LIGHT TRACKER ACTIVITY

Updated 2/10/2017

This project is based on the instructable by Mathias Leroy http://www.instructables.com/id/Solar-tracker-with-Lego-and-Arduino-Dual-Axis/

1 INTRODUCTION

As a renewable resource, solar energy is becoming one of the most popular ways of producing electricity around the world. Two common ways of collecting solar energy involve the heliostat and solar panels. Heliostats consist of large mirrors that reflect the sun to one solar receiver while each solar panel converts the sun’s energy to current to provide power. How a heliostat works can be seen on the right side of image A in the figure below; image B shows how a solar panel works.

One of the challenges with solar energy is that to get the maximum energy, the system needs to be pointed directly at the sun. There are a couple methods to do this: one method uses software to track the sun based on the Earth’s orbit and the current date and time. The second method uses sensors to determine the location of the sun and move the system accordingly.

In this activity, a combination of Lego®’s, servo motors, circuit components, and a micro-controller (pictured below) will be used to track light. The same concepts used in this activity can be applied to both heliostats and solar panels to maximize the amount of energy produced during the day.
# 2 Objectives

The objective of this project is to build and test a light tracker while introducing students to concepts that are frequently used in mechanical engineering, electrical engineering, computer programming, and mathematics.

## 3 Parts

<table>
<thead>
<tr>
<th>Lego® Parts</th>
<th>Quantity</th>
<th>Picture</th>
</tr>
</thead>
<tbody>
<tr>
<td>Flat 1x6</td>
<td>7</td>
<td><img src="image1.png" alt="Flat 1x6" /></td>
</tr>
<tr>
<td>Flat 1x8</td>
<td>7</td>
<td><img src="image2.png" alt="Flat 1x8" /></td>
</tr>
<tr>
<td>Flat 2x8</td>
<td>3</td>
<td><img src="image3.png" alt="Flat 2x8" /></td>
</tr>
<tr>
<td>Flat 4x8</td>
<td>4</td>
<td><img src="image4.png" alt="Flat 4x8" /></td>
</tr>
<tr>
<td>Flat 4x4 with LDR holes</td>
<td>4</td>
<td><img src="image5.png" alt="Flat 4x4 with LDR holes" /></td>
</tr>
<tr>
<td>Flat 4x4 hollow</td>
<td>4</td>
<td><img src="image6.png" alt="Flat 4x4 hollow" /></td>
</tr>
<tr>
<td>Flat 4x4</td>
<td>1</td>
<td><img src="image7.png" alt="Flat 4x4" /></td>
</tr>
<tr>
<td>Normal 1x8</td>
<td>13</td>
<td><img src="image8.png" alt="Normal 1x8" /></td>
</tr>
<tr>
<td>Normal 2x4</td>
<td>8</td>
<td><img src="image9.png" alt="Normal 2x4" /></td>
</tr>
<tr>
<td>Base plate</td>
<td>1</td>
<td><img src="image10.png" alt="Base plate" /></td>
</tr>
<tr>
<td><strong>Electronic Parts</strong></td>
<td><strong>Quantity</strong></td>
<td><strong>Picture</strong></td>
</tr>
<tr>
<td>--------------------------------------</td>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>Arduino Duemilanove</td>
<td>1</td>
<td>N/A</td>
</tr>
<tr>
<td>Arduino USB cable</td>
<td>1</td>
<td>N/A</td>
</tr>
<tr>
<td>Bread board</td>
<td>1</td>
<td>N/A</td>
</tr>
<tr>
<td>10k Ohm resistor</td>
<td>4</td>
<td>N/A</td>
</tr>
<tr>
<td>Light dependent resistors (LDRs)</td>
<td>4</td>
<td><img src="image.jpg" alt="LDRs" /></td>
</tr>
<tr>
<td>Round end pin-pin wires</td>
<td>14</td>
<td><img src="image.jpg" alt="Round end wires" /></td>
</tr>
<tr>
<td>Block end pin-pin wires</td>
<td>8 total</td>
<td><img src="image.jpg" alt="Block end wires" /></td>
</tr>
<tr>
<td>(4 if still attached together)</td>
<td></td>
<td></td>
</tr>
<tr>
<td>Block end socket-socket wires</td>
<td>8 total</td>
<td><img src="image.jpg" alt="Block end socket-socket wires" /></td>
</tr>
<tr>
<td>(4 if still attached together)</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
4 INSTRUCTION

This section will include three subsections: tracker construction, wiring, and base construction. The first subsection describes how all the moving parts will be put together. The second subsection focuses on the wiring and electronics part of the activity; more information on the electronic components and the coding for the Arduino can be found in Appendix A. The last subsection explains how to build the base for the tracker system. This base doesn’t have to look identical to the demonstration unit, but it must be able to hold the breadboard, Arduino microcontroller, and the tracker in place.

4.1 TRACKER CONSTRUCTION

Tracker construction consists of building two parts—the dual axis servo stand and the tracking head. To build the tracker head, follow the steps below.

1. Connect two flat 4x4 (with LDR holes) Lego®’s using one flat 1x6 Lego®. Repeat for the other two flat 4x4 Lego®’s with LDR holes. The result is shown below.

2. Use two regular 1x8 Lego®’s to connect the two sets together in a square, as shown below.
3. Use a normal 2x4 Lego® in the middle of the square to attach the sets. The normal 2x4 long side should be parallel to the normal 1x8 long sides.

4. Install an LDR onto the Lego®’s by threading the two LDR wires through the two holes of the Lego® piece. Connect the LDR wires to the jumper wires. Repeat for the three other LDRs.
   a. To connect the LDR wires to the jumper wires, gently push the LDR wire leads into the black sockets at the end of the jumper wires. You should hear a subtle click when the LDR wire lead is properly connected to the black socket. To check the connection, very gently pull the LDR wire away from the socket; if properly connected, the wire will not come out of the socket.
b. Take note of which colors are connected to which LDRs because this will be important later on. The LDRs are labeled as follows when you’re looking at the sensor side of the head.

<table>
<thead>
<tr>
<th>Top Left</th>
<th>Top Right</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bottom Left</td>
<td>Bottom Right</td>
</tr>
</tbody>
</table>

5. Connect two of the flat 4x4 hollow pieces in the same fashion as step 1. Repeat to make another set.

6. Attach the flat hollow sets of 4x4 Lego®’s to the backs of the other assembled Lego®’s, putting the holes over the wires. To make it easier later one fill write the wire colors that go to each location in the table below.

<table>
<thead>
<tr>
<th>Top Left Wire Colors</th>
<th>Top Right Wire Colors</th>
</tr>
</thead>
<tbody>
<tr>
<td>Bottom Left Wire Colors</td>
<td>Bottom Right Wire Colors</td>
</tr>
</tbody>
</table>

Wire Color Reference Table

7. To make the head the proper height, add a stack of two normal 2x4 Lego®’s to the center of the assembly, attaching it to the flat hollow 4x4 sets. To maximize the amount of Lego®’s being held together, align this stack perpendicular to the first 2x4 Lego®, mentioned in step 3.
8. To ensure the head of the tracker fits into the stand, use one flat 4x4 Lego® on the bottom of the tracker, attaching it to the stack created in step 7.

9. Carefully slide the tracker head into the stand; the flat 4x4 piece should be the bottom-most part of the tracker, and its uncovered edges should fit neatly beneath the stand’s lipped edges. The Lego® is slightly larger than the stand, so the edges of the stand will have to bend a little (do not bend them so much that they break). Note: the LDRs at the top of the tracker head will be positioned towards the back of the stand, which is the side with the wires coming out of the mount.
10. Attach the stand to the front and center of the base plate using two normal 1x8 Lego®’s (otherwise, the screws in the customized Lego® will keep it from directly attaching to the base plate). In the photo below, three normal 2x6 Lego®’s act as a counterweight at the back of the stand piece to help keep the stand upright while it is moving. (This counterweight is not completely necessary—it’s just an added precaution to make sure nothing falls over.)

11. To complete the tracker head, put in the light blockers. The light blockers will be made from four 4x8 flat Lego®’s. Slide the 4x8 flat Lego®’s between each of the flat LDR Lego®’s, as shown in the pictures below. The 4x8 flat Lego® will fit between the 4 square LDR Lego®’s they tend to fall off easily so you may have to reinstall them several times.
1. Using the bread board diagram below, make the resistor circuit on the bread board and the Arduino. (Yes, the Arduino in the picture is not the one we are using, but the pins are the same—so do not panic!)

   a. The columns in the 2-column section bracketed by the red and blue stripes are connected and that the other, lettered, sections are connected along rows. Start putting in the resistors.
b. Now use square pin-pin jumper wires to start connecting the bread board to the LDRs. (The LDRs pictured below are organized as they would be if you were looking at the sensor side of the tracker head.)

c. Connect the ground (Gnd) and power (5V) pins to the bread board using round pin-pin jumper wires. (I usually use red for power and black for ground to keep things straight.)

d. Use more round pin-pin jumper wires to distribute the ground and power along the bread board.

e. Now connect the resistors to the analog pins as specified in the table and figure below.
2. Make the servo circuit.
a. The servo wiring is set up as shown below, using the bread board connections for the ground and power on the servos.

b. Using round pin-pin jumper wires, connect PWM pin 09 to the horizontal servos signal socket and PWM pin 10 to the signal socket of the vertical servo.

3. The complete circuit should look like the one below.
4.3 BASE CONSTRUCTION

Make two sections in the back of the base plate to hold the Arduino and bread board. The pictures below show one possible way to do this, but you don’t have to do it that way—you can design it however you want as long as the base can hold the Arduino microcontroller, breadboard, and tracker in place.

5 TESTING

Now that you are done building the solar tracker, plug the power cord into the Arduino and test the system with a flashlight.

Once plugged in, the tracker will move to its specified start position according to its coding (discussed in Appendix B). Afterwards it will attempt to track the light source. Because of the physics of how they work, the light dependent resistors (LDRs) do not sense the light immediately, so they may not be able to see the flashlight if it is moving too quickly. This error is common in sensors and will be discussed in the appendix.

The tracker works based on differences in the light intensity it is reading in each quadrant. If too much background light is in the area, the tracker may not be able to determine where the maximum amount of light is, so it may move in random directions. If an error was made in wiring, the tracker may move in the wrong direction or even not move at all.

**Question:** Is the tracker working properly? If not, what could be causing it to not work (e.g., too much ambient light in the room, an error in the wiring, or something else)?
APPENDIX A: BASIC CIRCUITS

PHYSICS BEHIND ELECTRICITY

We must understand some basic physics principles to control electricity. The universe is made of matter; everything around you takes up space and has mass. Matter is composed of molecules that are made of atoms bonded together. Atoms consist of particles called electrons, protons, and neutrons. A simple description of an atom is the Rutherford model; although it is not the best model of an atom, it’s easy to use to explain what an atom looks like.

The center of the atom is the nucleus—this is where the protons and neutrons are located. The electron in the drawing is on the large circle, orbiting the nucleus. Protons and electrons have a charge associated with them: protons are positively charged, electrons negatively charged, and neutrons neutrally charged. Generally, most atoms have a neutral charge, which means that the charges from the electrons and protons cancel each other out. However, electrons can transfer to other atoms and cause a charge imbalance in the atoms involved in the transfer. In some materials (e.g., metals), the electrons are free to easily transfer to other atoms in the material; these materials are called conductors. In other materials (e.g., wood), the electrons are not able to move freely; these materials are called insulators.

“Every charged object has a certain potential,” [2] and “when two charged objects have different potentials, electrons tend to move” [2]—this movement of electrons is electricity.

DEFINITIONS

**Voltage:** a measurement of the potential difference and expressed in units of volts (V).

**Current:** the rate at which charge flows through a surface with an area. The units for current is amperes, or amps (A).

**Conductors:** materials in which electrons are free to move anywhere; they do conduct electricity.

**Conductivity:** a material property that represents a material’s ability to conduct electricity. The units are siemens (S).

**Semi-conductors:** materials in which electrons can move a little; they conduct less electricity than that of a conductor but more than that of an insulator.

**Insulators:** materials in which electrons are not able to move around; they do not conduct electricity.

**Resistivity:** a material property that represents a material resistance to electricity flowing; it is the inverse of conductivity. The units are ohm-meter (Ω*m).
**Resistance**: the ratio of potential drop to the current flow. Resistance is also determined by the shape and material. The units for resistance are ohms (Ω), and the two equations below can be used to calculate resistance.

\[ R = \frac{V}{I} \]  

(1)

Where,

- V = the potential drop, or voltage drop
- I = the current

\[ R = \rho \frac{L}{A} \]  

(2)

Where,

- \( \rho \) = resistivity
- L = length of the material
- A = cross-sectional area

**HOW LIGHT DEPENDENT RESISTORS WORK**

Light dependent resistors (LDRs) are also called “photoresistors” and sometimes “photocells.” The LDRs are made from a photoconductive material that has variable conductivity, depending on the amount of light it is exposed to. Because the conductivity increases with exposure, the resistivity decreases and in turn decreases the resistance according to Equation 2 in the previous section. LDRs are much more sensitive to light than other sensors but have a slow response time, which means they can pick up small changes in light but take a longer time to react to the light than other sensors do (their response time is on the order of a several milliseconds). For the purposes of this project, the slow response is acceptable because the sun does quickly change positions in the sky. If the LDR is exposed to intense sunlight or high humidity for long periods of time, its lifetime will decrease. More information about LDRs and other sensors can be found in “Applied Mechatronics” by A. Smaili and F. Mrad.

**BASIC CIRCUITS**

This section will briefly introduce circuit analysis. The first thing you need in a circuit is a source of current or voltage. An ideal voltage source produces a constant amount of voltage. An ideal current source provides a constant amount of current to the circuit. To keep it simple, this section will only go over circuits with a source and a resistor. The table below shows what the symbols in a circuit mean and look like.

<table>
<thead>
<tr>
<th>Name</th>
<th>Symbol</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ideal voltage source</td>
<td><img src="http://blanco.io/education/principles-of-ee-1/notes/" alt="Ideal voltage source symbol" /></td>
</tr>
</tbody>
</table>
Ohm’s Law: essentially the mathematical definition of resistance; a form of this equation can be seen in the definition section.

\[ V = I(R) \] (3)

Kirchoff’s Current Law (KCL): at any node of a circuit, the sum of the currents is zero.

Example: Using KCL, write an equation for the current of the figure above.

Solution:

Sign convention: The current going into the node will be considered positive, and the current leaving the node will be considered negative.

KCL states that the sum of the currents at any node must be zero. So,

\[ I_1 + (-I_2) + I_3 + I_4 + (-I_5) = 0 \]
Simplifying: \[ I_1 - I_2 + I_3 + I_4 - I_5 = 0 \]

Rearranging: \[ I_1 + I_3 + I_4 = I_2 + I_5 \]

Each of these three equations is correct.

**Kirchoff's Voltage Law (KVL):** the changes in potential (voltage) around any closed loop must equal zero.

**Example:** Each resistor has a voltage drop associated with it. For the circuit shown above, find the voltage drop across each resistor using a combination of Ohm’s Law and KVL.

Note: The arrow in the circuit is a loop and represents the current in the circuit; positive current usually flows from positive to negative. Because this is the case for V1, the current will be negative.

**Solution:**

Starting at V1, the equations are as follows.

\[ -V_1 + V_2 + V_3 + V_4 - V_5 = 0 \]

Substituting values: \[ -10 + V_2 + 34 + V_4 + V_5 = 0 \]

Using Ohm's Law (V = iR): \[ -10 + 2I + 34 + 4I + 6I = 0 \]

Solving for I: \[ 2I + 4I + 6I = 12I = -24 \]

\[ I = -2 \text{ A} \]

Do not worry about the negative sign; it is negative because the direction on the drawn loop was incorrect—in reality the current would be going in the opposite direction. Now Ohm’s Law will be used on each resistor to find each of their voltage drops.
\[ V_2 = (-2)2 = -4V \]
\[ V_4 = (-2)4 = -8V \]
\[ V_5 = (-2)6 = -12V \]

Again, these numbers are negative to correct the direction of the drawn loop. This KVL example is very simple; more complex examples can easily be found using Google’s search engine.

**Series connection:** when circuit elements have a common node with no other element connected to it.

**Parallel connection:** when the elements share two common nodes.

Simplify the circuit by finding the “equivalent resistance” so that a circuit with more than one resistor can be simplified to have one resistor.

**Example:** (This example is from [http://www.calvin.edu/~svelleest/circuitExamples/equRes/](http://www.calvin.edu/~svelleest/circuitExamples/equRes/))
Find the equivalent resistance for the circuit below, assuming the numbering is the same as their resistance value.

![Circuit Diagram](http://www.calvin.edu/~svleest/circuitExamples/equRes/)

**Solution:** Note that the resistors are numbered (corresponding to their value) and the nodes are lettered.

Starting from the drawn circuit’s top-right end and working to the bottom-right side, it can be seen that resistors 3, 6, and 9 in series.

\[ R_{eq1} = R_3 + R_6 + R_9 = 3 + 6 + 9 = 18 \]

Now we will redraw the circuit with the new equivalent value.

![Redrawn Circuit](http://www.calvin.edu/~svleest/circuitExamples/equRes/)

Now we can see that R5 and R18 are in parallel as they are both connected to nodes c and f.

\[
\frac{1}{R_{eq2}} = \frac{1}{R_5} + \frac{1}{R_{18}} = \frac{1}{5} + \frac{1}{18} = \frac{23}{90}
\]

\[ R_{eq2} = \frac{90}{23} = \sim3.9 \]
Now we will redraw the circuit again.

From this redrawn circuit, we can see that R2, R3.9, and R8 are in series.

\[ R_{eq3} = R_2 + R_{3.9} + R_8 = 2 + 3.9 + 8 = 13.9 \]

Redrawn picture,

Now we can see that R4 and R13.9 are in parallel.

\[
\frac{1}{R_{eq4}} = \frac{1}{R_4} + \frac{1}{R_{13.9}} = \frac{1}{4} + \frac{1}{13.9} = \sim0.3219
\]

\[ R_{eq4} = \frac{1}{0.3219} = \sim3.1 \]

Redrawn circuit,
From: http://www.calvin.edu/~svleest/circuitExamples/equRes/

\[ R_{eq5} = R_7 + R_{3.1} + R_1 = 7 + 3.1 + 1 = 11.1 \]

The final circuit with the final resistance value is shown below.

From: http://www.calvin.edu/~svleest/circuitExamples/equRes/

---

APPENDIX B: BASIC ARDUINO CODING

There are many coding languages, but Arduino uses a version of C++. In this language, everything must be initialized, as seen in the solar tracker code below. Some basic code structures will also be shown below. The information in this section—and much more information—can be found at https://www.arduino.cc/en/Tutorial/HomePage [8]

Note: A semicolon must be used to end every non-loop line of code.

**Comment lines**: lines of code that are used for explanations or comments but are not actually run as code.

// This is used for a single-line comment; any text after the double-slash will be a comment

/* This is a multiline comment; use this to comment out whole blocks of code

Use */ to end the commenting block

Commenting lines in and out can be very useful when trying to troubleshoot a code.

**For loop**: used to repeat a section of code for a specified number of iterations.
for (initialization; condition; increment) {
    //statement(s);
 }

A set of curly brackets must surround the section of code that is to be repeated.

**While loop:** will repeat a section of code when specific conditions are true.

```
while(expression){
    // statement(s)
}
```

}This code is useful when you want something to be repeated only if it reaches a certain condition or value. A set of curly brackets is also used to signify which section of the code is in the loop.

**If statements:** a line of code that says if this condition is true, then run the following section of code.

```
if (someVariable > 50)
{
    // do something here
}
```

}The “if statement” can be especially useful in systems that use sensors, like this project. Similar to loops, “if statements” must use curly brackets to specify which section of code they apply to.

**Else statements:** used in combination with “if statements.” They are essentially an add-on saying that if that condition is not met, run this section of code instead of doing nothing.

```
else
{
    // do something different here
}
```

}This code can also be used in combination with the next line of code. Again, similar to loops, “else statements” must use curly brackets to specify which section of code they apply to.

**Else if statements:** used in combination with “if statements.” They specify that if the conditions specified for the “if statement” are not met but the conditions specified by the “else if statement” are met, run the following section of code.

```
else if (pinFiveInput >= 1000)
{
    // do something here
}
```

} The advantage of this line of code is that you can do more than one conditional operation for the same variable. The curly brackets are used here as well.

The Arduino website is a great resource; it has many examples and tutorials as well as information on fun projects you can do with an Arduino.
SOLAR TRACKER CODE

/* SOLAR TRACKER 1.5
* by Mathias Leroy
* Novembre 2014
* modified 8/16/2016
*/

// INITIALIZATION

#include <SoftwareSerial.h>
//#include  <printf.h>
#include <Servo.h>
Servo servoH; // pin09 horizontal servo
Servo servoV; // pin10 vertical servo

// analog read pin connections
int analogPinTopLeft   = 0; // yellow orange
int analogPinTopRight  = 1; // blue green
int analogPinBottomRight = 2; // purple grey
int analogPinBottomLeft  = 3; // red brown

// PARAMETERS

int initAngleH =90;
int minAngleH  =10;
int maxAngleH  =160;
int initAngleV =20;
int minAngleV  =20;
int maxAngleV  =130;
int slowingDelay=5;
int sesitivityH=75;
int sesitivityV=75;
int stepH=5;
int stepV=5;

// VARIABLES
//----------------------------------------
int angleH =initAngleH;
int angleV =initAngleV;
char printLine [50];
int valueTopLeft = 0;
int valueTopRight = 0;
int valueBottomRight = 0;
int valueBottomLeft = 0;

// SETUP
//----------------------------------------
void setup()
{
    servoH.attach(9);
    servoH.write(initAngleH);
    servoV.attach(10);
    servoV.write(initAngleV);
    Serial.begin(9600);
    Serial.println("Ready! :-)" tentative code: Serial.println("Ready! :-)");
}

// LOOP

```c
// -----------------------------------------------
void loop()
{
  Serial.println("<<< Start Loop ---");

  // read values
  valueTopLeft     = analogRead(analogPinTopLeft);
  valueTopRight    = analogRead(analogPinTopRight);
  valueBottomRight = analogRead(analogPinBottomRight);
  valueBottomLeft  = analogRead(analogPinBottomLeft);

  // print values
  sprintf (printLine, "%d | %d 
", valueTopLeft, valueTopRight);
  Serial.print(printLine);
  sprintf (printLine, "%d | %d 
", valueBottomRight, valueBottomLeft);
  Serial.print(printLine);
  delay(300);

  // compute averages
  int averageTop = ( valueTopLeft + valueTopRight ) / 2;
  int averageRight = ( valueTopRight + valueBottomRight ) / 2;
  int averageBottom = ( valueBottomRight + valueBottomLeft ) / 2;
  int averageLeft = ( valueBottomLeft + valueTopLeft ) / 2;

  // print averages
  sprintf (printLine, "- %d - \n", averageTop);
  Serial.print(printLine);
  sprintf (printLine, "%d - %d \n", averageLeft, averageRight);
  Serial.print(printLine);
  sprintf (printLine, "- %d - \n", averageBottom);
  Serial.print(printLine);
  delay(slowingDelay);

  // Horizontal decision & action
```
Serial.println(angleH);
Serial.print("\n");

if ( (averageLeft-averageRight)>sesitivityH && (angleH-stepH)>minAngleH ) {
  // going left
  Serial.println("going left");
  Serial.print("\n");
  delay(slowingDelay);
  for (int i=0; i < stepH; i++){
    servoH.write(angleH--);
    delay(20);
  }
}

else if ( (averageRight-averageLeft)>sesitivityH && (angleH-stepH))
  servoH.write(angleH++);
  delay(20);

// vertical decision & action
Serial.println(angleV);
Serial.print("\n");

if ( (averageTop-averageBottom)>sesitivityV && (angleV-stepV)>minAngleV ) {
  // going up
  Serial.println("going up");
  Serial.print("\n");
  delay(slowingDelay);
  for (int i=0; i < stepV; i++){
    servoV.write(angleV--);
    delay(20);
  }
}
else if ((averageBottom-averageTop)>sesitivityV && (angleV+stepV))

    Serial.println("going down");

    servoV.write(angleV++);
    delay(20);

    Serial.println("--- End Loop >>>");


