of three activities that introduce basic systems used in many underwater robots to gather information for ocean exploration.

Microcontrollers, sensors, and servos are systems that allow robots to interact with their environment. Microcontrollers send signals to servos and other actuators that cause the robot to take actions according to instructions that are included in the microcontroller's program. Often, those "action signals" are sent as a response to environmental information that comes from sensors.

Ocean explorers use many types of sensors that may be attached to ships, CTDs, underwater robots, or used independently. These include sensors that measure pressure (often to determine depth), temperature, salinity, pH, dissolved oxygen, redox potential (an indicator of hydrothermal vent activity), optical backscatter (an indicator of methane oxidation), and colored dissolved organic matter (an indicator of petroleum products). Some of these sensors are also available for use in classrooms. Many others are suitable for use with microcontrollers, including sensors that measure acceleration, force, bending, magnetic fields, light, infrared, motion, humidity, and distance using ultrasonic waves (see the *PING)))™ Sonar Simulation* activity in the Multibeam Mapping section of the *NOAA Ship Okeanos Explorer Education Materials Collection, Volume 2: How Do We Explore?* Additional Technology Activities for an example of the latter sensor).

The majority of sensors measure various environmental conditions with electric circuits whose current flow changes in response to changes in the environment. Environmental conditions are related to changes in current flow by calibration. This involves graphing changes in current flow against known levels of the environmental condition that caused these changes. Information from calibration graphs is often built into modern instruments, so the instruments appear to provide direct readings of the condition they are supposed to measure; but these readings are based on current flow in an electric circuit.

In addition to sensors that can be used with microcontrollers, Vernier Software and Technology produces a wide variety of data collection systems specifically designed for classroom educators. These systems are based on sensors that are connected to handheld calculators or computers, and include more than 40 different types of sensors. Adapters are available to connect Vernier sensors to NXT bricks, and Vernier offers cables, connectors, and adapters that allow similar connection to other microcontrollers.

This activity uses a microcontroller to read a photocell (also called a photoresistor), which is a sensor whose electrical resistance changes with changing light levels.

Materials

For a Lego® NXT brick system:

- NXT brick microcontroller
- Lego® Servo Motor
- USB cable
- Computer with Lego Mindstorms NXT software installed
- Colored filters, approximately 2" x 2"; red, green, and blue

For an Arduino system:

Arduino Board (many versions are available; the Duemilanove is one of the simplest and easiest to learn)
Prototyping board (most types will work; the Adafruit Protoshield connects directly to the Arduino board, and has other nice features (*http://www.adafruit.com/index.php?main_page=product_info&cPath=17_21&products_id=51*)
3 - Jumper wires to connect the servo cable to the Arduino Board
Cadmium sulfide photocells (also called photoresistors; often available in "variety packs;" *e.g.*, Radio Shack 276-1657)
Resistor - see Notes to Educators
Colored filters, approximately 2" x 2"; red, green, and blue
USB cable
Computer with Arduino software installed (download from *www.arduino.cc/en/Main/Software*; installation instructions are linked from the same page)

Procedure

Note: These instructions assume that you have completed the "Getting Control with Microcontrollers" activity.

For a Lego® System

1. Connect the Color Sensor to Port 3 of the NXT brick with a connector cable. Turn the brick on by pushing the orange button.

2. Launch the MINDSTORMS NXT software.

3. In the box beside "Create a new program" enter "ColorSensor" or whatever name you choose, then click on the "Go >>" button.

4. In the programming palette (on the left), click on the middle tab at the bottom (with the three colored squares), then scroll your mouse over on the yellow square with an arrow, and click on the Color Sensor programming block (with the red, green, and blue circles).

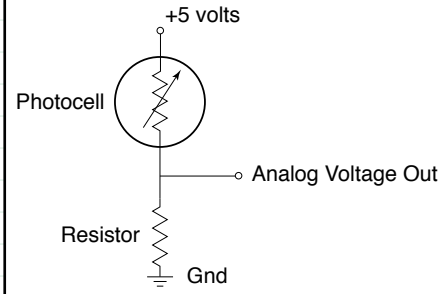
5. Drag the Color Sensor block so that it is on top of the square outlined with a blue dashed line, and click to drop the block in this location.

6. In the Configuration Menu (lower left) Set "Port" to "3," and "Action" to "Light Sensor." The "Light" box to the right of "Function" should be unchecked.

7. The feedback box on the left side of the configuration panel shows the current light value (0-100) being measured by the sensor. Hold your hand over the photocell, and the reading should decrease. Increase the light falling on the photocell and the reading should increase.

8. ***Challenge:*** Are photocells equally sensitive to all colors of light? Use the colored filters to find out.

Figure 1. A simple circuit.



For an Arduino System:

This activity requires a simple circuit (Figure 1). In this circuit, the photocell is connected to a resistor, and power leads are connected so that current flows through the resistor and photocell. An analog input port of the microcontroller is connected to read the

voltage at the junction between the resistor and photocell. As more light strikes the photocell, its resistance decreases, so the voltage increases.

The easiest way to build this circuit is with a prototyping board. Figure 2 shows a prototyping board next to an Arduino board, and Figure 3 shows the two boards attached. When the two boards are joined, sockets from the Arduino board are connected to sockets on the prototyping board to simplify circuit building.

Figure 2. Arduino board (left) and prototyping board (right).

Figure 3.

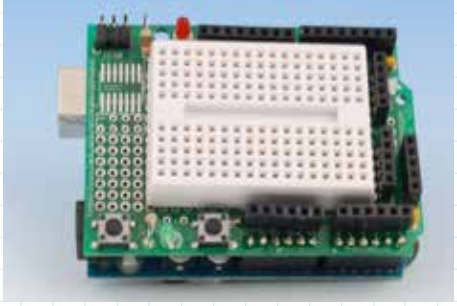


Figure 3. A prototyping board attached to an Arduino board. Some of the holes are electrically joined by connections inside the board. On this board, there are 17 vertically-oriented sets of 5 connected holes on the top half of the board, and another 17 vertically-oriented sets of 5 connected holes on the bottom half of the board. Holes on the top half of the board are not connected to holes on the bottom half of the board. These holes are used to connect the components of various circuits. Since there are 34 separate sets of holes, this board can hold a lot of components!

1. Install the cadmium sulfide photocell so that its two leads are in separate sets of holes (Figure 4).

2. Install the resistor so that one of its leads is in the same set as one of the leads from the photocell, and the other lead is in a set of holes by itself (Figure 5).

Figure 4.

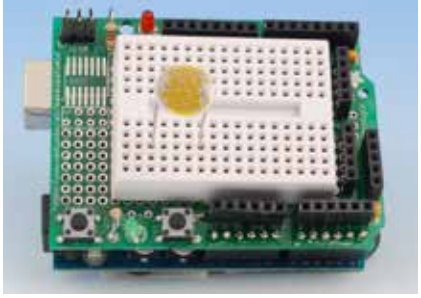
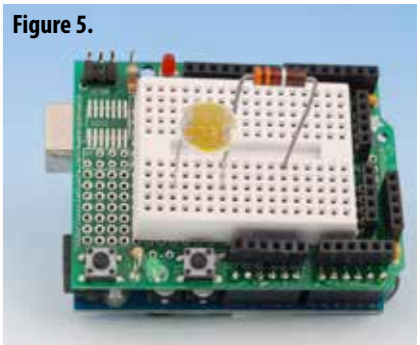
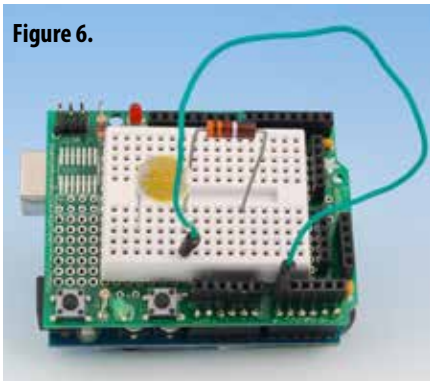


Figure 5.



3. Connect the set of holes that has a lead from the photocell AND a lead from the resistor to the socket for Analog pin 0 from the Arduino board. (Figure 6).

Figure 6.



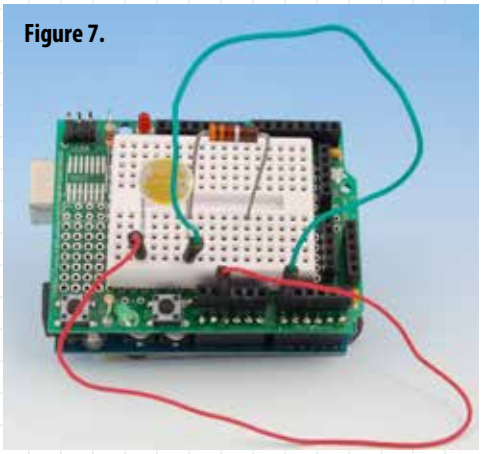


Figure 7.

4. Connect the other lead from the photocell to the 5V socket on the prototyping board. (Figure 7)

5. Connect the other lead from the resistor to the the Gnd socket on the prototyping board. (Figure 8).

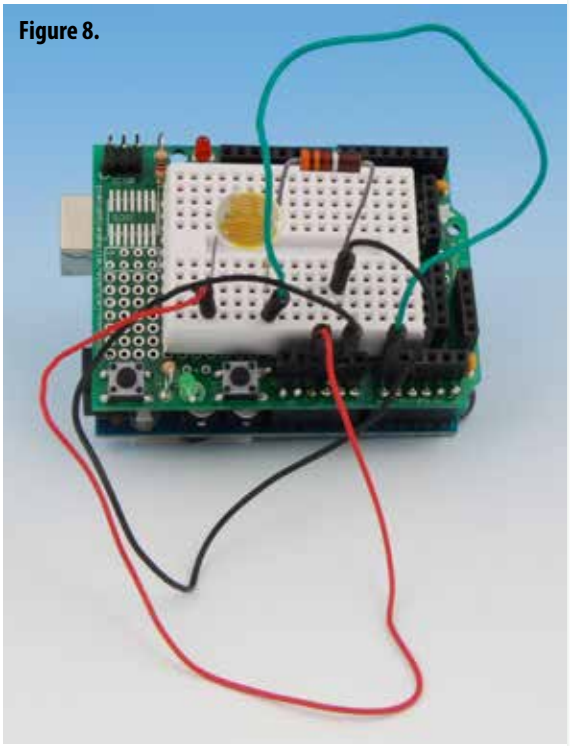


Figure 8.

6. Connect the Arduino board to your computer with a USB cable.

7. Launch the Arduino software. Select "New" from the "File" menu. Select the folder in which you want your program to be stored, and name your program "Servo" or or whatever name you choose.

8. Type the following program into the Arduino Sketch Editor (the main window of the Arduino software):

```
//Program to read photocell
void setup()                        //Line 1
{
 Serial.begin(9600);                //Line 2
}
 void loop()                        //Line 3
{
 Serial.print('Analog reading = ');  //Line 4
 Serial.println(analogRead(0));     //Line 5
 delay(2000);                       //Line 6
}
```

9. Click on the "Verify" button. If everything is correct, the message "Done compiling" will appear at the bottom of the window. If an error message appears, carefully check your typing and be sure your program is identical to the one listed above.

10. Click on the "Upload" button (second from right) to send your program to the Arduino microcontroller. Two small LEDs built into the board will flash, and in a few seconds, the "Done uploading" message will appear in the bottom window. Your program should now be running, reading the voltage at the junction of the photocell and resistor, and printing the reading to the serial monitor every two seconds. To see the serial monitor, click on the serial monitor button (on the right) at the top of the Sketch Editor window. You should see a series of readings. Hold your hand over the

photocell, and the reading should decrease. Increase the light falling on the photocell and the reading should increase.

11. ***Challenge:*** Are photocells equally sensitive to all colors of light? Use the colored filters to find out.

12. For more things you can do with photocells and microcontrollers, see *http://learn.adafruit.com/photocells*.

13. Here's what the lines in the program mean:
You should already know about //, semicolons, and curly braces. If not, review the *Getting Control with Microcontrollers* activity.

Line 1 says that the next block of code will be a function named setup().

Line 2 sets the data rate in bits per second (baud) for serial data transmission. In this case it is set to 9600 baud. Some computers may need lower rates, or may accept higher rates. You can check the documentation for your computer's USB port to find out.

Line 3 says that the next block of code will be a function named loop().

Line 4 sends a stream of text to the serial port. The content of the text is defined by the characters between the single quote marks in parentheses.

Line 5 sends the value read at analog pin 0 to the serial port, followed by a carriage return.

Line 6 tells the microcontroller to wait 2,000 milliseconds (2 seconds) before repeating the loop.

Resources

Anonymous. Vernier Software and Technology [Internet]. Available from: *http://www.vernier.com/*; Web site describing sensors and interfaces for measuring a wide variety of environmental parameters; includes lesson plans and activities

Fried, L. Photocells [Internet]. Adafruit Industries. Available from: *http://learn.adafruit.com/photocells*; one of several tutorials on microcontrollers and electronics

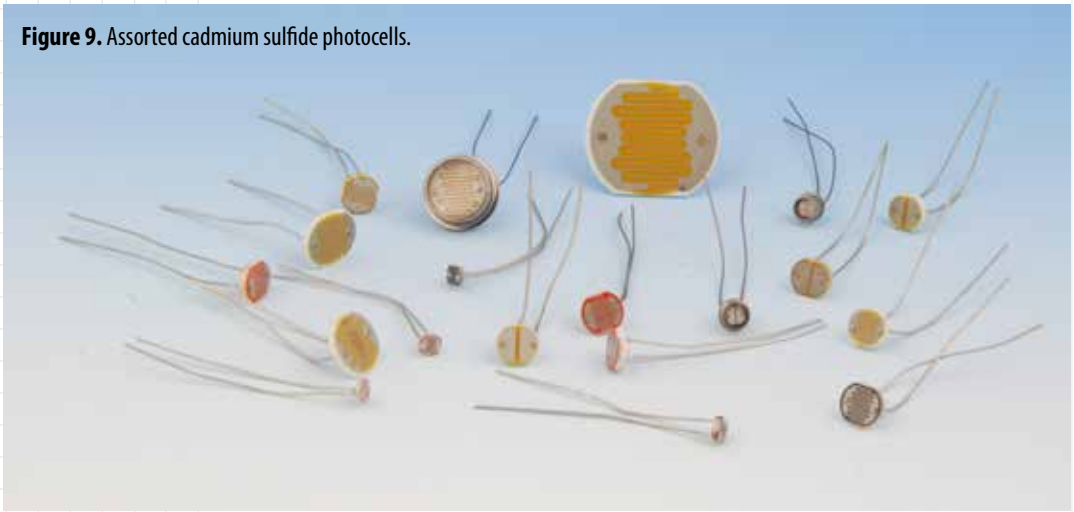
Palmisano, J. S. Schematics - Photoresistor [[Internet]. Available from: *http://www.societyofrobots.com/schematics_photoresistor.shtml*; directions for working with photocells

Notes to Educators

1. Most photocells are not equally sensitive to all wavelengths of light. Students should find that photocells are more sensitive to red and green light than to blue light.

2. For Arduino users: Cadmium sulfide photocells come in many shapes and sizes, and most will work for this activity. Figure 9 shows an assortment that was purchased for a few dollars from an electronic supply store. The optimum size for the resistor in the circuit shown in Figure 1 depends upon the characteristics of the specific photocell used for the circuit. There are two ways to find the resistor size:

Figure 9. Assorted cadmium sulfide photocells.



(a) If you have a multimeter, set it to measure resistance (ohms) and measure the resistance of the photocell when it is fully illuminated (exposed to the brightest light available), and measure the resistance again when the photocell is completely shielded from light. Multiply the two resistance measurements, and take the square root of the result. This is the correct value for the resistor.

(b) If you do not have a multimeter, buy resistors in the following sizes: 1,000 ohm; 2,200 ohm; 3,300 ohm; 4,700 ohm, 6,800 ohm, and 10,000 ohm. Starting with the 10,000 ohm resistor, try different resistors in the circuit until a satisfactory result is obtained. "Satisfactory" means that the photocell does not saturate (readings stop changing) at high or low light levels. What you want is for every different light intensity to produce a unique number in the microcontroller output.

Send Us Your Feedback

We value your feedback on these lessons, including how you use it in your formal/informal education settings. Please send your comments to:
oceanexeducation@noaa.gov

For More Information

Paula Keener, Director, Education Programs
NOAA Office of Ocean Exploration and Research
Hollings Marine Laboratory
331 Fort Johnson Road, Charleston SC 29412
843.762.8818 843.762.8737 (fax)
paula.keener-chavis@noaa.gov

Acknowledgments

Produced by Mel Goodwin, PhD, Marine Biologist and Science Writer, Charleston, SC. Design/layout: Coastal Images Graphic Design, Charleston, SC.

Other Resources

Bohm, H. and V. Jensen. 1998. Build Your Own Programmable Lego Submersible: Project: Sea Angel AUV (Autonomous Underwater Vehicle). Westcoast Words. 39 pp.

Bohm, H. 1997. Build Your Own Underwater Robot and Other Wet Projects. Westcoast Words. 148 pp.

Pinner, W. 2010. Bulk Geo-Tagging of Images Using SCS Timestamped NMEA GGA, HDT and ROV Data [Internet]. OceanDataRat.org. [cited February 2, 2011]. Available from *http://www.oceandatarat.org/?p=124*

Sea Perch Program [Internet] *http://www.seaperch.org/index*. Detailed instructions for building a simple remotely operated underwater vehicle; based on designs from "Build Your Own Under Water Robot and Other Wet Projects" by Harry Bohm and Vickie Jensen can be found at *http://www.seaperch.org/teacher_tools*.

Images:

The newest ROV, the *Deep Discoverer*
http://oceanexplorer.noaa.gov/okeanos/explorations/ex1304/background/plan/media/rov.html

The ROV *Little Hercules* aboard the *Okeanos Explorer*.
http://oceanexplorer.noaa.gov/okeanos/explorations/ex1104/background/hires/plan_little_herc_hires.jpg

The ROV sled *Seirios* being deployed from the *Okeanos Explorer*.
http://oceanexplorer.noaa.gov/okeanos/explorations/ex1104/background/plan/media/plan_seirios_launching.html

The *Quest 4000* remotely operated vehicle during the Submarine Ring of Fire 2012: Northeast Lau Basin Expedition.
http://oceanexplorer.noaa.gov/explorations/12fire/logs/sept19/media/quest4000.html

The *Quest 4000* remotely operated vehicle collecting geological samples with the manipulator claw during the Submarine Ring of Fire 2012: Northeast Lau Basin Expedition.
http://oceanexplorer.noaa.gov/explorations/12fire/logs/sept19/media/claw.html

The Autonomous Benthic Explorer (ABE), a free-swimming robot.
http://oceanexplorer.noaa.gov/explorations/10chile/background/exploration/media/exploration1.html

NOAA Ocean Explorer Web page on submersible technology; includes many images of underwater vehicles.
http://oceanexplorer.noaa.gov/technology/subs/subs.html

Video:

Dave Lovalvo, project manager for NOAA OER's Deep Submergence Group gives a tour of the new 6000m rated ROV system his team built.
http://oceanexplorer.noaa.gov/okeanos/explorations/ex1302/edu.html

Geologists and biologists explore the sea floor during the March 23 ROV dive from the NOAA Ship *Okeanos Explorer* during the Gulf of Mexico 2012 Expedition.
http://oceanexplorer.noaa.gov/okeanos/explorations/ex1202/logs/mar23/media/movies/highlights0323_wm640.wmv

A gas capture device is assembled and installed on the *Little Hercules* ROV, then deployed by the team on the *Okeanos Explorer* during the Gulf of Mexico 2012 Expedition.
http://oceanexplorer.noaa.gov/okeanos/explorations/ex1202/logs/apr18/media/movies/highlights0415_video.html