Arduino Based Physics labs

Veronique Lankar PhD - lab assistant: Elias Benichou, NYCischool High School, NY

ISBN-13: 978-0-9896278-5-6

sketches: https://github.com/Louise555/arduino_based_labs

Prepared by Dr. Veronique Lankar 7/21/2016

Experiment : Acceleration due to gravity with a laser/photo transistor sensor.

PURPOSE:

Compute the acceleration due to gravity using a laser/phototransistor sensor

MATERIALS:

phototransistor sensor with 2 states (HIGH and LOW) from elab peers

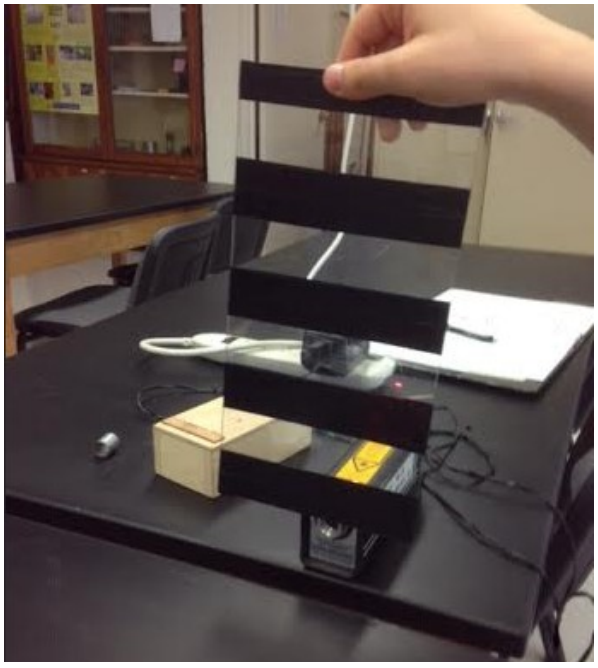
http://www.elabpeers.com/laser-detector.html

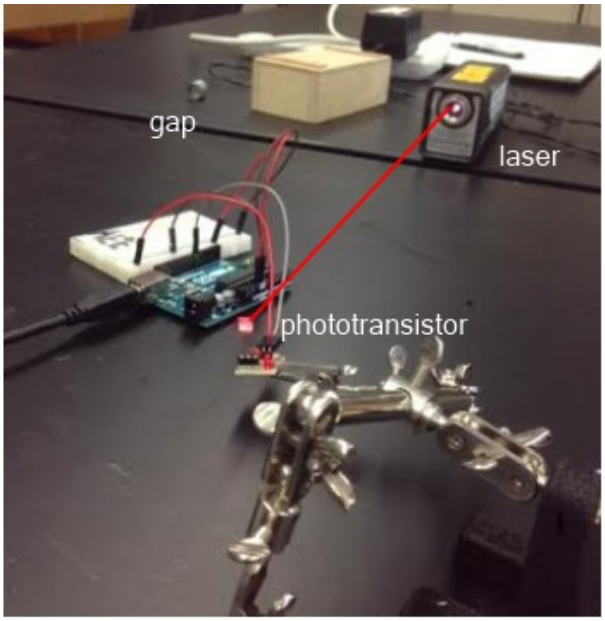
The laser can also be bought from elabpeers. We used a Helium-Neon laser block instead.

arduino , polycarbonate board 6in x 12 in (from ebay. Some vendors can cut the board for you).

Black electrical tape. Mask the board every 3.5cm as shown below. , helping hands (ebay).

Foam padding so the board does not break when it reaches the ground





To get started with Arduino (software / hardware) –

Skip this part if you already know your way around Arduino.

step1: download the IDE of Arduino

https://www.arduino.cc/en/Main/Software

step2: connect arduino to a USB port. Make sure your computer sees it. (open device manager to see the name of the port (com 1, com 2 etc..).

step3: open Arduino

From Tools select Arduino Uno

From Tools select Serial Port then select the right port. (the one Arduino is connected to).

Step4: Let's test the connection using the sketch blinking LED included in the library of Arduino. From File select Examples then select Basics then select Blink.

Upload the sketch in the microcontroller .

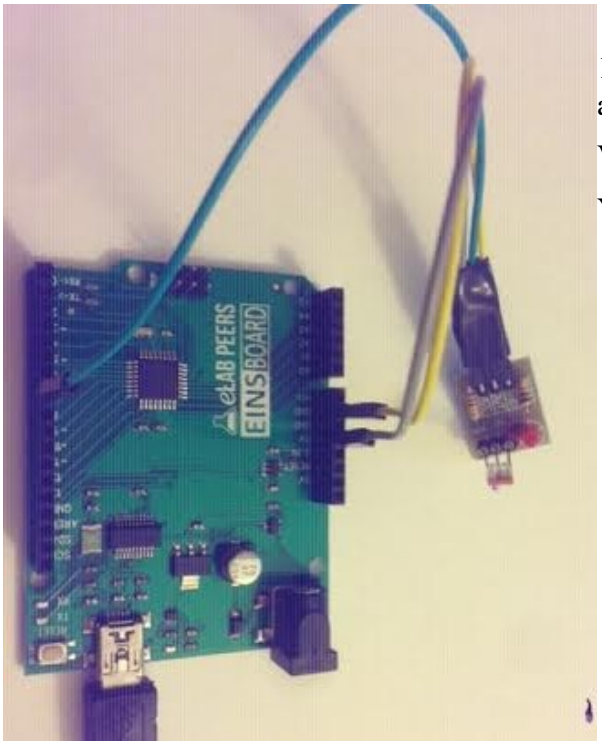
The small LED (included in the board) next to pin 13 should start blinking. You can change the rate at which it is blinking.

Step5: Now you can open the sketch *sketch_to_test_Laser_and_phototransistor* . Arduino will ask you to create a special folder for the sketch. Say yes and upload the sketc.

BACKGROUND:

The acceleration due to gravity is about g = 980cm/s/s . The goal of this lab is to estimate g.

PROCEDURE:

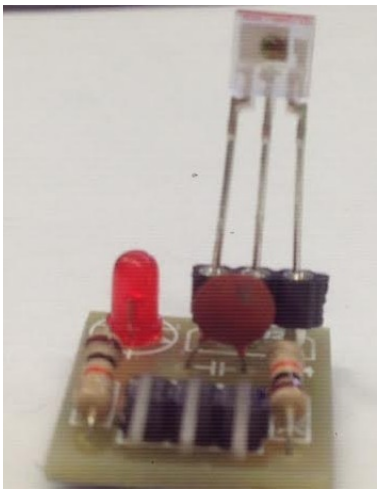


1) Connect the sensor (laser detector=phototransistor) to the arduino. The data wire (middle) is plugged in digital pin 7.

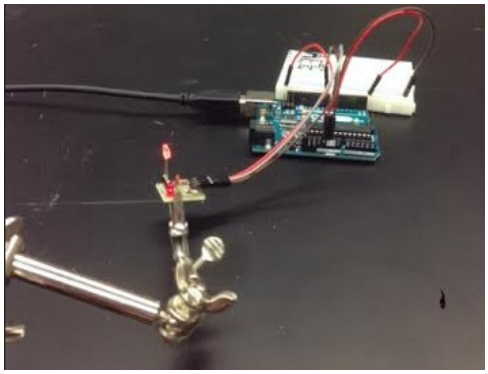
VDD in 5V. Gnd in ground.

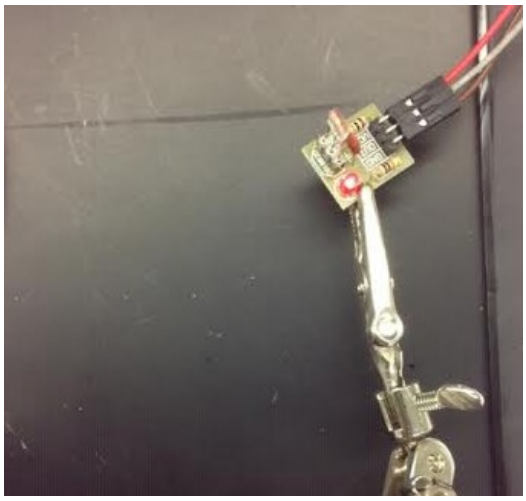
You can use the website below to see the same set up.

http://www.elabpeers.com/laser-detector.html



2) Set up the experiment as shown in the front page. Make sure there is a small gap between the table with the laser box and the table with the arduino/computer. The helping hands is holding the laser detector connected to arduino.





3) Aim the laser at the little dot and upload the sketch **s*ketch_to_test_Laser_and_phototransistor*** to test it. Open the Serial monitor (select the right baud rate 9600). You should get 1 when there is nothing between the laser and the detector and 0 when you put your hand in between. See front page.

4) upload the sketch ***sketch_laser_phototransistor*** . Open the Serial monitor and move your have in between the laser and the detector to test the code. Close the Serial monitor.

5) Open the Serial monitor again. Hold the plate vertically as shown in the cover so it is ready to fall between the 2 tables. Drop it. You should only use 8 lines of data and the line 0,0. Here is an example:

0	0
1	46
2	76
3	100
4	122
5	140
6	159
7	176
8	191

The second column has the time in ms.

6) Copy the data in a spreadsheet. (comma separated format). Build 2 new columns 3 and 4 : col 3 has the time in seconds (divide col 2 by 1000) and col 4 has the distance in cm: 3.5cm, 7cm, 10.5, 14cm… since the gaps are 3.5cm large and (you need to enter those numbers). It looks like this:

0	0
0.046	3.5
0.076	7
0.1	10.5
0.122	14
0.14	17.5
0.159	21
0.176	24.5
0.191	28

ANALYSIS

for instructors: the key is found on the last page.

1) make a scatter plot *distance (cm) versus time (s)* . Give a title to your graph and label the axis.

2) Insert a trend line: quadratic function also called a polynomial of degree 2. It is supposed to be a parabola. Display the equation on your graph. Save the graph to print it later.

3) the equation displayed is : y = ____ x^2 . Neglect the other terms. You know that in free-fall the equation *distance vs time* should be y=1/2 g x^2 with g the acceleration due to gravity = 980cms/s

so y= ½ (980) x^2 or **y = 490 x^2**

Compare with your equation. What is your experimental g ? (multiply by 2 the coefficient in front of the t^2)

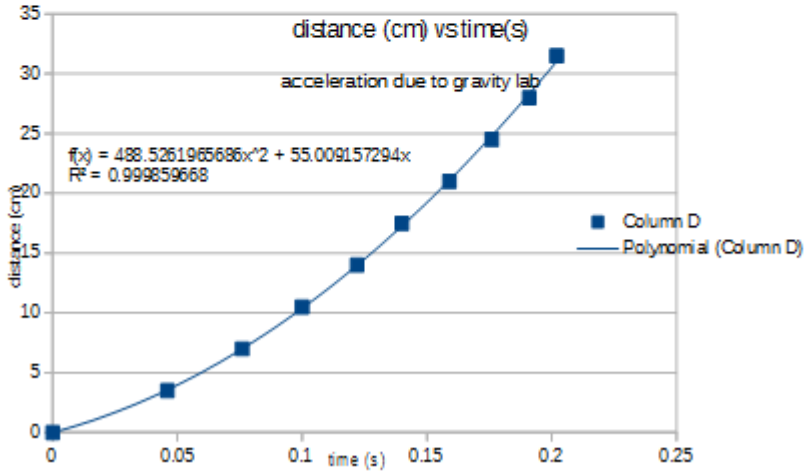
Compute the % error between your value and 980.

CONCLUSION

For instructors:

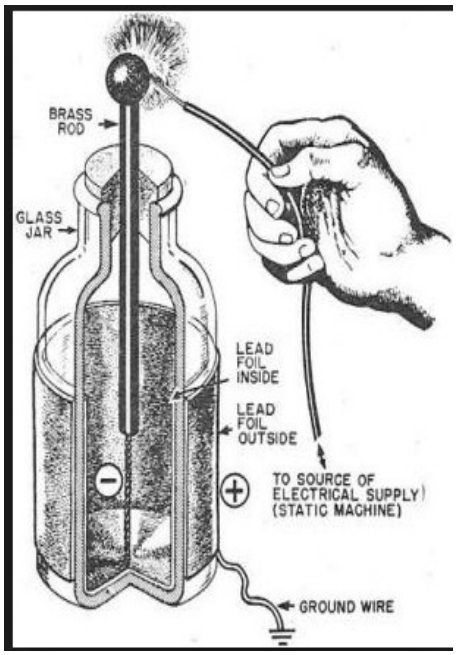
index measure	time (ms)	time(s)	distance(cm)
0	0	0	0
1	46	0.046	3.5
2	76	0.076	7
3	100	0.1	10.5
4	122	0.122	14
5	140	0.14	17.5
6	159	0.159	21
7	176	0.176	24.5
8	191	0.191	28
9	202	0.202	31.5

we find D=488.5 t^2
so we find g= 488.5 x 2 = 977cm/s/s
compared to 981cm/s/s
It's a 0.4% error



EXPERIMENT : CAPACITOR

MATERIAL : capacitor 1000 F, 4.5K resistor, arduino, breadboard



DATE_____

AUTHOR_____

BACKGROUND

The leyden jar was the first capacitor (see above picture). It was use to store charges and to produce electricity when necessary. A capacitor is the modern version. The simplest capacitor is 2 conductors (like 2 aluminum foil) separated by an insulating material called the dielectric. In electronics, capacitors like resistors are key ingredients of many electronics circuits.

Capacitors are characterized by their **capacitance C**. This quantity measures the ability to store charges. Capacitance is specified in **FARADS** (F). A 1 FARAD capacitor connected to a 1V supply will store $6.28\ 10^{18}$ electrons (number of electrons in 1 coulomb) !!

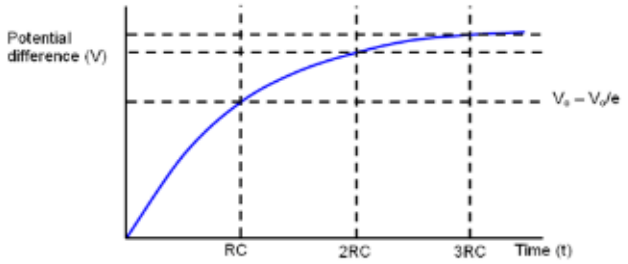
Capacitors used in electronics are usually very small and have very small capacitance computed in picofarads (10^{-12} F) or nanofards (10^{-9}F) or microfarads (10^{-6}) . **A capacitor can be very hazardous !!**
They can deliver a fatal current in a small amount of time. Never touch the 2 ends of the capacitors with your fingers, never touch the naked part of the alligators heads touching the terminals of a capacitor.

The capacitor can be charged by a battery. The charge of the capacitor is proportional to its voltage:

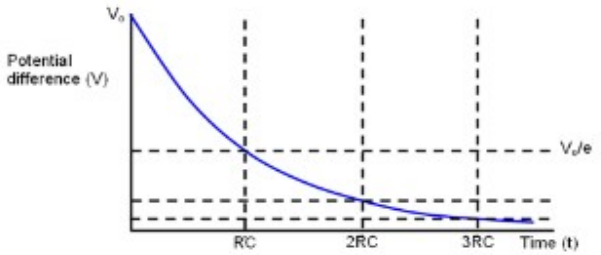
total charge (coulombs) in capacitor = capacitance (F) x voltage across (V)

or Q = V x C or **V = Q/C**

When you charge a capacitor, the voltage (or charge) increases with time. The graph of voltage (or charge) versus time is not linear but exponential. See the second diagram below. After a time T =RC (also noted tau τ) , the capacitor is 63% charged (or 0.63 x voltage of the battery V). T is a constant for a given capacitor and a given resistor. T is called the time constant.



Once charged, a capacitor can hold the charge for a long time. Some charge will leak out as time goes by. If you discharge a capacitor in another component, like a resistor, the discharge is exponential. The charge (voltage) will decrease exponentially. Again, the time for the capacitor to lose 63% of its charge is called the time constant T. In this lab we will study the discharge of a capacitor.



So after a time T, the capacitor loses 63 % of its charge, after another T (so 2T), it will loses another 63 % of the remaining charges, after another T (3T) it will lose another 63% of the remaining charge … so it goes very fast. (exponential decay).

If you discharge the capacitor in a resistor of resistance R then **T = RC**
so if you increase R or C or both, you increase the time it takes for the capacitor to charge or discharge.

In this lab, you will show that the discharge of a capacitor follows an exponential decay. You will experimentally compute RC (time T to lose 63% of the remaining charges) and you will compare this value to the theoretical value RC.

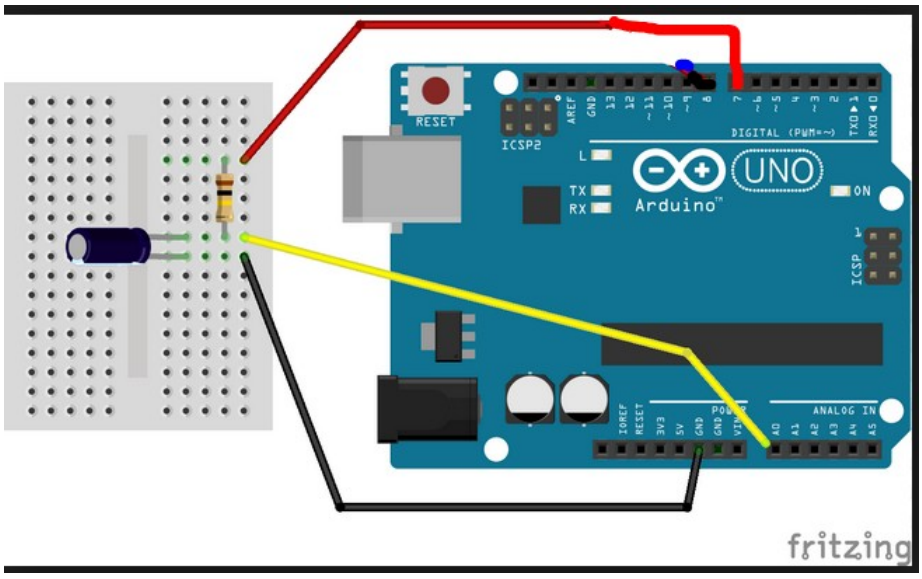
The mathematics equation that describe the discharge of a capacitor is :

$$V = Vo\ e^{(-t/T)}$$

V is the voltage across the capacitor. Vo is its voltage at t=0. T is the time constant.

PROCEDURE.

1) Build the circuitry. Pin 7 (5V or 0V) – resistor 4.5K capacitor 1000u – Gnd
pin A0 is used to read the voltage across the capacitor. See schematic.



2) Open the sketch for the capacitor. Connect the arduino to the computer. Upload the code in arduino.
Open the Serial monitor and select **115200** for the baud rate. . Pin 7 is 5V and the capacitor is charging. The voltage should first increase rapidly then slowly until it reaches about 4.9V. Pin 7 is switched to 0V and the capacitor is discharging. Wait for the full discharge.

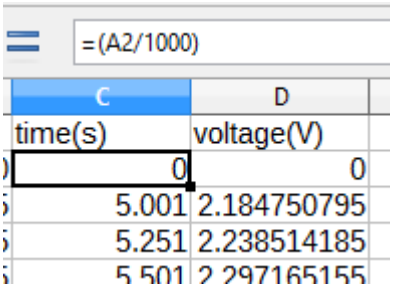
3) Copy and paste only the values of the charge of the capacitor in a spreadsheet f(col A and col B). Include the largest value. You should get values like:

A	B
time(ms)	voltage(V)
0	0
5001	2.184750795
5251	2.238514185
5501	2.297165155

4) Copy and paste the values for the discharges in col F and col G Include the largest value.

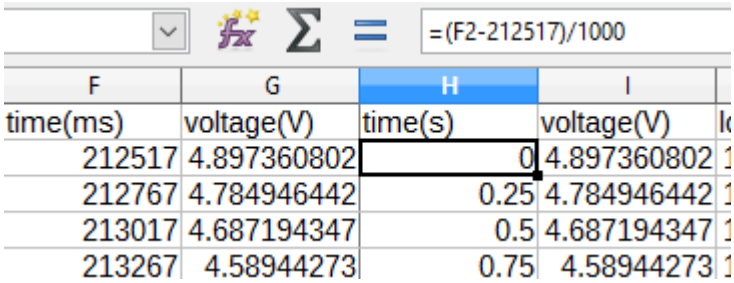
F	G	
ime(ms)	voltage(V)	t
212517	4.897360802	
212767	4.784946442	
213017	4.687194347	
213267	4.58944273	
213517	4.496578693	

ANALYSIS
for instructors: see the results on the last page.
1) Build a col C such as the times are in seconds: col C = col A/1000



copy col B in col D

2) Likewise Build col H using col F such as the time starts at 0 and are in seconds:



copy col G in col I

3) Make a scatter plot of *voltage(V) vs time(s)* for the charge of the capacitor. (so use Col C and Col D)

Save the plot and attach it to your lab report. Is the graph consistent with the theory ?

4) Make a scatter plot of the discharge of the capacitor. Use col H and I.

5) insert a trend line. Select an exponential and force the y-intercept to the maximum value. (4.9V in my case), Display the equation on the graph.

V = _____

6) according to the equation what is your experimental time constant tau ?

(remember in theory V = 5 exp(- t/tau) so if you get 5 exp(-0.22 t) that means 1/tau = 0.22 and you can solve for tau)

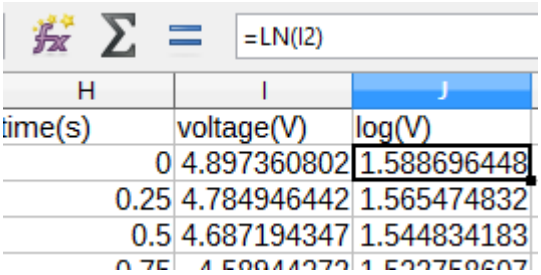
7) tau is the time constant and its value in theory is R C = 4.5s in our case.
Are the values close?

8) Using the value find the time for which voltage = 0.37 (5V) (or 37% of 5V) This is also the time constant.

tau = _____

Compute the % error (compare to 10s).

9) extra: take the natural log (ln) of the voltage and save in col J.

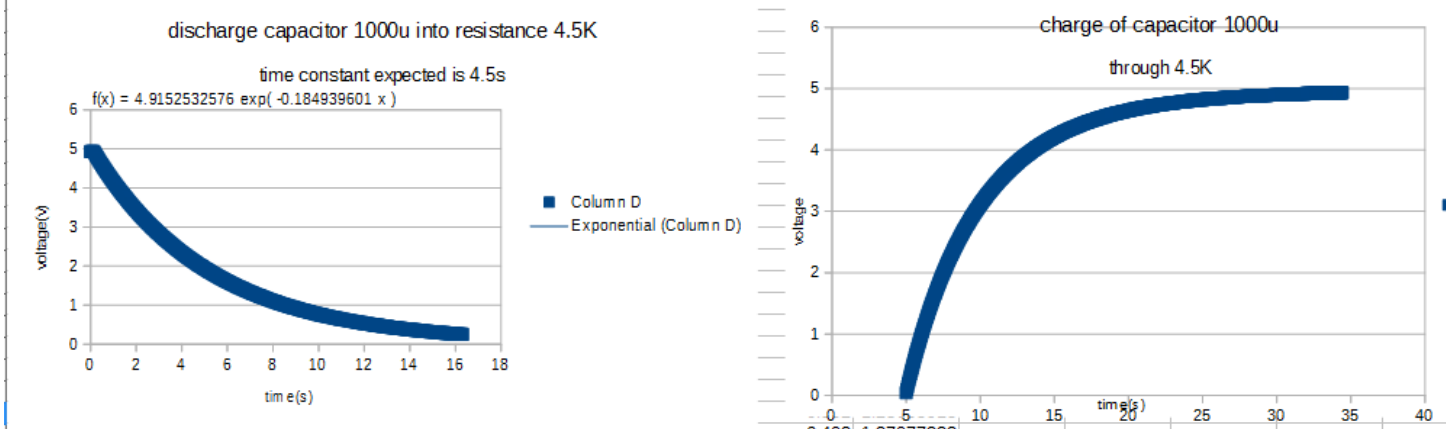


10) Plot ln(v) vs time (col H and col J). Explain why you get a line. Find the equation of the best fit line.

What does the slope represent?

CONCLUSION

time (ms)	voltage	time (s)	voltage		time(ms)	voltage	time (s)	voltage
32264	4.93157387	0	4.93157387					
32314	4.93157387	0.05	4.93157387		5001	0.04398827	5.001	0.04398827
32364	4.93157387	0.1	4.93157387		5051	0.09775171	5.051	0.09775171
32414	4.93157387	0.15	4.93157387		5101	0.14662756	5.101	0.14662756
32464	4.87781048	0.2	4.87781048		5151	0.20039101	5.151	0.20039101
32514	4.82893467	0.25	4.82893467		5201	0.24926686	5.201	0.24926686
32564	4.78005886	0.3	4.78005886		5251	0.29814272	5.251	0.29814272
32614	4.73118258	0.35	4.73118258		5301	0.34701858	5.301	0.34701858
32664	4.68230677	0.4	4.68230677		5351	0.39589443	5.351	0.39589443



33714	3.8123167	1.45	3.8123167		6402	1.27077222	6.402	1.27077222
33764	3.77810359	1.5	3.77810359		6452	1.30987287	6.452	1.30987287

Prepared by Veronique Lankar 7/21/16 with the assistance of Elias Benichou (NYCischool)

Experiment : car on an inclined plane – constant acceleration

PURPOSE:

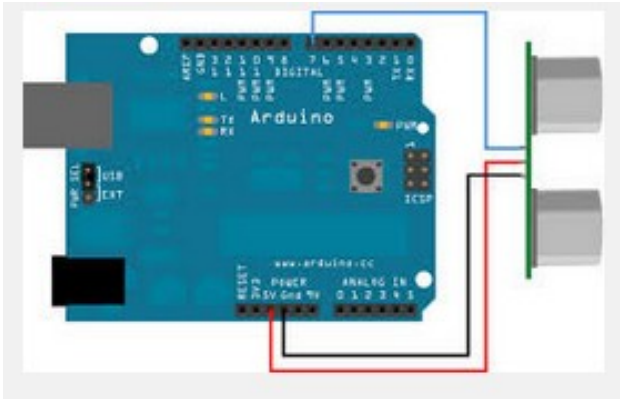
1) understand how to find the acceleration of a car using the graph distance vs time

2) understand how to find the speed using the graph distance vs time

3) show that the acceleration along an inclined plane is constant.

MATERIALS:

inclined plane (2m long), helping hands, car, arduino with USB cable, Ping Ultrasonic Range Finder sensor, sketch to upload.



BACKGROUND:

The force that pulls an sliding object along an inclined plane is F = g sin(A) – frictional force

with g = 9.81m/s/s and A the angle of the inclined plane.

Newton's second law gives us the acceleration a of the sliding object. a = F/m with m the mass of the object.

So **a = g sin(A) – (frictional force) / m**

The acceleration is constant as long as the friction is constant. A car is not really a sliding object, but it is a good approximation as long as the wheels have a small mass compared to the car.

Since the acceleration is constant then the distance covered along the inclined plane (x-axis) is given by:

$x(t) = 0.5\ a\ t^2$ if the This graph is a parabola

If we graph x(t) as a function of the time t and if we use the spreadsheet to fit the scatter plot then we can find the acceleration of the car. We can also compute the slopes of the graph at different time. If we plot the speed versus time, we should get a straight line. **V(t) = a t** . The slope of the graph should be the acceleration a.

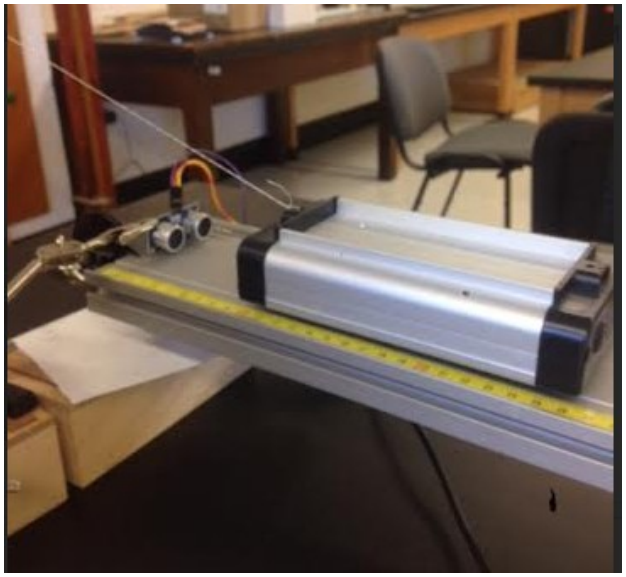
Another way to compute the acceleration is : **a = 2 x (average speed) / time**

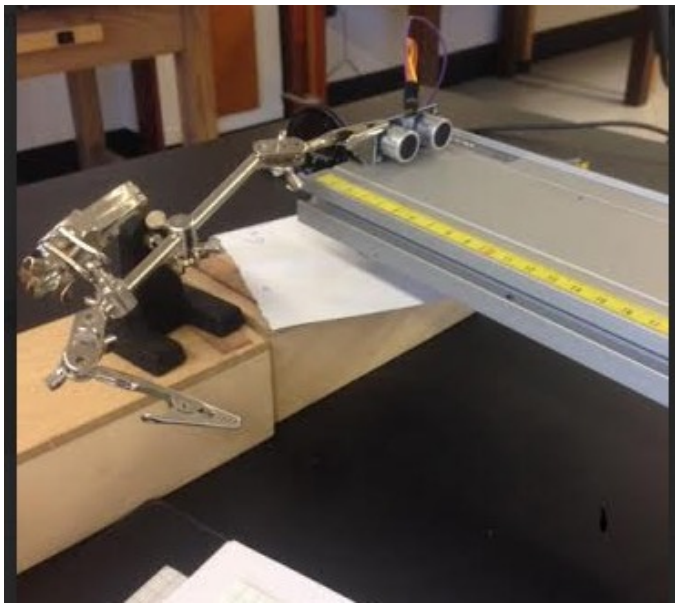
average speed = (distance covered)/ time

PROCEDURE

Make sure not to bump into the ultrasound and not to press on it. It is very fragile.

1) using the helping hands hold the sensor as shown. Connect the sensor to arduino (use pin 7 for the data wire) and arduino to the computer with the USB cord. Upload the sketch into the arduino.





2) The sensor is placed at 10 cm from the car. Open the serial monitor to make sure you are recording data. And that the data make sense. You should read about 10cm.

3) Hold the car with a string. Place a heavy block at a given distance down the plane. The block will stop the car. You will unplug the cable as soon as the car reaches the block.

4) Open the Serial monitor and let go with the car. As soon as the car bumps into the block, unplug the monitor.

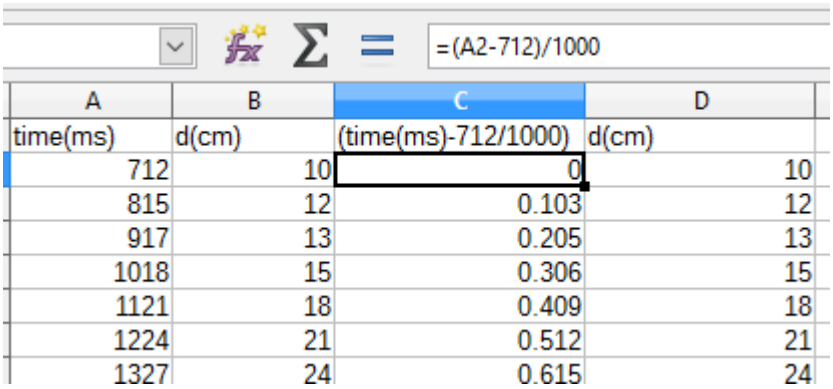
5) copy and paste the measurements into a spreadsheet but don't include the time after it hits the block. Make sure the distance increases. Include 3 points before it was moving. So it is clear that the car starts with a speed of 0.

ANALYSIS

instructors: the results are on the last page.

Now you have 2 columns (A and B) in your spreadsheet. One for the time in milliseconds (ms) and one for the distance in cm. Include a label at the top of each column. (see below)

1) Make a new column C . In this column the time starts at 0 and is in seconds. Here is an example:



A	B	C	D
time(ms)	d(cm)	(time(ms)-712/1000)	d(cm)
712	10	0	10
815	12	0.103	12
917	13	0.205	13
1018	15	0.306	15
1121	18	0.409	18
1224	21	0.512	21
1327	24	0.615	24

column C = (column A – 712)/1000.

2) copy column B in Column D

3) Make sure all the points are consistent. The distance should increase.

Compute the total distance covered d = _____ cm

the total time is _____ s

So the average speed = d/t = _____ cm/s

the acceleration is = 2 (average speed) / (total time) so a = _____ cm/s/s

4) make a scatter plot *distance (cm) versus time (s)*. So use column C and column D. You should get a parabola.

5) Fit a quadratic equation (exponential of degree 2) using the spreadsheet.

You get x = _____ . The theoretical equation is $x = xo + 0.5\ a\ t^2$

By comparing the 2 equations you can find a = _____ cm/s/s .

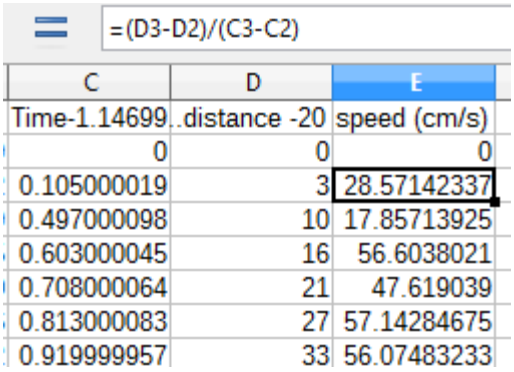
Is this number close to the one found in 3) ? Compute the % error.

6) We can compute the speed (approximation) between 2 points by using the equation:

speed = (x2 – x1) / (t2 - t1). The speed is the slope of the parabola.

Build a column E . The first cell has a 0 (initial speed) then fill the cells using the equation:

speed=x2-x1/t2-t1. The cells are populated with the speed between 2 consecutive points. For example:



C	D	E
Time-1.14699	distance -20	speed (cm/s)
0	0	0
0.105000019	3	28.57142337
0.497000098	10	17.85713925
0.603000045	16	56.6038021
0.708000064	21	47.619039
0.813000083	27	57.14284675
0.919999957	33	56.07483233

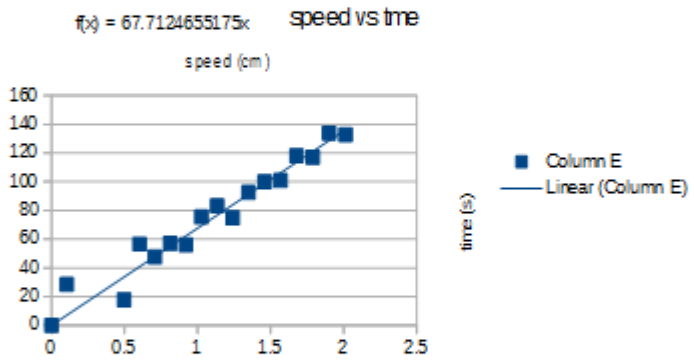
7) make a scatter plot of the *speed vs time*. You should get a line. Fit a best fit line and record the equation here:

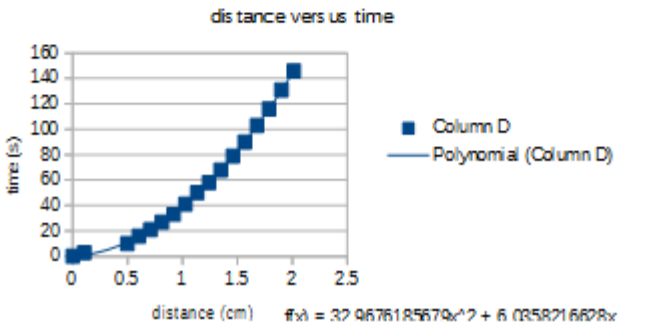
v(t) = _____. What is the slope? _____ Do you get the same acceleration as before ?

CONCLUSION

Sheet1

time(s)	distance(cm)	Time-1.14699.	Distance -19	speed (cm/s)
1.146999955	19	0	0	0
1.251999974	22	0.1050000191	3	28.57142337
1.644000053	29	0.4970000983	10	17.85713925
1.75	35	0.6030000449	16	56.6038021
1.855000019	40	0.7080000639	21	47.619039
1.960000038	46	0.813000083	27	57.14284675
2.066999912	52	0.9199999571	33	56.07483233
2.173000097	60	1.0260001421	41	75.47156639
2.280999899	69	1.1339999438	50	83.33348634
2.388000011	77	1.2410000563	58	74.76627653
2.496000051	87	1.3490000963	68	92.5925583
2.605999947	98	1.4589999914	79	100.0000954
2.714999914	109	1.567999959	90	100.9174612
2.825	122	1.6780000449	103	118.1817259
2.936000061	135	1.7890001059	116	117.1170528
3.048000097	150	1.9010001421	131	133.9285281



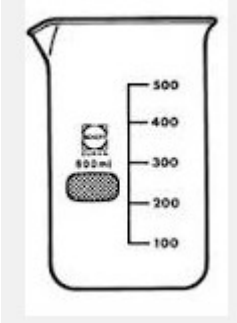


Prepared by Veronique Lankar 7/21/16

EXPERIMENT : specific heat of aluminum

MATERIAL : piece of aluminum, glass beaker (about 400ml) with hot plate, another beaker 400ml , digital scale, temperature sensor DS18B20 water proof from adafruit. String. Regular thermometer.

Purpose: To measure the specific heat capacity of aluminum (0.215 cal/g C)



Background

The amount of thermal energy that a single gram of a specific material must absorb in order to change its temperature by one degree is the material's specific heat capacity, or specific heat. The specific heat of water is a standard to which specific heats of other substances are compared.

The specific heat of water is 1cal/g C . So it takes 1cal (4.184 joules) to raise the temperature of 1 g of water by 1 C. Aluminum has a specific heat of 0.215 cal/g C. Copper has a specific heat of 0.092 . That means that it is easier to raise the temperature of copper than the same mass of water. For the same same heat provided.

The greater the specific heat, the less temperature will rise when a given heat energy is absorbed.
As the specific value decreases, the ability to deliver heat to a cooler object increases (better conductivity).

When a hot piece of metal is submerged in water, the heat lost by the metal Q is absorbed by the water. The temperature of the metal will drop by ΔTm. The temperature of the water will rise by ΔTw.

The heat absorbed by the water is given by the equation : $Q_w = m_w C_w \Delta T_w$

m_w is the mass of the water accepting the metal. C_w is 1 cal/g C.

The heat provided by the metal is given by the equation: $Q_m = m_m C_m \Delta T_m$
with m_m is the mass of aluminum
C_m is the specific heat of aluminum 0.215 cal/gC

Because of the conservation of energy we have Qm = - Qw
The heat lost by the metal = heat gained by the water.

$$m_m C_m \Delta T_m = -(m_w C_w \Delta T_w)$$

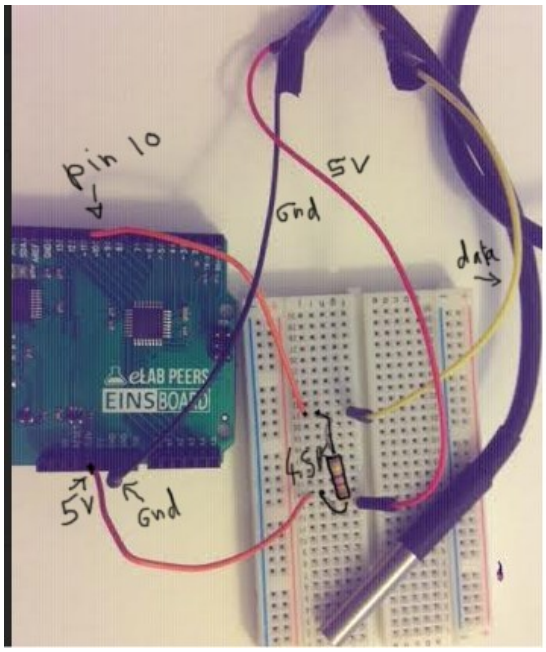
m_m = mass of metal
C_m = specific heat of metal
ΔT_m = change of temperature of metal
m_w = mass of water
C_w = specific heat of water
ΔT_w = change of temperature of water

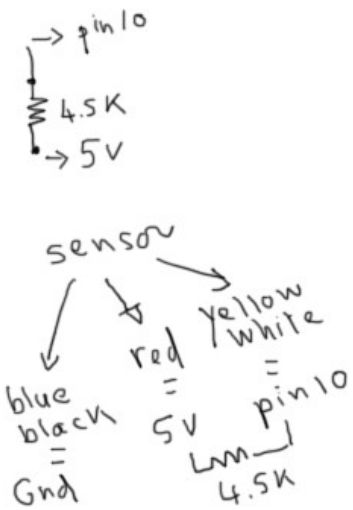
With C_w=1. The negative sign will disappear because the change of temperature of the metal is negative.

The goal of the lab is to show that C_m = 0.2 about.

PROCEDURE

1) Connect the sensor to arduino . The red wire is for 5V . The black or blue wire is for Gnd. The yellow or white wire is for the data wire. You also need a 4.5k resistor between 5V and data wire. (pull-up resistor).





2) connect the USB wire to the laptop. Upload the sketch in the chip. Open the Serial monitor and check if the sensor is working properly. Close it.

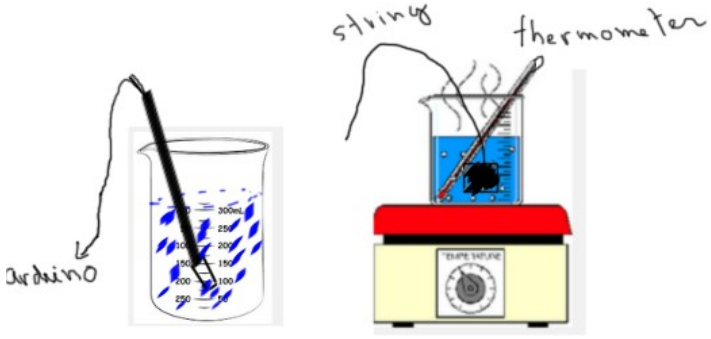
3) Use a 400 ml beaker and fill it with 300mL of cold water. 300ml of water has a mass of 300g.
Report m_w is the table below.

4) Measure the mass of the piece of aluminum m_m (g) and report in the table below. Attach a string to the piece so you can dip it in hot water. See front page.

5) Place the temperature sensor in the cold water and leave it here.

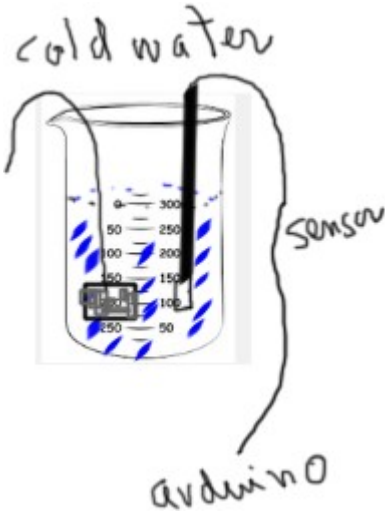
6) Place another beaker on the hot plate (see below). Place the thermometer inside (not the sensor). Submerged the metal in the water. Switch on the plate to heat the water up. Be careful not to burn yourself. You will still gently the water time to time wand wait for the temperature to reach 90C.

In the mean time. Open the Serial monitor and record the temperature of the cold water as T1w in the table below. Leave the serial monitor open.



7) When the thermometer reads 90C, quickly and gently move the metal in the cold water. Observe the Serial monitor. The temperature of the water is rising. **Do not close the Serial monitor from now one.**

8) When the temperature stops rising and reaches a maximum, wait to get a few more values and unplug the USB cord.



9) In the table below record the 90C as the initial temperature of the metal T1m.

10) In the table below record the maximum temperature recorded by the sensor. The maximum temperature reached by the water. this is T2w and T2m since at equilibrium they reach the same temperature.

11) Copy and paste the data from the serial monitor in a spreadsheet (comma separated data).

Mass of aluminum (g)	m_m=
Mass of water (g)	m_w=
Initial temperature of metal	T1m =
Final temperature of metal	T2m =
Change of temperature metal	ΔTm=T1m-T2m=
Initial temperature of water	T1w=
Final temperature of water	T2w=
Change temperature of water	ΔTw=T2w-T1w=
Heat gained by water Qw	Qw=ΔTw . m_w=
Heat lost by metal Qm	Qm = Qw=

ANALYSIS

instructors: results on the last page

1) Complete the table and compute the change of temperature and the heat exchanged.

2) $Qm = Cm \cdot \Delta tm \cdot m_m$

$Qm = Qw$

Use the equations to compute the specific heat of aluminium. Cm.
The theoretical value is 0.215 cal/ C g.

Compute the % error

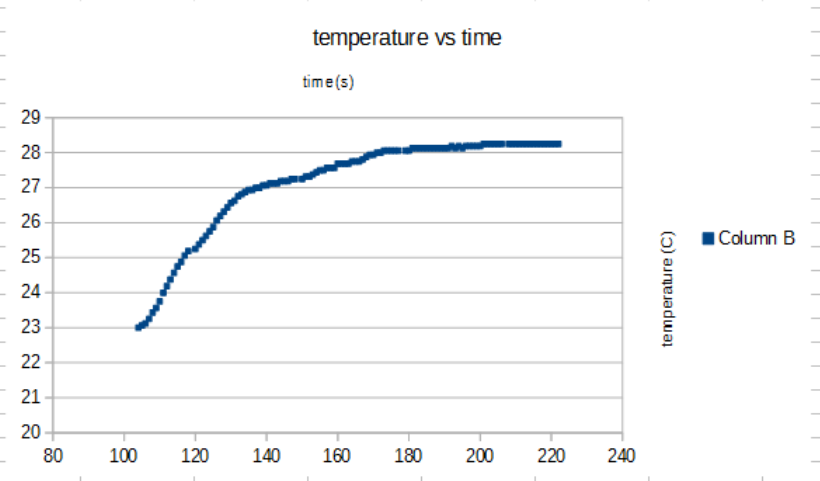
3) In the spreadsheet build 2 new columns.

One for the time in seconds and one for the temperature.
Make a scatter plot temperature vs time.

Discuss the rate of the slope.

CONCLUSION

time(s)	Temp in C
104	23
105	23.0625
106	23.125
107	23.25
108	23.4375
109	23.5625
110	23.75
111	24
112	24.1875
113	24.375
114	24.5625
115	24.75
116	24.875
117	25.0625
118	25.1875
120	25.25
121	25.375
122	25.5
123	25.625
124	25.75
125	25.875
126	26.0625
127	26.1875
128	26.3125
129	26.4375
130	26.5625



mass of Al block = 110.4g
mass of water = 272.5
temperature initial of water = 23
temperature final of water = 28.25
hot block temperature = 90 C

c x mass block x (90-28.25) = mass water x (28.25-23)

C=0.21

EXPERIMENT : inductor

MATERIAL : inductor pasco 3200 turns with an iron/ferrite core (total L inductance is 0.7H) in my case). A 10 ohms resistor. Connection wire with alligator clips for the inductor. Arduino. Note that without the core, the inductance falls to about 0.2H. So you need a core to boost the inductance. I used a ferrite rod.

Purpose: to show the charge and discharge of an inductor. To show that the time constant is L/R with L the inductance. Here the inductor + iron core has an inductance of about 0.7H in my case. The total resistance is 155 ohms. The inductor has a resistance of about 145 ohms.

https://www.pasco.com/prodGroups/coils-and-cores/index.cfm#orderItem_SF-8613

Background

An inductor charges and discharged in a DC circuit very much like a capacitor. In the case of an inductor, the magnetic field expands when we close the circuit and collapses when we open the circuit.

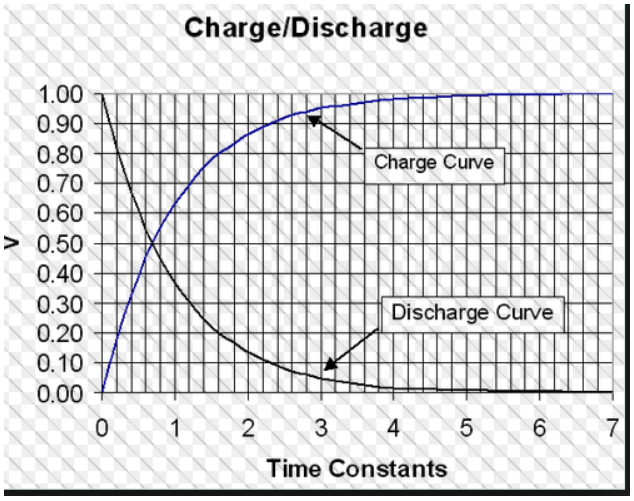
An inductor opposes the change of current in the circuit and the current will not build up instantaneously or will not go to zero instantaneously. During the charge the current will increase until it reaches a maximum Imax = 5 volts / (145+10).ohms in our case.

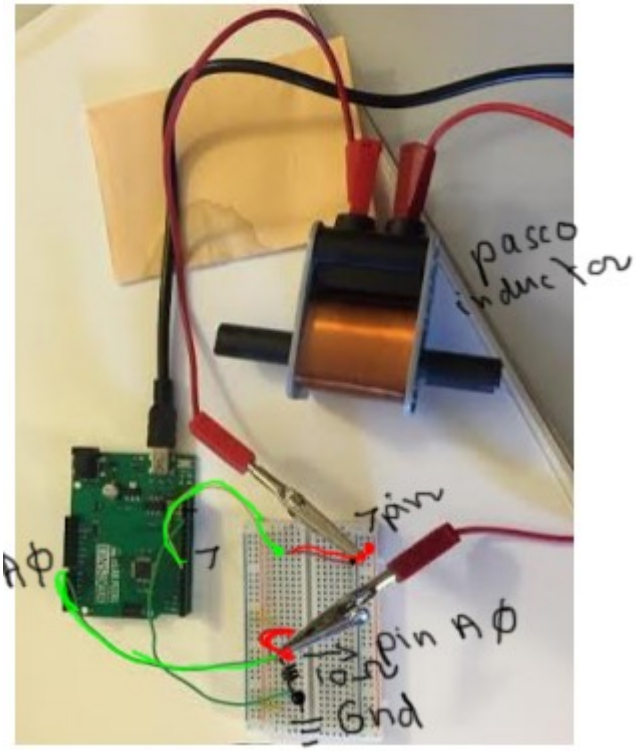
The inductor is not ideal and has a resistance 145 ohms. We add a 10 ohms resistor in series.
When we open the circuit, the current decreases exponentially (exponential decay).

I = Imax exp(- t/tau) with tau = L/(R total) R total= 145+10ohms and Imax = 5 /(145+10)

The voltage across a resistor is proportional to the current flowing through it. V = R I (Ohm's law) so we will measure the voltage across the resistor 10 ohms so we expect an exponential decay during the discharge

V = V max exp(- t/tau) with tau = L/R with Vmax = 10xImax.



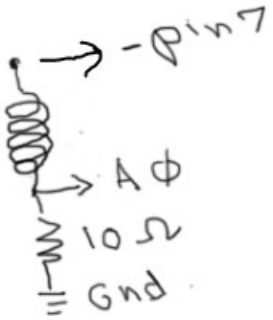


PROCEDURE

1) build the circuit :pin 7 – inductor – resistor 10ohms – Gnd

The pin A0 will measure the voltage across the resistor.

Pin 7 is a digital pin. It will be 5V when charging the inductor. 0V for the discharge.



The inductor has an iron core to boost its inductance to about 0.7H.

2) Connect arduino to computer with an USB cable and upload the sketch inductor into the arduino.

3) Open the Serial monitor and select the **baud rate to 115200**. Close and open again.

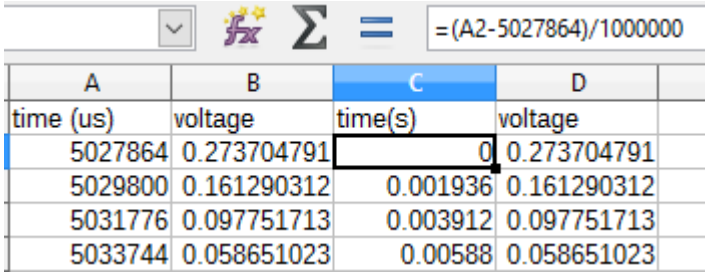
4) the inductor should charge and discharge.

5) The charge should look like:

```
5000288,0.0048875851
5002224,0.1075268864
5004200,0.1710654973
5006168,0.2052785873
5008120,0.2297165155
5010096,0.2443792819
5012064,0.2541544437
5014024,0.2639296054
5015984,0.2639296054
```

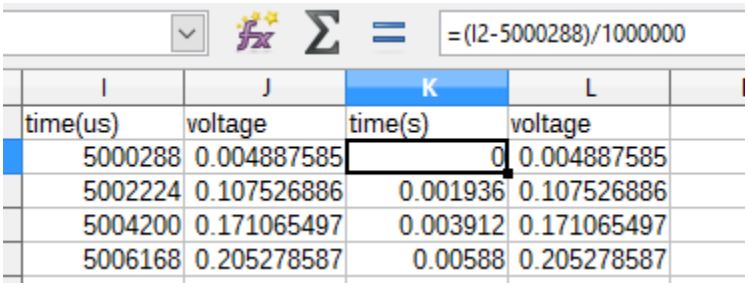
The time is in microseconds and the voltage across the resistor in volts.

6) copy and paste the discharge in the spreadsheet (col A and B) and make 2 new columns (C and D) with the time starting at zero and the time in seconds. (divide by 1000000)



A	B	C	D
time (us)	voltage	time(s)	voltage
5027864	0.273704791	0	0.273704791
5029800	0.161290312	0.001936	0.161290312
5031776	0.097751713	0.003912	0.097751713
5033744	0.058651023	0.00588	0.058651023

7) copy the charge in the spreadsheet the same way.



I	J	K	L
time(us)	voltage	time(s)	voltage
5000288	0.004887585	0	0.004887585
5002224	0.107526886	0.001936	0.107526886
5004200	0.171065497	0.003912	0.171065497
5006168	0.205278587	0.00588	0.205278587

ANALYSIS

instructors: results on the last page.

1) what is the theoretical time constant L/Rmax ?
(with L = 0.7 and Rmax =155 about. You can verify by measuring the resistance of the inductor and the inductance with the core. The resistance of the inductor is about 145 ohms and we need to add 10 ohms for the resistor)

L/Rmax= _____s

2) make a graph for the charge of the inductor.

3) make a graph for the discharge of the inductor and fit the graph with an exponential decay.

The equation is : _____ In theory Vmax exp(-t/tau).

So the time constant is _____ s

Compare to the value found 1) Compute the % error.

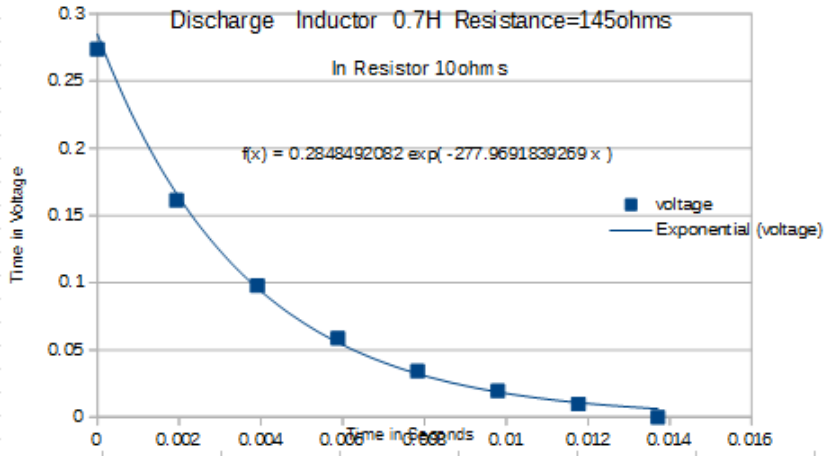
4) The theoretical Vmax = 10 x I max with Imax = 5/(145+10). 10 is the resistance of the resistor we are using.

Do you get a Vmax close to this value ? (according to equation)

Compute the % error.

time (us)	voltage	time(s)	voltage
5027864	0.273704791	0	0.273704791
5029800	0.161290312	0.001936	0.161290312
5031776	0.097751713	0.003912	0.097751713
5033744	0.058651023	0.00588	0.058651023
5035696	0.034213099	0.007832	0.034213099
5037656	0.019550342	0.009792	0.019550342
5039628	0.00977517	0.011764	0.00977517
5041572	0	0.013708	0

discharge

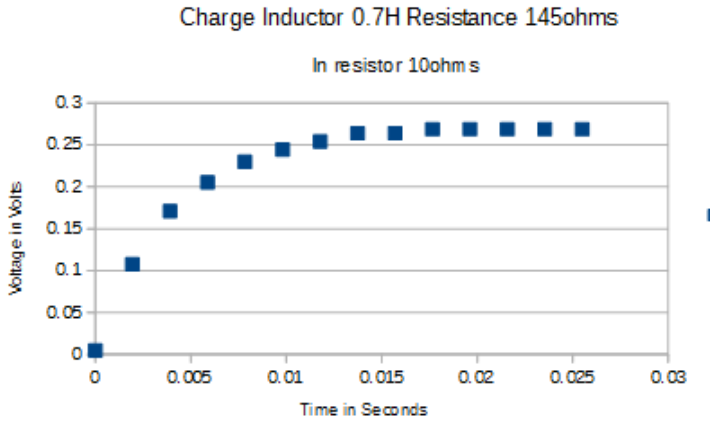


arduino output:

```
5000288,0.0048875851
5002224,0.1075268864
5004200,0.1710654973
```

time(us)	voltage	time(s)	voltage
5000288	0.004887585	0	0.004887585
5002224	0.107526886	0.001936	0.107526886
5004200	0.171065497	0.003912	0.171065497
5006168	0.205278587	0.00588	0.205278587
5008120	0.229716516	0.007832	0.229716516
5010096	0.244379282	0.009808	0.244379282
5012064	0.254154444	0.011776	0.254154444
5014024	0.263929605	0.013736	0.263929605
5015984	0.263929605	0.015696	0.263929605
5017944	0.26881721	0.017656	0.26881721
5019904	0.26881721	0.019616	0.26881721
5021864	0.26881721	0.021576	0.26881721
5023824	0.26881721	0.023536	0.26881721
5025784	0.26881721	0.025496	0.26881721
5027864	0.273704791	0.027576	0.273704791

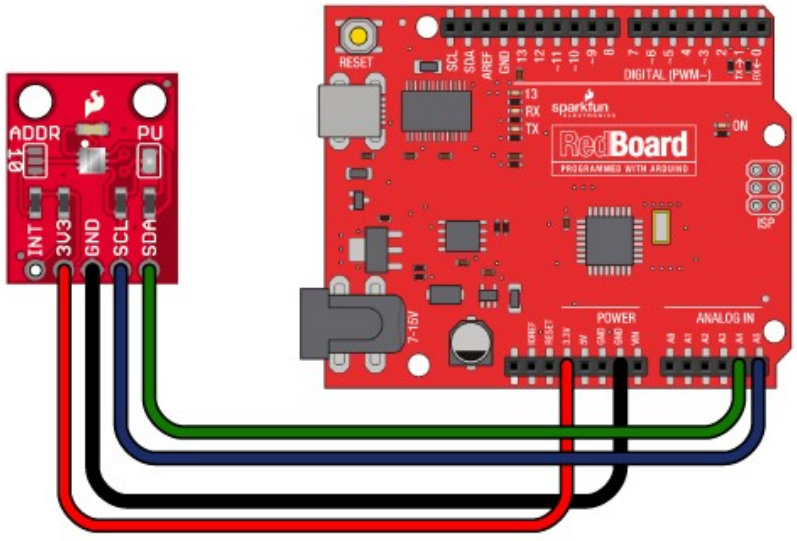
charge



Prepared by Dr. Veronique Lankar 7/21/2016

EXPERIMENT: THE INVERSE-SQUARE LAW

Materials: ;Light sensor TSL2561 from sparkfun (sensor that simulates the human eye). Incandescent light bulb. Meter tape.



https://learn.sparkfun.com/tutorials/tsl2561-luminosity-sensor-hookup-guide
to learn more about the sensors, to download the libraries the sensor needs to function.
The sketch needs two libraries:
#include <SparkFunTSL2561.h>
#include <Wire.h>

DATE_____

AUTHOR_____

PARTNER_____

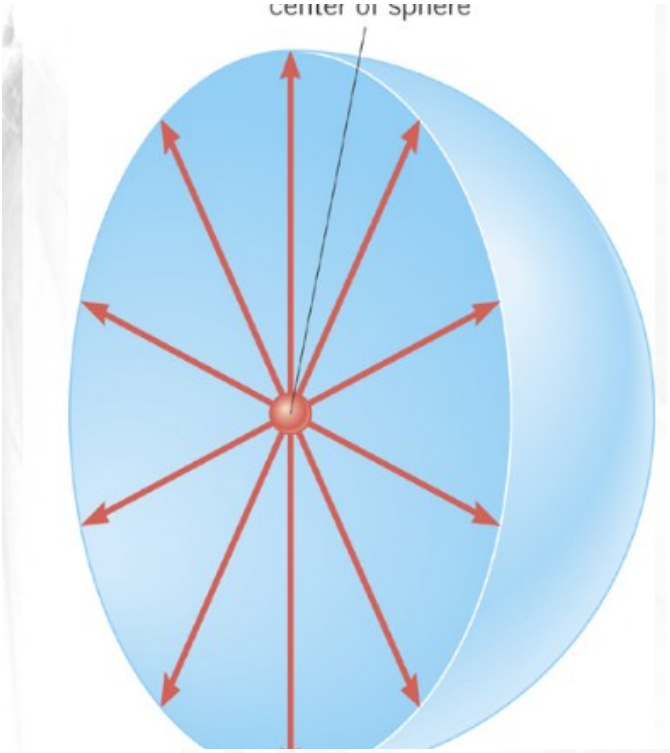
PARTNER_____

BACKGROUND

The inverse-square law relates the apparent brightness b, of a star, with its luminosity L :

$$b = \frac{L}{4\pi d^2}$$

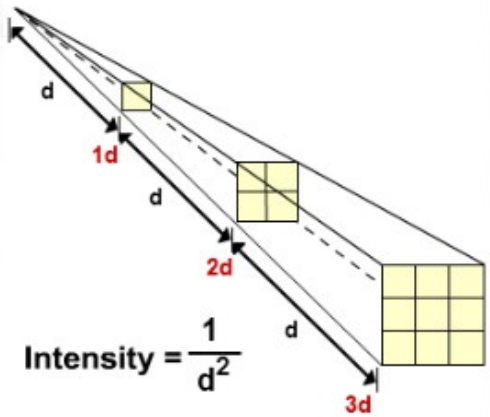
The **luminosity L** is the **power** of the star = **how much energy is produced per second**. The unit is the Watt (W). b is the apparent **brightness** of the star at a distance d (m). The unit is watts per square meters (W/m^2). The apparent brightness b drops off as the inverse-square of the distance to the star.

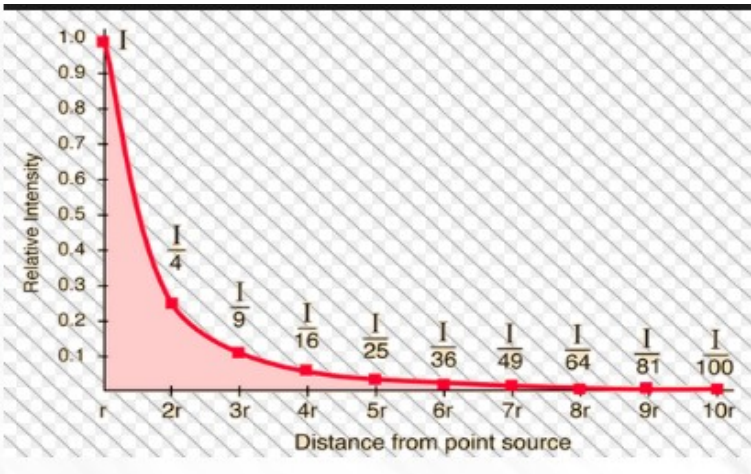


The total amount of energy L (luminosity or intrinsic brightness) flowing through the sphere per second is constant but the energy per unit area b per second (apparent brightness) decreases with distance d by the same amount factor that the area increases.

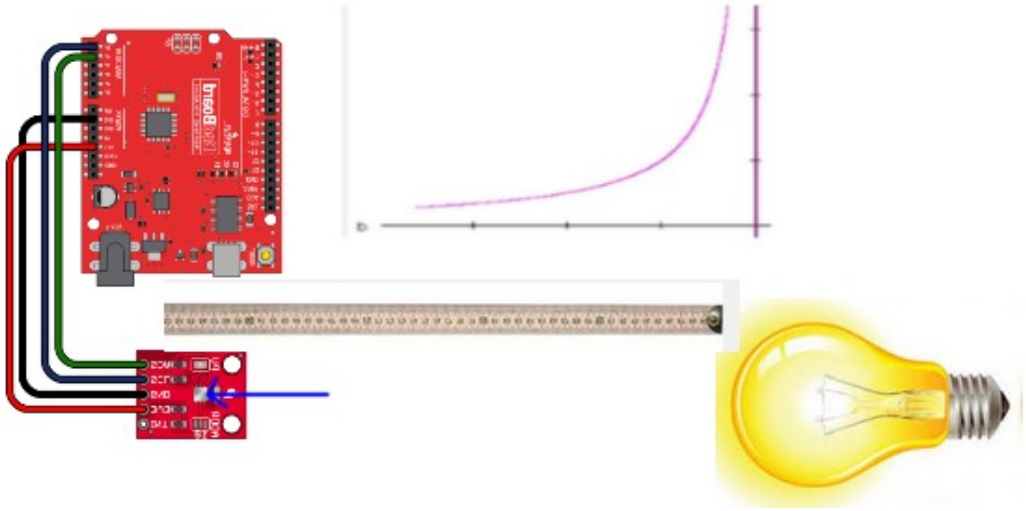
The red arrow is the distance d between the star and a detector. THIS IS THE INVERSE SQUARE LAW

SAME LAW APPLIES TO :
GRAVITY, ELECTRIC FIELD, RADIATION, SOUND ..





If we assume that our light source in this experiment is a point source, we should be able to investigate the validity of the inverse-square law, and determine the luminosity of the light source.



This sensor is designed to simulate the human eye and displays light intensity in lux. 1 lux = 1 lumen/m^2

PROCEDURE

1) See the front page. The sensor needs 3.3V and not 5V. If you provided 5V instead of 3.3V,it will be damaged. It communicates with arduino through the protocol I2C. Although you don't have to worry about it. Upload the sketch in arduino.

2) Open the Serial monitor to see if the sensor is working. Close the serial monitor.

3) Prepare your set up. The light is taped on the table. A meter stick is touching the light bulb. Open the serial monitor and record the light intensity at different distances from the light bulb. Record in the table below. (I am skipping 40cm on purpose)

Distance (cm)	**Intensity lux**		**Distance (cm)**
10sm			75
15cm			80
20cm			85
25cm			90
30cm			
35			
45			
50			
55			
60			
65			
70			

ANALYSIS

note: A key for the instructor is provided at the end.

Don't put the materials away but switch off the bulb.

1) Copy the distances and the intensities in a spreadsheet. Make a scatter plot. Label the axis and give the graph a title.

2) Do you get an inverse square law ? To find out use your TI to trace $y = 1/x^2$.
OR you can go on line on https://www.wolframalpha.com/ and type ***plot 1/x**2***
Is your graph similar to what you get with the TI or website ?

3) so if the distance from the source increases by a factor of 3 (example: you move from 1m away from the source to 3m away from the source, then the intensity of the light decreases by a factor of _____. (or is divided by _____)

4) Use your graph to predict the intensity of light when the distance is 40cm b1 = _____

Run the experiment with a distance = 40cm . b2= _____

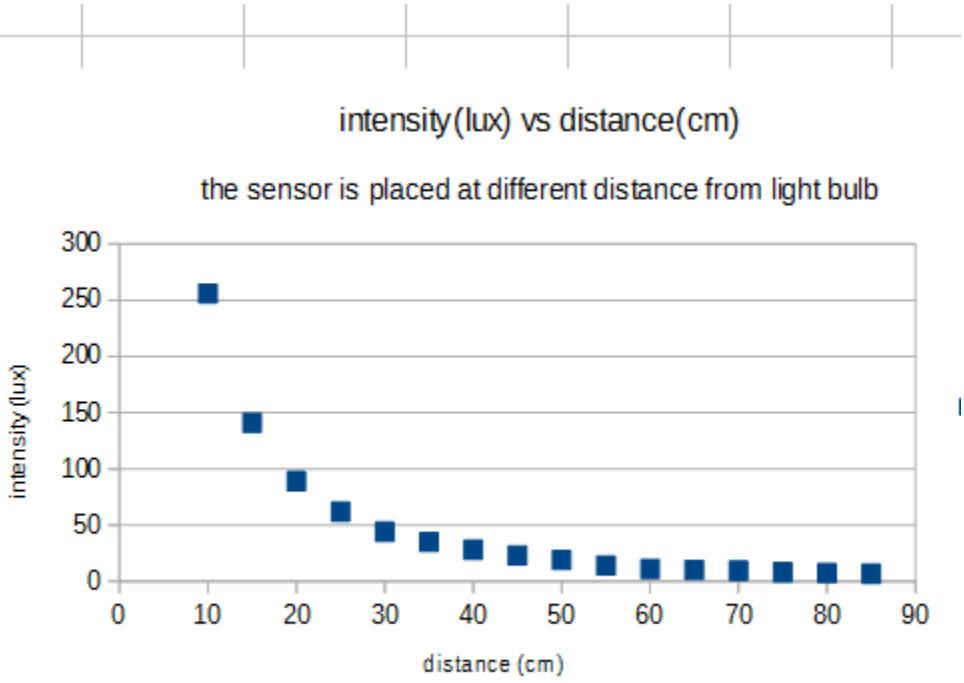
Compute the % error = _____
((b2– b1) /b1) x100

CONCLUSION What was the goal of this lab ? Did you reach the goal of the lab ?

For instructor:

distance (cm)	intensity(lux)
10	256
15	141
20	89
25	62
30	44
35	35
40	28
45	23
50	19
55	14
60	11
65	10
70	9.5
75	8
80	7.4
85	6.7

Sensor
TSL2561
sparkfun

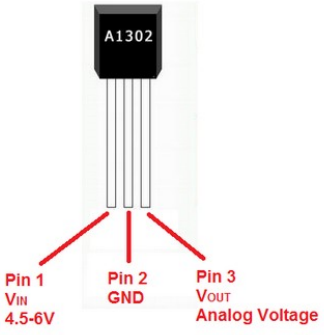


Prepared by Veronique Lankar 7/21/16

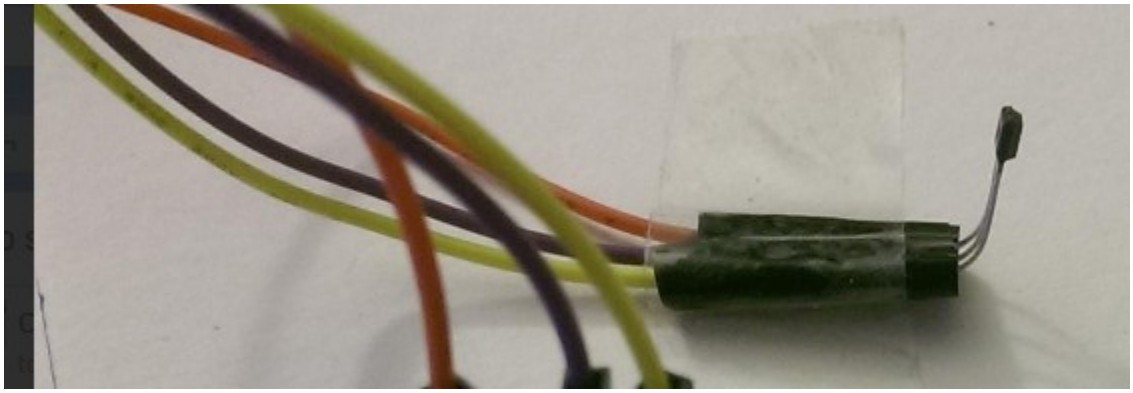
EXPERIMENT : magnetic field in a solenoid

MATERIAL : inductor pasco 3200 turns (inductance is about 0.2H without core), DC power supply that can provide 3V, 4.5V, 6V, 9V, 12V. I used an enercell AC adapter that provides all those voltages with a max current of 1A. I split the ends to get 2 terminals and soldered alligator clips. Connecting wires to connect power supply to solenoid. Multimeter to measure current. Hall effect sensor A1301/A1302 with 3 jumper wire male-female. Use an electrical tape to secure the connections sensor-wires.
see http://www.learningaboutelectronics.com/Articles/Hall-effect-sensor-circuit.php
https://www.pasco.com/prodGroups/coils-and-cores/index.cfm#orderItem_SF-8613

Purpose: to show the linear relationship between the magnetic field B and the current I inside a solenoid.
$B=u\ N/l\ I$ with $u=4\ pi\ 10^{-7}$, l is the length of the solenoid and I is the current going through the solenoid.

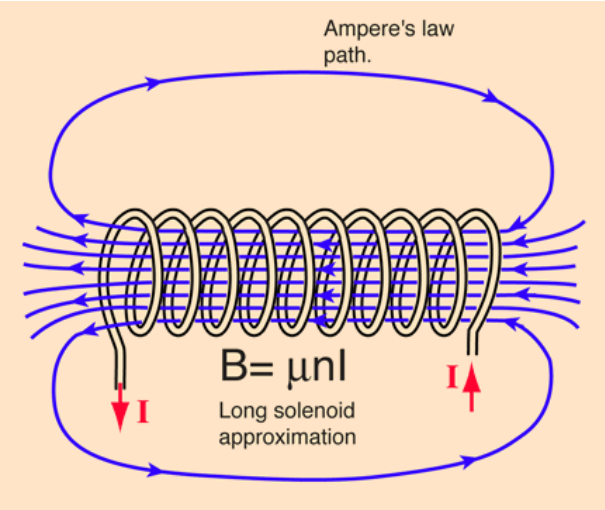


Round face facing you



Background

A long solenoid behaves like a bar magnet and its magnetic field is B = uN/l I (in Tesla)
u is a constant. u=4 pi 10^{-7} T m /A,



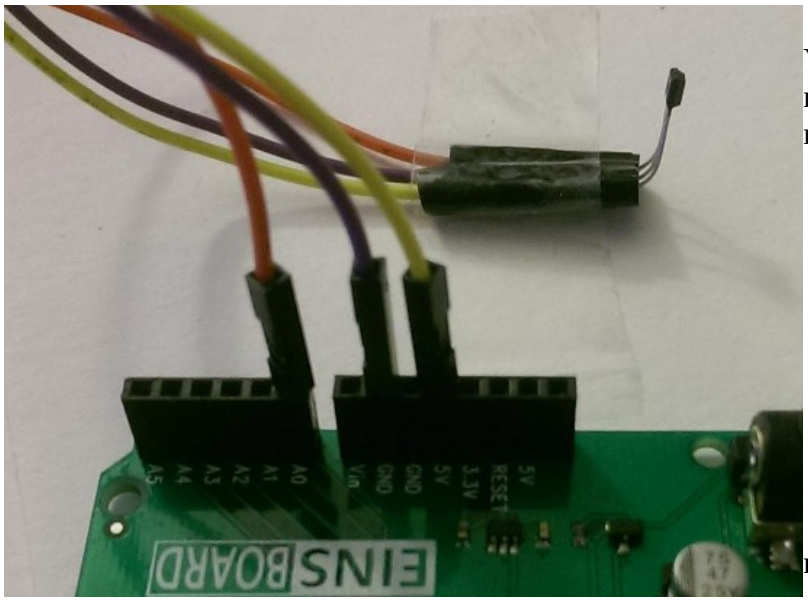
l is the length of the solenoid and I is the current going through the solenoid

The goal of the lab is to show that the slope of B versus I is uN/l

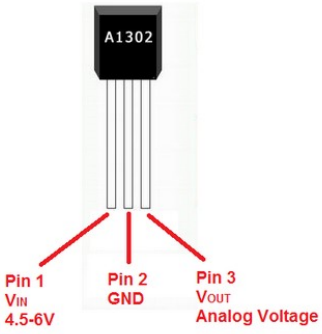
http://hyperphysics.phy-astr.gsu.edu/hbase/magnetic/solenoid.html

PROCEDURE

1) connect the sensor to arduio. Pay attention to the side of the sensor.



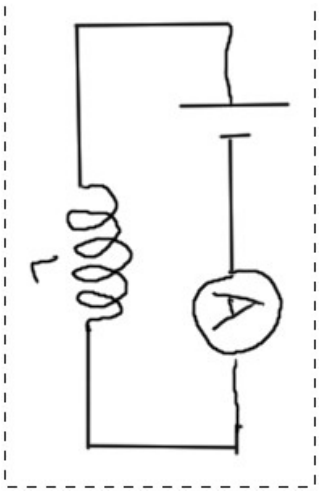
Yellow is 5V here
red is A0
middle one is Gnd

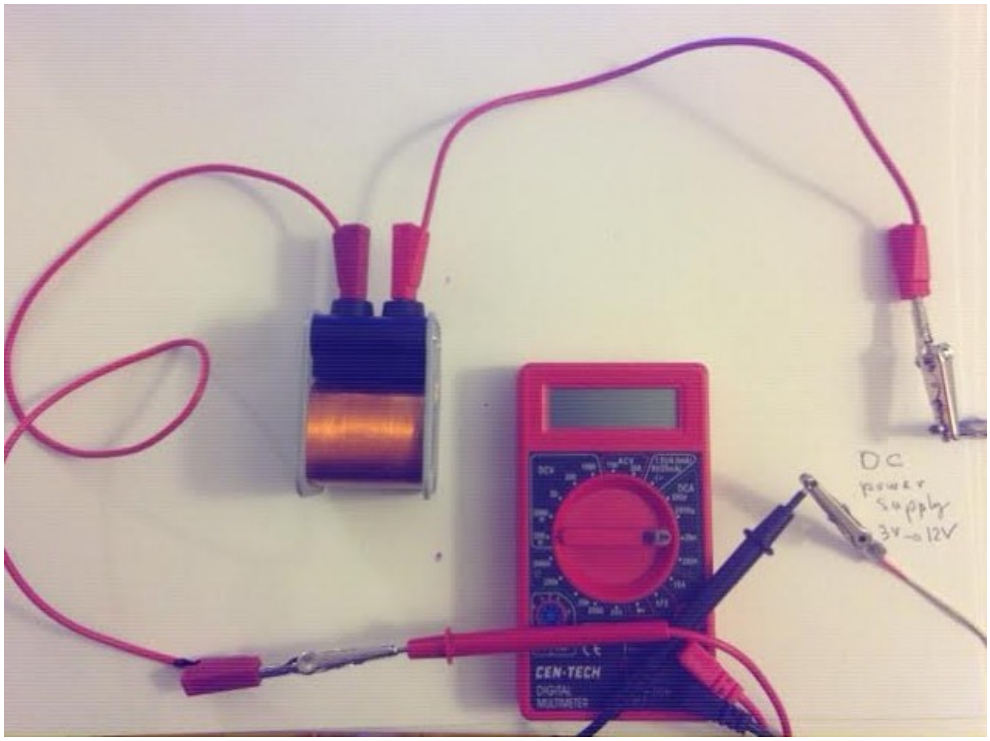


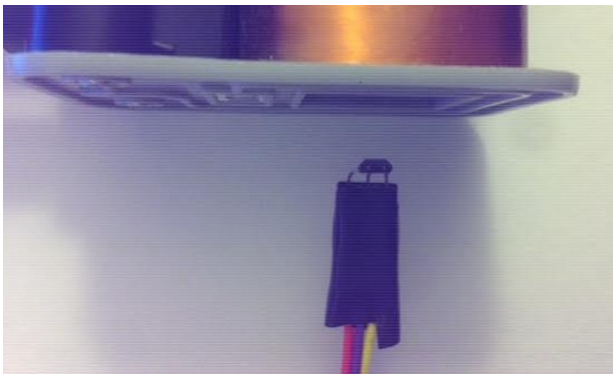
round face is facing you here.

2) Connect the arduino to the computer. upload the sketch in the arduino.

3) Build the circuit: power supply – ammeter – solenoid – power supply







3) Close the circuit with the power supply of 3V. Insert the sensor curved face first inside the solenoid. Open the Serial monitor and measure the magnetic field in Gauss. Move the sensor so you get the largest number. Record in the table below. The unit is Gauss. 1 Gauss=0.0001. Measure the current and report in the table. If you read mA then divide by 1000 to get Amps. Don't worry about negative sign. It is to tell the North from the South face of the electromagnet.

Voltage (Volts)	**Current(Amps)**	**B (Gauss)**
3		
4.5		
6		
7.5		
9		
12		

4) Repeat 3) for the other voltages.

ANALYSIS
instructors: the results are on the last page.

1) With a ruler measure the length of the solenoid. (the length of the coil)
l = _____- cm = _____m

2) The number of turns is N=_____- turns

3)In theory B = uN/l I . So we expect a linear relationship between B and I with a constant of proportionality equals to uN/l. Compute uN/l with $u=4pi10^{-7}$ T m/A

uN/l = _____ Tm/A

multiply by 10,000 because the magnetic field measured by the sensor is in gauss (1T =10,000 G)

uN/l = _____ Gm/A

4) Using the previous table, make a scatter plot B versus I.

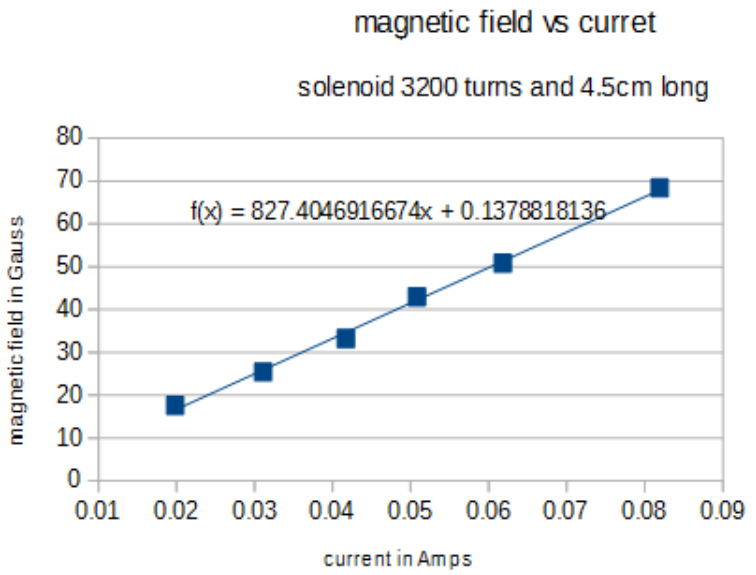
5) insert a best fit line and display the equation. You get B = ___ I and the slope is _____ Gm/A

6) Compute the % error between 3 and 5

CONCLUSION

3	0.01983	17.6
4.5	0.0311	25.4
6	0.0417	33.2
7.5	0.0508	43
9	0.0618	50.8
12	0.0819	68.4

voltage (V)	current(A)	magnetic filed (G)



B = uN/l l with N=3200 and l = 4.5cm = 0.045 and u=4pi10^-7 so the theoretical slope is uNl = 894 about.
The experimental slope is 827. So this is a 7% error. Consistent with the properties of the sensor.

Prepared by Veronique Lankar 7/21/16 with the assistance of Elias Benichou (NYCischool)

Experiment : ENERGY CONTENT IN a Marshmallow (or peanut)

PURPOSE:

1) use a Temperature sensor attached to arduino (DS18B20 from adafruit) to measure the change in temperature of water that is heated by a burning sample of food

2) calculate the amount of thermal energy absorbed by the water

3) compute the energy content in Cal of the sample of food

4) visualize the graph temperature vs time when the water get heated

Note from author: The Cal in the marshmallow will be off with the material I used. But you can still get a nice graph. To get a better result, one needs a true calorimeter and the experiment with a peanut works better. Doing the lab with a marshmallow is more fun and avoid issues with food allergy.

MATERIALS:

https://www.adafruit.com/products/381?gclid=CJHSjaGC-80CFVhbhgodcNMLiA

stand, wire to kebab the marshmallow or holder, lighter, holder for can, marshmallow, digital scale, jumper wires, silver empty silver soda can, arduino with USB cable.

BACKGROUND:

When burning food heats a known quantity of water, the amount of heat given off by the food is theoretically equal to the amount of heat gained by the water. The following is an equation that describes this idea:

$$Q = m \times c \times \Delta T$$

where Q is the amount of heat, m is the mass of the water, c is the *specific* heat of the water, and ΔT is the change in temperature of the water.

The specific heat of water is:

$$c = 1 \frac{\text{calorie}}{\text{gram°C}} = \frac{4.18 \text{ joule}}{\text{gram°C}}$$

Q is also the energy content (Cal) of the food samples of mass m. The goal is to find the energy in Cal/g of the food burnt.

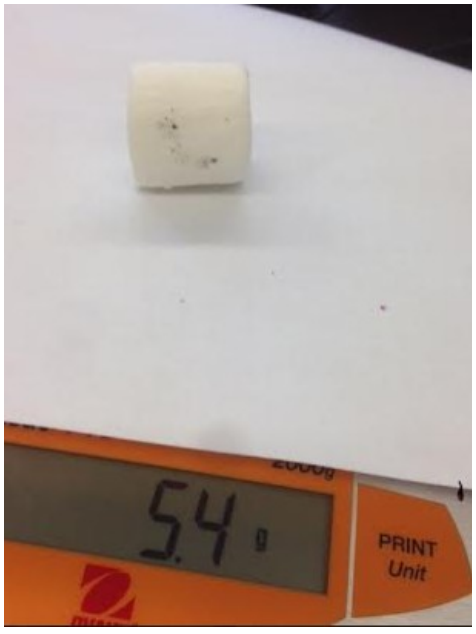
PROCEDURE:

You will use an arduino and a water proof thermometer sensor to measure the temperature of the water. The serial monitor of the arduino will display the time (ms) and the temperature (C)

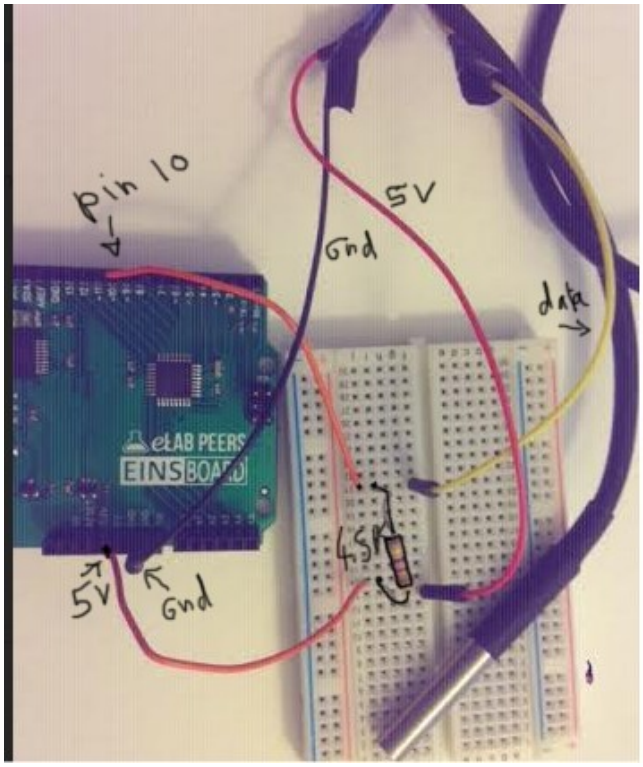
1) fill the beaker with water to the level 100ml.

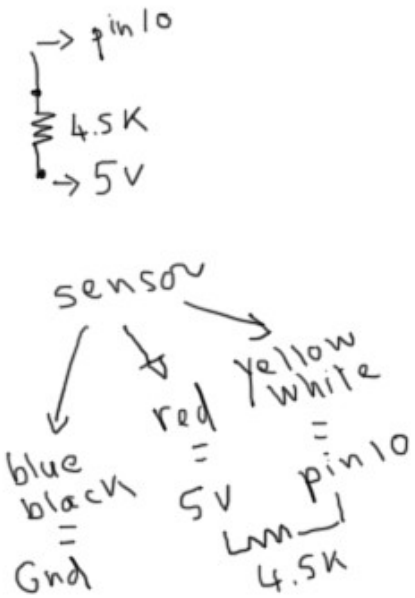
You know the mass of the water because 1ml of water weighs 1g. Report the mass of water in the data Table.

2) place a paper on a digital scale and zero it. Record the mass of the marshmallow in the table below.



3) connect the thermometer sensor to the arduino as shown below. The red wire is for 5V. The black or blue for Gnd. The yellow or white wire (data wire) is for pin 10. there is a 4.7K resistor between the yellow wire and the red wire (between the data wire and the 5V wire).



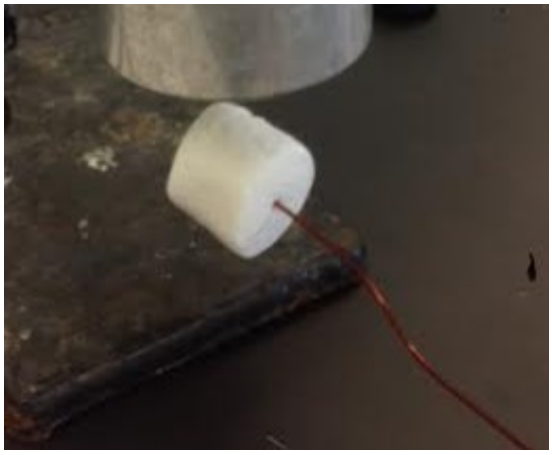


4) Connect arduino to the computer with an USB port .

5) upload the code in the microcontoller. open the Serial monitor to see if the sensor is working properly and is displaying temperature. Close it.

6) set up the experiment. See front page. The beaker in the holder. The probe in the water. Open the serial monitor. Wait for the temperature to stop changing and **record the initial temperature of water in the data Table.**

7) kebab the marshmallow



8) Set the marshmallow in fire and place it just below the water. Wait for the marshmallow to be fully burnt and **stir the water with the probe at the same time**. The temperature of the water should increase. When the temperature reaches a maximum unplug the USB. Copy and paste the data in a spreadsheet from the minimum temperature to the maximum. Temperature. The format in . csv (separated by a comma). Save the file. Record the maximum temperature in the table below.

Data Table

1	Mass of water	g
2	mass of marshmallow+wire before	g
3	mass of marshmallow + wire after	g
4	Change in mass = food gone	g
5	initial temperature	C
6	final temperature	C
7	temperature change, ΔT	C
8	heat, Q in cal (row 1 x row7)	cal
9	Q in Cal (divide by 1000) Cal is good calories	Cal
10	Cal/g – (divide row 8 by row 4)	Cal/g

ANALYSIS

see the graph I get for burning a marshmallow at the end of this page

1) compute the change in temperature and record in the data Table.

2) Compute the energy content in cal of your sample. Q = (mass of water) x (change in temperature)
Record in the table.

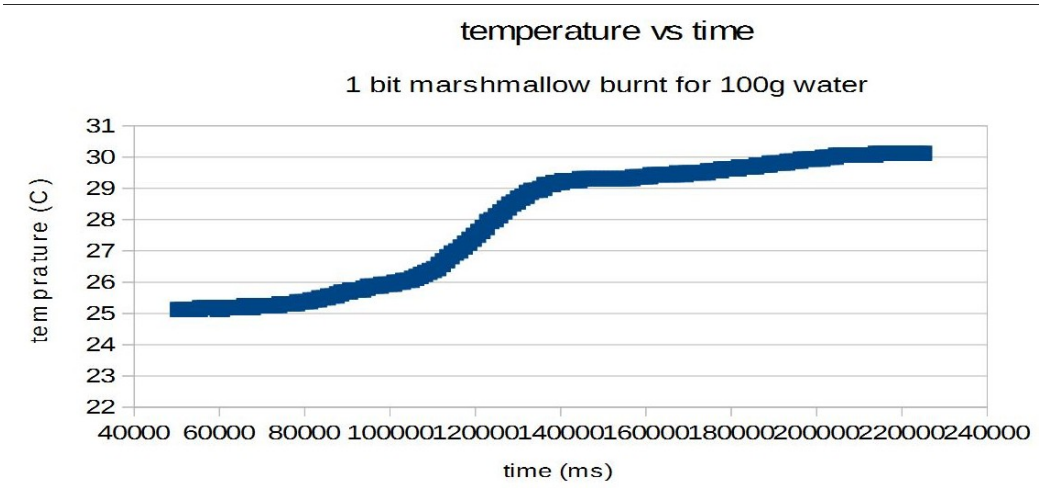
3) To convert to food Cal (upper case C) divide by 1000. Record in the table.

5) Compute the amount of Cal/g and report in the table. (divide the energy in Cal by the mass of food gone)
Report in the table.

6) Compute the amount of Cal in 100g of food. Is it consistent with 300 Cal in 100g ?
How can you explain the difference.

7) Open the spreadsheet. You have 2 columns. One is for time (ms) and one is for temperature.
Make a scatter plot temperature versus time and discuss the graph. Label the axis and give a title to the graph.
Print the graph and attach it to your report.

CONCLUSION

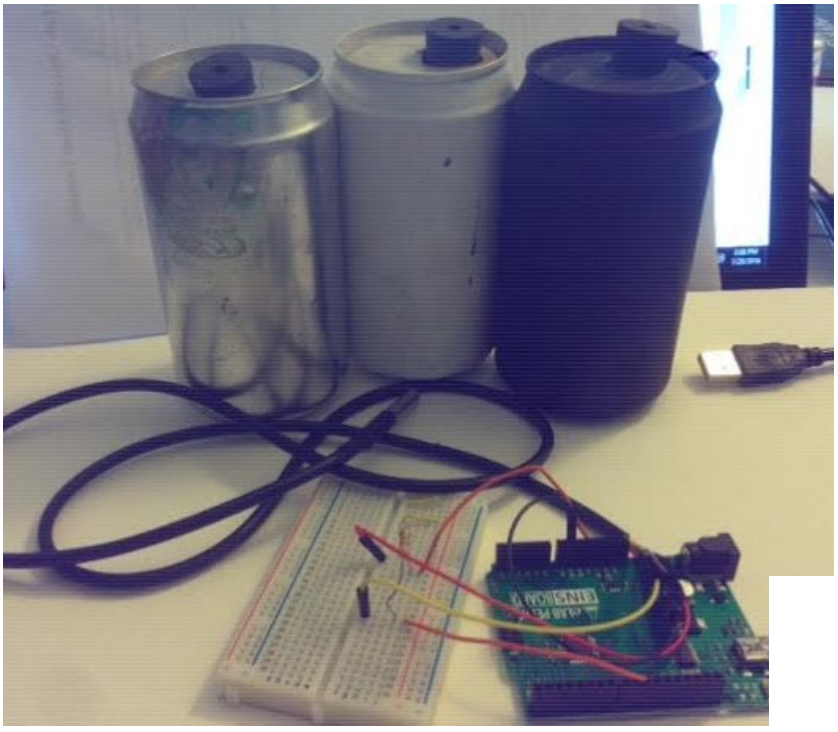


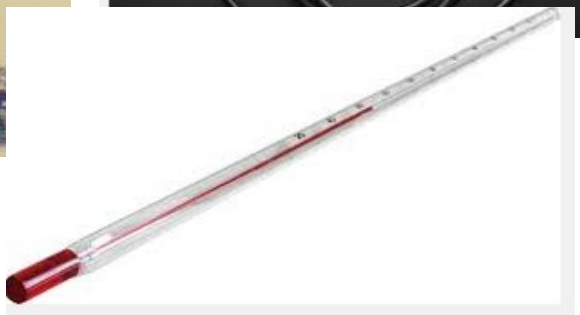
Prepared by Veronique Lankar 7/21/16 with the assistance of Elias Benichou (NYCischool)

EXPERIMENT : cooling by radiation
white / silver / black cans.

MATERIAL : 3 cans. One painted silver, one white and one black. Temperature sensor DS18B20 water proof from adafruit. 2 Beakers 400mL. Hot plate. Regular thermometer.

Purpose: To show that the rate of cooling decreases with time and follows an exponential decay
- To show that black object (for the same area and mass) cool faster than white that cools faster than silver.





Background

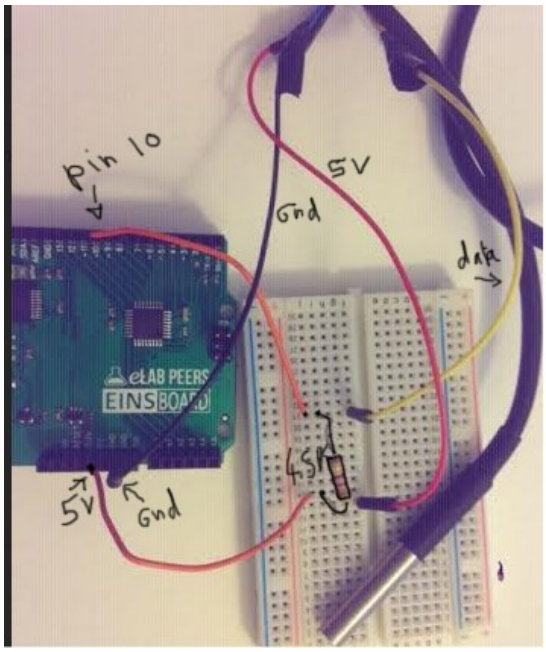
The rate at which objects cool down depends on the difference of temperature between the object and the environment (Newton's law of cooling). We will observe the cooling of a mass of water inside a white, black, and silver can. The hot water is initially at a temperature of Ti and will reach the temperature of the room To.

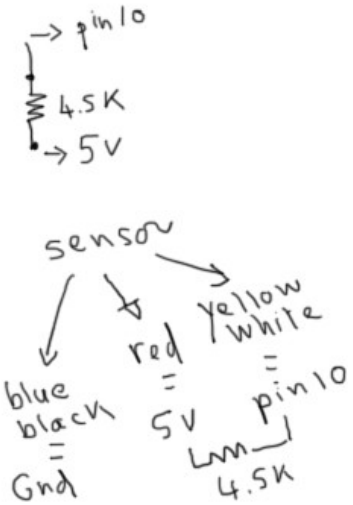
You will show that : **dT/dt = - k (T-To)** or **(T – To)= (Ti-To) exp(-kt)**
To is the temperature of the room
Ti is the temperature of the water before it cools down
k is a constant that depends on the object (here color).

You will also compare **k** for the different colors.

PROCEDURE

1) Connect the sensor to arduino . The red wire is for 5V . The black or blue wire is for Gnd. The yellow or white wire is for the data wire. You also need a 4.5k resistor between 5V and data wire. (pull-up resistor).



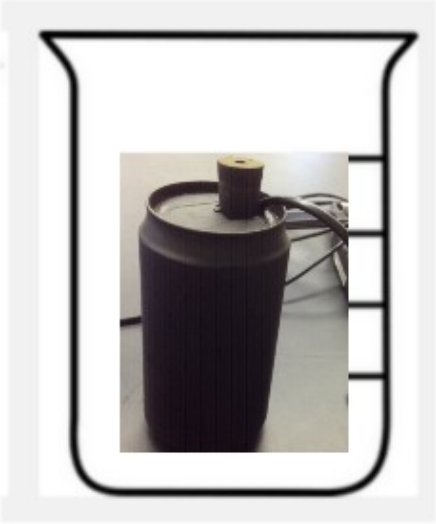


2) connect the USB wire to the laptop. Upload the sketch in the chip. Open the Serial monitor and check if the sensor is working properly and record the room temperature To To= _____ C. Close the Serial monitor.

You need to wear gloves during the procedure.

3) Use a 400 ml beaker and fill it with 300mL of water. 300ml of water has a mass of 300g. Place the beaker on a hot plate with a thermometer inside and heat it up until you reach 60C.

4) When it reaches 60C poor the hot water inside the can very slowly. The can can be placed in another beaker to make sure it does not fall. Insert the temperature sensor. Open the serial monitor. Stir time to time.



5) When the can reaches room temperature unplug the USB cable and copy and paste the data in a spreadsheet (comma separated data) . Close the Serial monitor.

6) repeat for the other 2 cans. Make 2 other spreadsheets.

ANALYSIS
instructors: the results are on the last page.
For each can:
1) Make 2 new columns with: time starts at 0 and is in seconds and Temperature – To.
Here is an example where the temperature of the room To is 22C.

B	C	D
Temperature C	time (seconds) starting 0	Temp – 22
57.0625	0	35.0625
57.0625	0.001035	35.0625
57.0625	0.002071	35.0625
57.0625	0.003107	35.0625

2) make a scatter plot (temperature – To) vs time (s) . Do you get an exponential decay ?

3) Fit an exponential to the data using insert trend line.

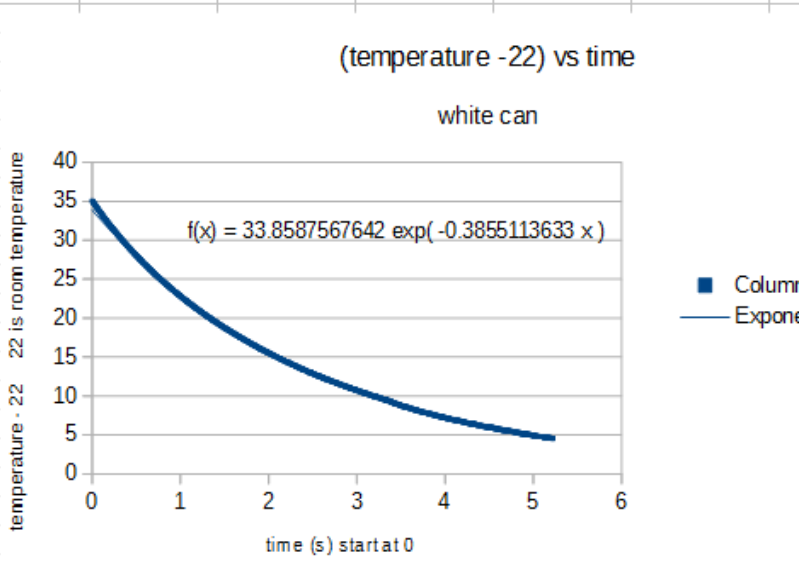
You get (T-To) = _____ exp(-…. t)

What is the coefficient inside the exponential ?

4) Repeat for the other can and compare the coefficient. Which can has the largest coefficient ?

The largest coefficient means the smallest time constant. Discuss.

time (us)	Temperature C	(time (s)=(time(us)-431851)/1000000	Temp – 22
431851	57.0625	0	35.0625
432886	57.0625	0.001035	35.0625
433922	57.0625	0.002071	35.0625
434958	57.0625	0.003107	35.0625
435993	57	0.004142	35
437028	57	0.005177	35
438064	57	0.006213	35
439100	56.9375	0.007249	34.9375
440135	56.9375	0.008284	34.9375
441170	56.9375	0.009319	34.9375
442206	56.9375	0.010355	34.9375
443242	56.875	0.011391	34.875
444277	56.875	0.012426	34.875
445313	56.875	0.013462	34.875
446348	56.875	0.014497	34.875
447384	56.8125	0.015533	34.8125
448419	56.8125	0.016568	34.8125
449455	56.8125	0.017604	34.8125
450491	56.75	0.01864	34.75
451526	56.75	0.019675	34.75
452561	56.75	0.02071	34.75
453597	56.75	0.021746	34.75
454633	56.6875	0.022782	34.6875
455668	56.6875	0.023817	34.6875
456704	56.6875	0.024853	34.6875
457740	56.6875	0.025889	34.6875
458775	56.625	0.026924	34.625

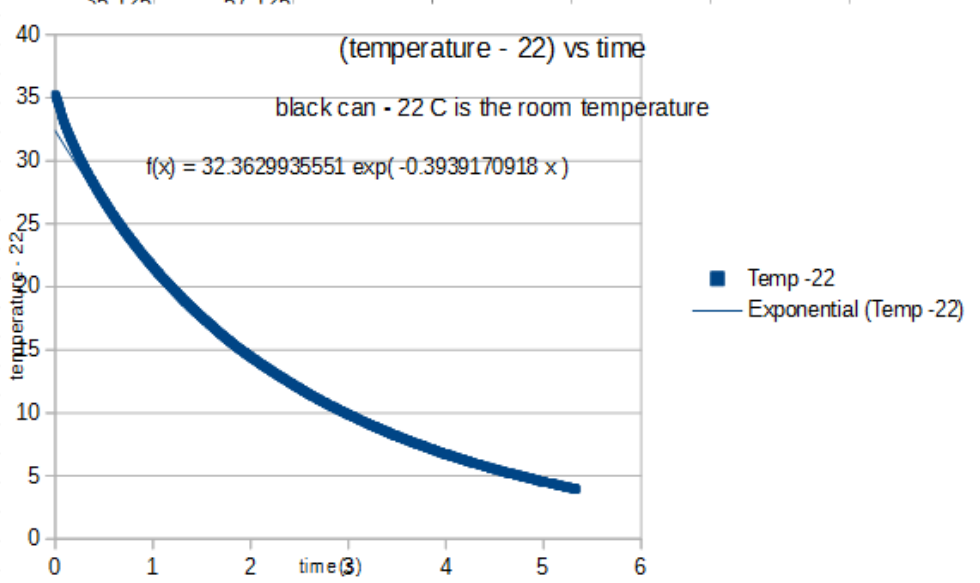


the temperature of the room WAS 22.

To find the exponential decay we have to remove 22 from temperature

T – Troom = Tinital exp(-kt)

time(us)	(time (s)=(time(us)-1069)/1000000	Temp -22	temperature	
1069	0	35.25	57.25	A= time in milliseconds arduino
2104	0.001035	35.25	57.25	D= temperature in celsius recorded by arduino
3139	0.00207	35.1875	57.1875	C = D – 22 with 22 temperature in the room
4174	0.003105	35.1875	57.1875	B = (A -1069/1000) so time in seconds and starts at 0
5211	0.004142	35.1875	57.1875	we plot C vs B
6246	0.005177	35.125	57.125	
7281	0.006212	35.125	57.125	
8316	0.007247			
9352	0.008283			
10387	0.009318			
11423	0.010354			
12459	0.01139			
13494	0.012425			
14529	0.01346			
15565	0.014496			
16601	0.015532			
17636	0.016567			
18671	0.017602			
19706	0.018637			
20743	0.019674			
21778	0.020709			
22813	0.021744			
23848	0.022779			
24885	0.023816			
25920	0.024851			
26955	0.025886			
27991	0.026922			
29027	0.027958	34.625	56.625	



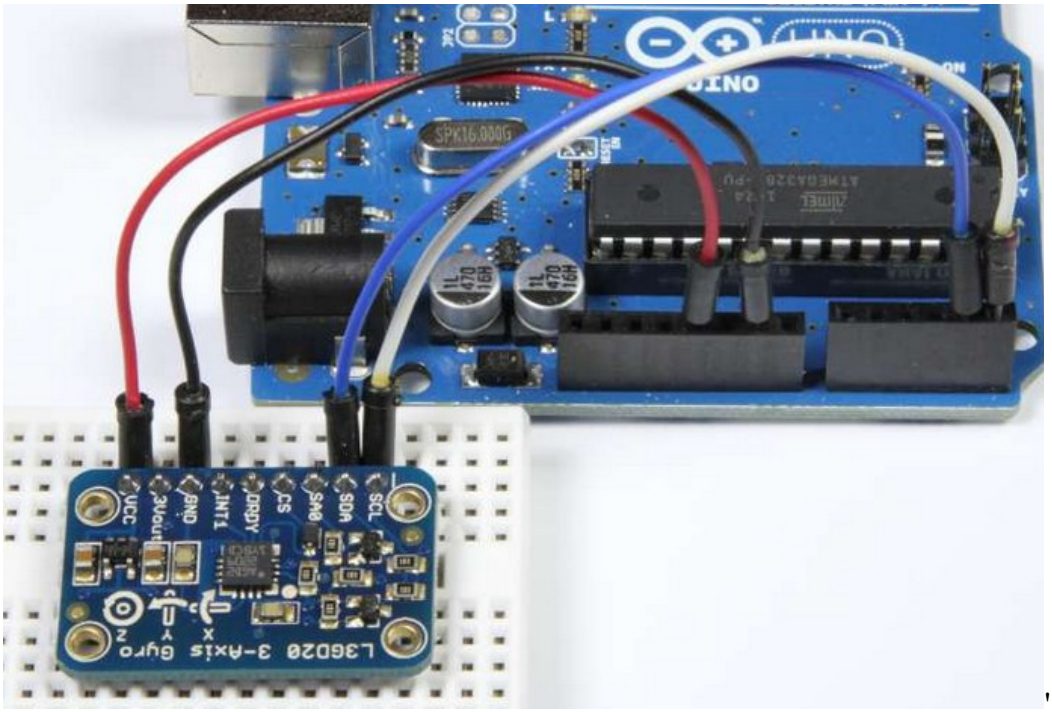
Prepared by Dr. Veronique Lankar 7/21/2016

with the assistance of Sean Hefferman (student @ Physics Department, Manhattan College) as part of a Summer research project (2016). (Arduino based acquisition systems)

EXPERIMENT: PHYSICAL PENDULUM / rotational speed.

Materials: sensor gyroscope L3G20 from adafruit (sensor that measured the rotational speed in 3 axes)
https://www.adafruit.com/products/1032
A copper pipe from hardware (about 30-40cm long), stand
arduino, arduino shield + small breadboard

Goal: visualize the sinusoidal behavior of the rotational speed of the physical pendulum. Compute the period and compare to the theoretical one.
Compute the maximum rotational speed and compare to the theoretical one.



https://learn.adafruit.com/adafruit-triple-axis-gyro-breakout
to learn more about the sensor, to download the libraries the sensor needs to function.
The sketch needs two libraries:
#include <Wire.h>
#include <Adafruit_L3GD20.h>

BACKGROUND

A physical pendulum is simply a rigid object which swings freely about some pivot point. It obeys Newton's second law and for small angles the period is given by the equation :

$$T = 2\pi\sqrt{\frac{I_{\text{support}}}{mgL_{cm}}}$$

http://hyperphysics.phy-astr.gsu.edu/hbase/pendp.html

I is the rotational inertia of the pendulum and **Lcm** the distance between the center of mass and the pivot. The mass of the pipe is **m**. The unit is in seconds.

In the case of a rod (pipe) we have **I=1/3 (m) L^2** with **L** the length of the pipe.

Lcm = ½ L. The center of mass is at the center of the pipe.

So the equation becomes : **T = 2pi sqrt (2L/3g))** . So the effective g is 3g/2

The equation of motion, describing the angle as a function of time, is **Amax cos((2pi/T) t)**
(or if it is damped exp(-t/tau) Amax cos((2pi/T) t))
with **Amax** the maximum angle reached by the pendulum. So for example if we start with an angle of 20 degrees then Amax = 20.

The rotational speed is **Amax 2pi/T sin(2piT t)** (derive the previous expression if we neglect the damping.
So maximum rotational speed is **Amax 2pi/T or 20 x 2pi / period if the Amax is 20 degrees.**

So we expect a sinusoidal graph for the rotational speed and we expect the period to be :

T = 2pi sqrt (2/3 (L/g))

And we expect the maximum rotational speed to be **Amax 2pi/T**

PROCEDURE

Your if you can make 2 holes at one end of the pipe to thread a copper wire, that will make a great pivot point.

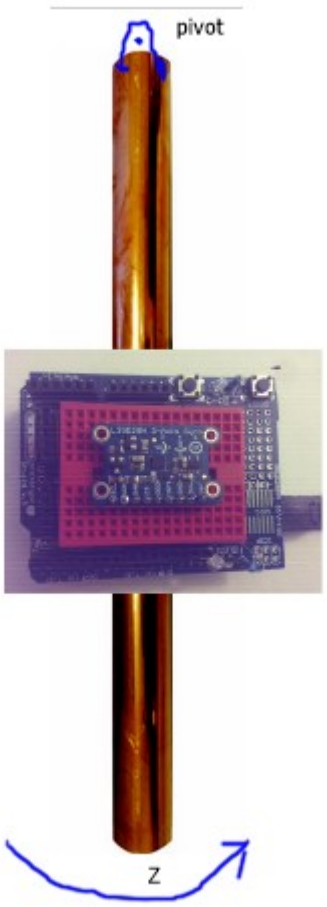
1) make the right connections between arduino and the gyroscope plugged in breadboard. See the front page.
You can improve the connections if you use more solid jumper wire.

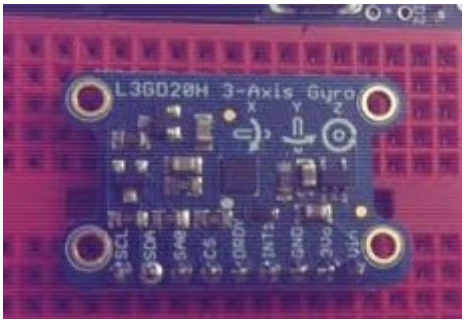
2) connect the arduino to the computer with an USB cable. Upload the sketch. Make sure all the libraries have been installed (see front cover). Open the serial monitor and rotate the sensor in the 3 directions to see if the sensor is working. We will use only the z direction.

3) If it is working then place the breadboard on the shield (glue?) and plugged the shield in the arduino.

4) Tape the arduino at the center of the pipe so the center of mass is not changed. As shown below. You will use only the z direction.

The arduino will oscillate left to right. Try to arrange the cable so it is not on the way of the oscillating pipe. And the cable should not pull on the pipe. The cable can be upward and your computer above the stand.





5) Open the Serial monitor and make sure it works. We are only interested in the first data (time in ms) and the last data (speed along the z direction). Close the Serial monitor.

6) Move the pipe by about 20 degrees and let it oscillate for a several oscillations then unplug the cable.

7) copy and paste the data in a spreadsheet. We will be working on the first column (ms) and the last column (z)

Your data looks like:

A	B	C	D
time(ms)			speed(deg/s)
2	6	-28	122
53	1	-24	106
104	5	-20	86
156	7	-14	54
207	3	-8	18
259	0	-2	-19
311	-4	5	-53
362	-5	11	-77
413	-1	19	-100

ANALYSIS
The key for the instructor is on the last page.

1) Use only column A (time in ms) and the column D (angular speed of pendulum)
Make a scatter plot but connect the dots.
What graph do you get ?

2) Compute the period T. (time between 2 peaks). T1=_____

3) Measure the length of the tube L = _____ cm
and compute the theoretical period T = 2pi sqrt (2L/3g) with g= 980cm/s/s

T2= _____

Compute the % error between T1 and T2. (T2-T1)/T2 x 100= _____

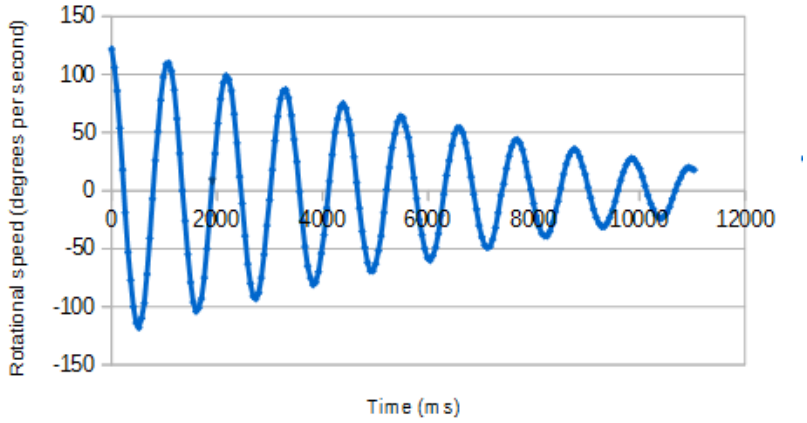
4) according to your data what is the maximum speed in deg/s = _____

Compare to the theoretical speed Amax 2pi / T2 with Amax = 20 degrees if this was your initial angle.

Compute the % error.

CONCLUSION

time(ms)			speed(deg/s)
2	6	-28	122
53	1	-24	106
104	5	-20	86
156	7	-14	54
207	3	-8	18
259	0	-2	-19
311	-4	5	-53
362	-5	11	-77
413	-1	19	-100
465	-2	25	-114
517	-8	28	-118
569	-11	30	-110
620	-6	28	-97
671	-5	24	-72
723	-3	16	-41
775	-2	6	-7
826	-4	-3	26
878	-2	-12	51
929	-1	-21	78
980	4	-27	98
1033	7	-31	109
1084	5	-31	110



Prepared by Veronique Lankar 7/21/16

Experiment : Simple harmonic motion – vertical spring.

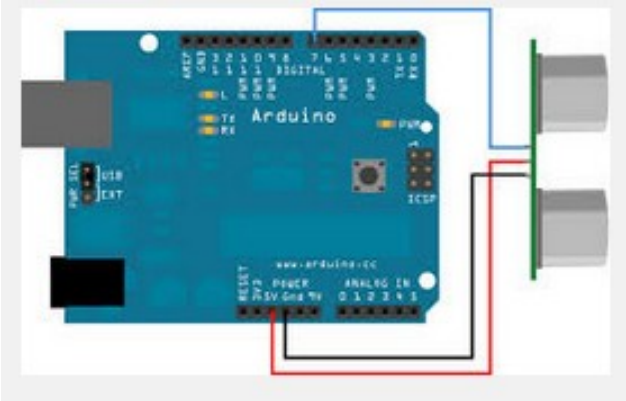
PURPOSE:

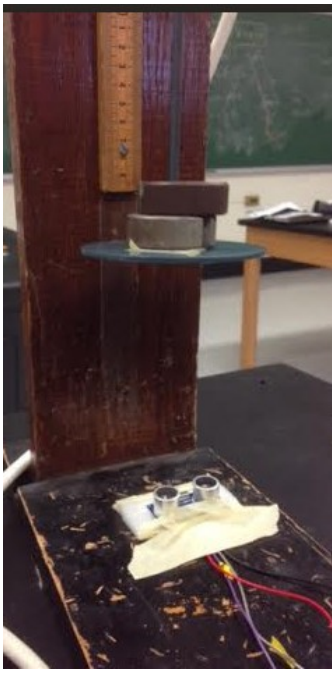
1) visualize the sinusoidal motion of a mass on a vertical spring

2) understand the SHM

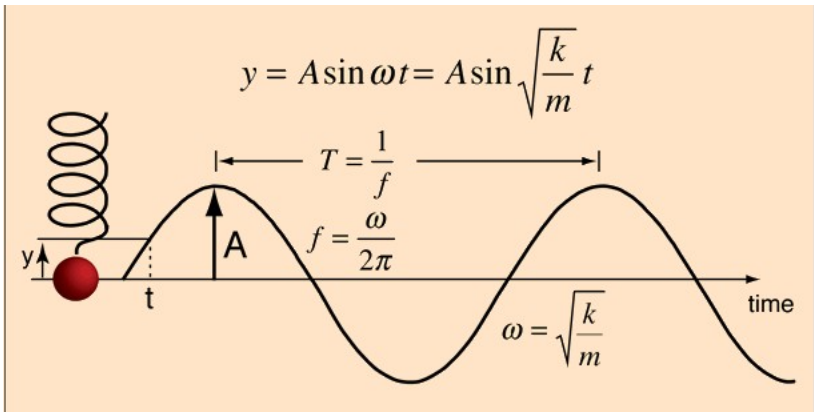
3) show that the period of the spring is 2pi sqrt(k/m)

MATERIALS: stand with large holder, 2 x 200g mass, arduino with USB cable, Ping Ultrasonic Range Finder sensor, sketch to upload.





BACKGROUND:



http://hyperphysics.phy-astr.gsu.edu/hbase/shm.html

A mass on a spring of spring constant oscillated with a period $T = 2\pi\sqrt{\frac{m}{k}}$.

The motion of the spring is sinusoidal.

m is the (total mass attached to the spring including holder + 1/3 mass of the spring). If the spring has a small mass then you can ignore the 1/3 in the equation.

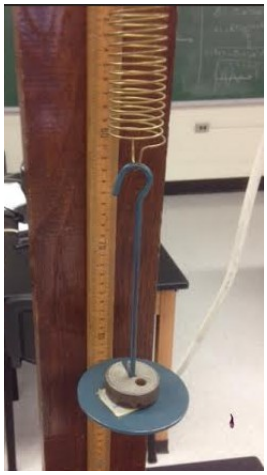
PROCEDURE

1) first we need to find k, the spring constant using Hooke's law. k x = M g.

In this case the spring is not moving and is at rest with just its holder. Add a mass of M = 200g on the holder and measure the displacement of the holder.

x= _____cm = _____m

Compute the spring constant k = (0.2 x 9.81) / (displacement x) = _____ N/m



2) Now place the 400 g on the holder.

Measure the mass of the spring if your spring is heavy = _____ g

the total mass m is **400g + mass holder in grams + 1/3(mass spring)** in grams = _____g

convert to kg (divide by 1000) = _____ kg.

Compute the expected period 2pi sqrt (m / k) = _____ seconds.

3) Connect the arduino to the sensor. Vcc is for 5V. Gnd goes to Gnd. The signal wire goes to pin 7. (see front page).Connect the arduino to the computer with the USB cord.

4) upload the ultrasound code in the arduino. Open the Serial monitor to check if the sensor works well. The sensor is fragile. Don't press it. Close the Serial monitor.

5) Set up the experiment as shown in the front page. Use a tape to secure the sensoe underneath the holder.

6) Pull the mass so it oscillates and open the Serial monitor. You should see the distance increasing and decreasing. Wait for about 10s and unplug.

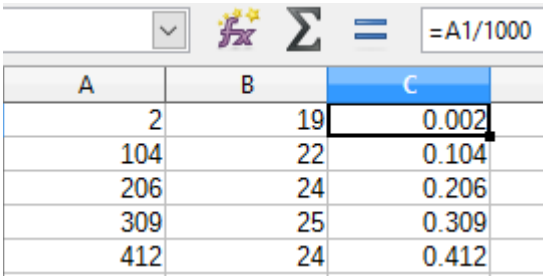
7) Copy the consistent values in a spreadsheet. In the spreadsheet, make sure you don't have outliers.

Your data should look like:

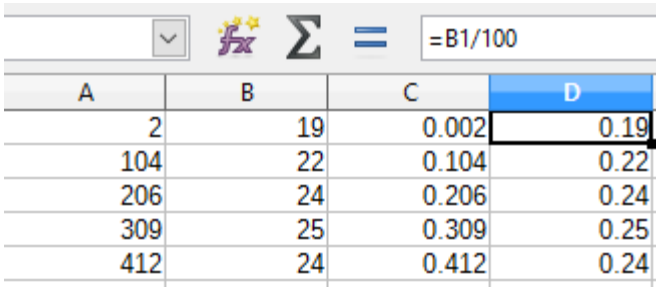
A	B
2	19
104	22
206	24
309	25
412	24
515	21
617	18
718	14
821	9
922	5
1024	3
1125	3
1226	3
1328	5
1430	9
1531	13

The time is in ms and the distance in cm.

8) Make a new column C where the time is in seconds. Column C = Column A /1000



9) Make a new column D where the distance is in meters. Column D = Column B/100



A	B	C	D
2	19	0.002	0.19
104	22	0.104	0.22
206	24	0.206	0.24
309	25	0.309	0.25
412	24	0.412	0.24

10) make a scatter plot *distance (col D) versus time (col C)*. You should get a sine wave ?
Include the scatter plot in your lab report.

ANALYSIS

1) Use the graph to compute the experimental period. (time between 2 peaks)

2) Compare the % difference between this experimental period and the one you computed in procedure 2)

3) Measure the amplitude of your sine wave A = (max distance – min distance)/2 =_____m

4) The equation of your graph is y = A sin ((2 pi / T) x) with T the period and pi is 3.14
plot that equation with wolfram alpha (website) and see if you get a similar graph

CONCLUSION

with the assistance of Sean Hefferman (student @ Physics Department, Manhattan College) as part of a Summer research project (2016).
(Arduino based acquisition systems)

Experiment : Simple harmonic motion – advanced

For Engineering majors or Physics majors.

PURPOSE:

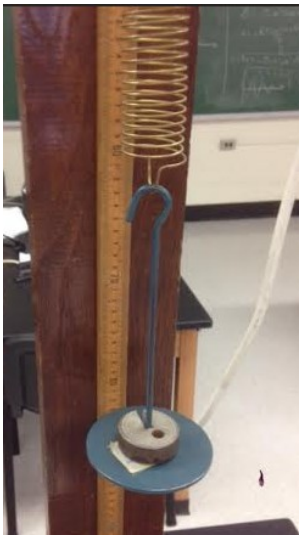
-Proving that the speed of a mass element of the spring depends linearly on the position between the pivot and the location of the element. This why the effective mass in the equation for the period is m/3 + M

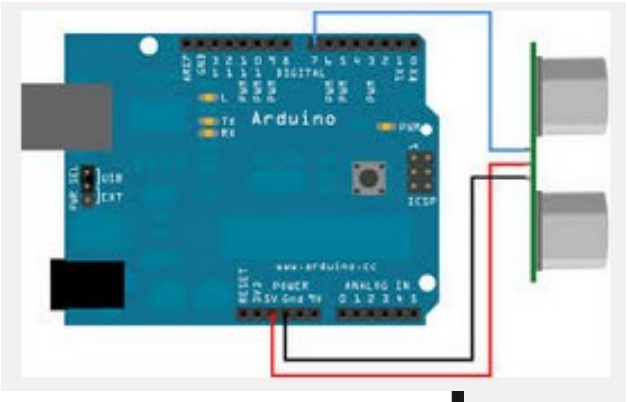
with m the mass of the spring. M the mass on the holder.

https://en.wikipedia.org/wiki/Effective_mass_(spring%E2%80%93mass_system)

- Proving that the period of the system is 2pi sqrt(k/(m/3 + M)) with M the mass of the load on the spring and m the mass of the spring.

MATERIALS: stand with large holder, large spring (k =10N/m about), 2 x 200g mass, arduino with USB cable, Ping Ultrasonic Range Finder sensor, sketch to upload, foil plate type pizza plate.





We will use only the lid

as a reflector for the ultrasound.

BACKGROUND:

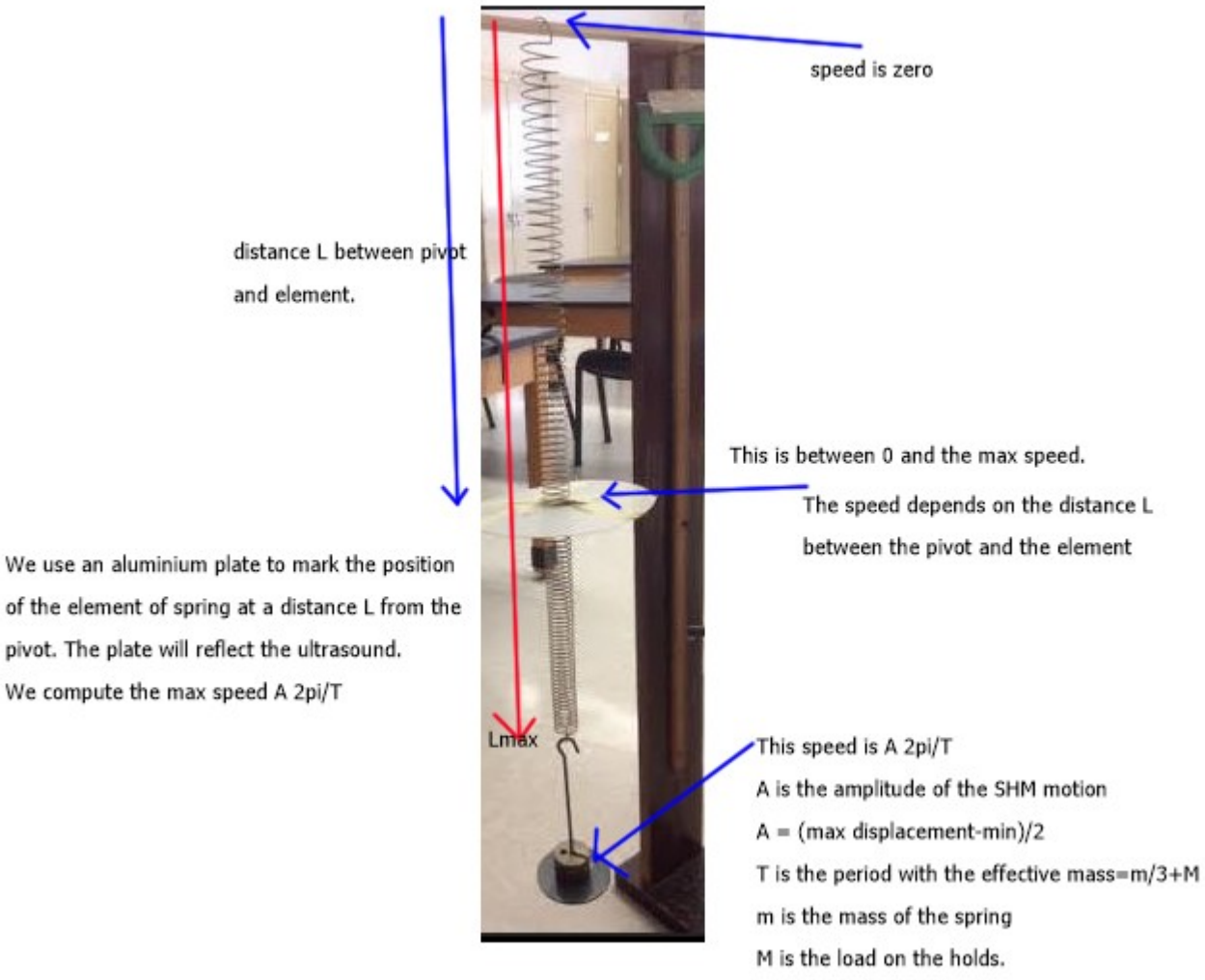
Consider a mass M on a vertical spring oscillating.

Displacement = **A cos((2pi/T) t)**

speed of the mass is **A 2pi/T sin ((2pi/T) t)** if we neglect the damping.

So the maximum speed is **A 2pi/T**

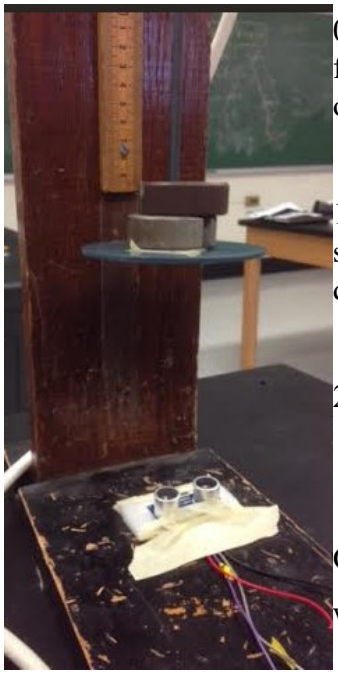
This experience is to prove that the speed of an element on a spring depends on the distance between the pivot and the location of that element. The speed is zero at the pivot and increases linearly until reaching the maximum **A 2pi/T**



ultrasound to record the SHM of the plate.

PROCEDURE - ANALYSIS

my results on the last page (Dr. V. Lankar)



0) measure the length of the spring (with holder and load) when it is at rest. From fixed point to the end of the spring. So the length Lmax of the spring before it oscillates (see previous page in red). Record in the table below.

1) proceed as for the previous SHM motion. Obtain the sinusoidal motion of the spring with the ultrasound sensor placed below the holder. See the previous lab for details.

2) Using your sine graph, compute the period of the motion by using 2 peaks.

T = _____s. Record in the table below.

Compare to the theoretical one. T=2pi sqrt(k/(m/3 + M))

with m the mass of the spring and M the mass of the 0.4Kg on the holder.

3) Using your sine wave, measure the amplitude A of the spring (max-min)/2 and multiply by 2 pi/T to get the maximum speed of the holder.

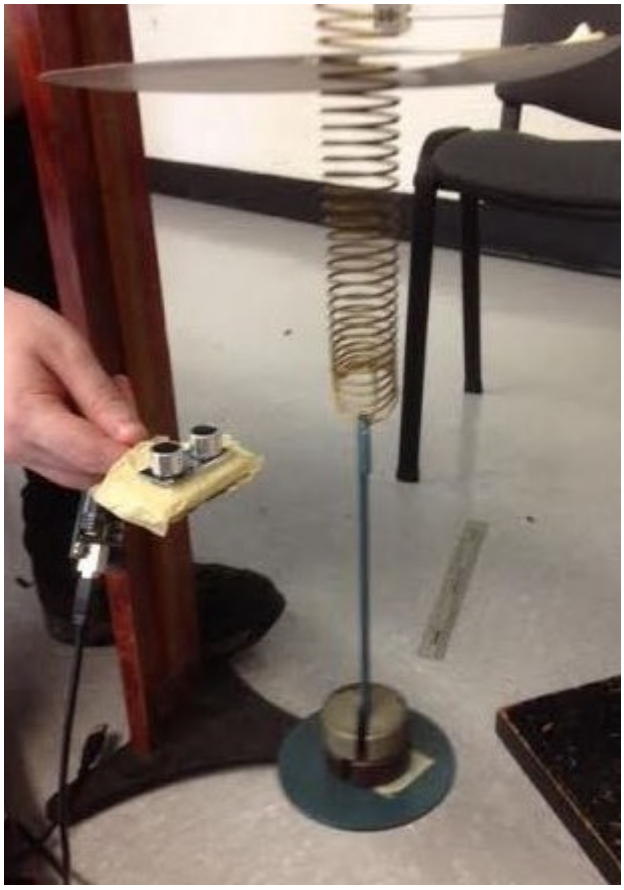
Vmax= _____ m/s . Record the value in table.

Period	Length L	Max speed

4) Use a foil plate cut half way and position it at a different location on the spring. The plate will reflect the ultrasound so you can measure the speed of the element of spring at that location. Measure the distance L (see previous page) between the fixed end and the foil. Record in the table.

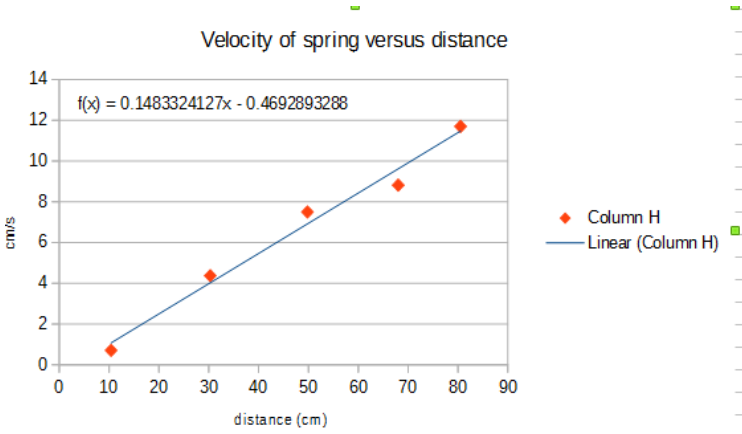
5) See next page. Place the sensor below the plate and let the load oscillate again. Repeat 1 to 3

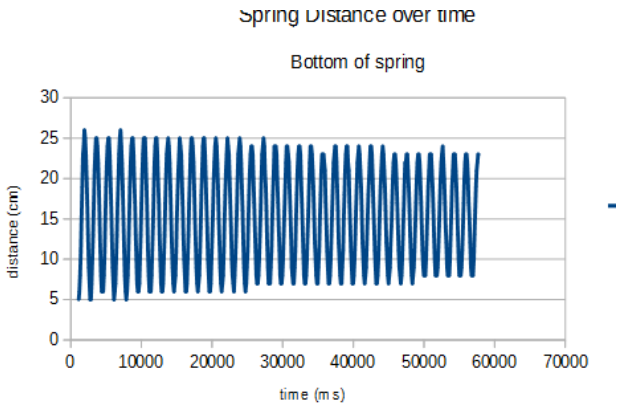
for d(The period should stay the same)



6) Make a scatter graph speed vs length L. You should get a straight line.

Here are the results we found:





3570	25	Velocity	period	distance up spring
3673	25	0.71	1.4	10.4
3776	24	4.375	1.6	30.3
3878	22	7.5	1.6	49.8
3981	19	8.82	1.7	68
4083	15	11.7	1.7	80.5

APPENDIX A: Arduino – IDE and common functions to control the I/O pins – some elements of C – some bitMath

I. Structure of a sketch
II. inclusion
III. Types
IV. digital input/output
V. analog input or analog output
VI. managing time
VII. Other functions (map(), random(), SetPwmFrequency() ...)
VIII. using the serial monitor
IX. some C basics – data structure
X. bitwise operation – tuning hardware register

for I2C and SPI data buses see appendix B.

I. structure of a sketch in 3 parts

1) define the variables, include libraries
2) void setup(){ } this is executed only once
3) void loop {} this is done over and over again as soon as you power the chip.
The main {} is added when the code is compiled (part of arduino.h)

II. inclusion (in the first part)

```
# define LED 2  //C with replace LED by 2. It works like a dictionnary
# define ledON  digitalWrite(LED,LOW)     // elegant way to create short cut
# define pushOn digitalRead(LED) ==0    // to test if the pin LED is on
const byte LED = 2; // good way to define a constant and save space
# include <liquidCrystal.h> // to include a library
```

III. Types

byte : 8 bits (1 byte) - between 0 and 255 – examples: byte a=14; byte a =B00001110; byte=0xFF;
int or short : 16 bits for Arduino Uno (2 byte) -23768 to 32768 – example: int a=-324;
unsigned int or word : 16 bits 0 to 65535 (or 2^{16} -1)
long: 32 bits (4 byte) -2147483648 to 2147483648
unsigned long: (4 byte) between 0 and (2^{32} -1)
float: floating-point numbers. 32 bits. (4 byte) -3.4028235 10^{38} to 3.4028235 10^{38}
void: for a function that does not return any thing
boolean: 1 byte. True is any non zero value. False is 0. example: state=True;
array: examples: int table[4]; byte pin[]={1,4,8,6}; char text[5]=”hello”;pin[0]=1;

char: 8 bits (1 byte) between -128 and 127. You can use it to represent an Ascii character.
Examples: char letter_A=' A' ;or char letter_A=65;
unsigned char: 1 byte. 0 to 255

conversions of types use:
byte() - char() - int() - word() (turns 8 bits to 16 bits) – float()
int myInt = 10;
float myfloat = float(myInt);

scopes:
static int a; // use to save the value of a variable inside a function.
Volatile int a; // to use with the option attachInterrupt()

IV. digital input/output

By default the pins are in an input state and float. Designed to be in a high impedance state.
If pin is used as output then LOW is 0V and HIGH is 5V
If the pin is used as input, LOW is for a voltage < 2V and HIGH is for a voltage >2V
byte LED=13; // using pin 13 as an example
pinMode(LED,OUTPUT); // pin 13 is an output
pinMode(LED,INPUT); // pin 13 is an input
pinMode(LED,INPUT_PULLUP); // the pin 13 is an input and connected to 5V through a resistor so it does not float.

val=LOW; or val = HIGH; // type of val is Boolean
digitalWrite(LED,val); // to write (3.3V or 5V)
digitalRead(LED); // to read the state of the pin
If a pin is an OUTPUT it can produce up to 40mA or it can sink 40mA if it is an input. input. But rhe uC can be damaged by more. https://www.arduino.cc/en/Tutorial/DigitalPins

V. analog output (using PWM mode)

byte LED=3; // using PWM pin 3 as an example.
analogWrite(LED,val); // val = 0..255. Use the PWM (pulse width modulation) . Pins :5,6,3,11,9,10 . The values 0 to 255 can be converted to a voltage between 0 and 5V. type for val is byte. The pin produces a square wave with frequency of 490Hz for most PWM pins except for pins 5 and 6. Pins 5 and 6 produce a 980 Hz square wave. The sensor gets the average value of the square wave. The average is between 0 and 5V. If analogWrite(LED,100) means the average voltage is 100/255 x 5 = 1.96V. In that case the duty cycle is 39%. One cycle of the square wave is high for 39% of the time and low for 61% of the time. If the pin is 3 then the period is 2ms. So the signal is high for 0.78ms.

V. analog input (using a 10-bits converter)

byte analogPin=A3; // using analog pin 3 (or A3)
val=analogRead(analogPin) // val=0..1023. It is a 10 bits number. The pins are A0...A5.
Type is unsigned int. 1023 is 5V by default unless you use a different reference.

Val has to be smaller than the reference.
analogReference(DEFAULT); // It is 5V. You don't have to specify this. 1023 means 5V
the resolution is 4.9mV. When you read 1 more bit it means 4.9mV more.
analogReference(INTERNAL);// the reference is 1.1V. In this case 1023 means 1.1V. Resolution is 1.1/1023
analogReference(EXTERNAL); // use your own reference and connect it to pin AREF. The reference has to be smaller than 5V.

VI. managing time

delay(ms); // in ms . Type is unsigned long. Up to about 50 days. When this counter runs, nothing else can be done by the arduino.
delayMicroseconds();// in microseconds. type is unsigned int. up to 65535 us or about 0.06 seconds.
Millis(); // type is unsigned long. Returns the number of milliseconds since the Arduino board began running the current program. Maximum is 50 days.
Micros();// type is unsigned long. Returns the number of microseconds since the Arduino board began running the current program. Maximum is 70 minutes, resolution is 4 microseconds.

VII. Other functions

min(x,y) / max(x,y)
constrain(x,a,b)
abs(x)
val2=map(val, fromLow, fromHigh, toLow, toHigh) // example map(val1,0,1023,0,5)
to map the value val1 from the scale 0 to 1023 to 0 to 5V. So val2=5 * val1 /1023.
The digital 10-bits value between 0 and 1023 is mapped into a value between 0 and 5V (voltage).
So val2=5 * val1 /1023. We can also map a value from [0,1023] to [0,255].
sqrt(x) //function square root
sin(x) … //trig functions
pulseIn(); // to measure time between HIGH to LOW or LOW to HIGH.
See https://www.arduino.cc/reference/en/language/functions/advanced-io/pulsein/

randomSeed(); followed by **random(); //** to generate a random number between -2,147,483,648 and 2,147,483,648 . We need to seed the function like: **randomSeed(analogA0)**. In that case, don't use pin A0. The pin floats (because of static electricity) and generate random numbers. We can also write random(0,10) for a random number between 0 and 9. Or **random(255)** to generate a number between 0 and 255.

lowByte() , highByte() , bitRead(), bitWrite(), bitSet(), bitClear() // to manipulate bits

attachInterrupt(), detachInterrupt(), noInterrupt() // This is to tell the processor to immediately stop what it was doing and handle some high priority processing based on external interrupt pins (only pins 2 and 3) . You need to introduce a special function called the Interrupt function that tells arduino what to do. Visit https://www.arduino.cc/en/Reference/AttachInterrupt.

setPwmFrequency(pin,divisor) // To change the frequency of the PWM pins.
See http://playground.arduino.cc/Code/PwmFrequency. Example setPwmFrequency(10, 1024); pin 10 is controller by the timer 1 which counts 510 bits one bit at a time. If the divisor is 1024 we get a frequency of 30Hz. 16,000,000 /510 is the frequency for a divisor of 1. if you divide by 1024 you get 30Hz. See lab4 about this function and appendix B for more on timers.

shiftOut(DATA,CLOCK,MSBFIRST, B_number)// to send a binary number to a shift register. DATA is the arduino pin number connected to pin14 of the 74HC595 IC. CLOCK is the arduino pin number (usually 10) connected to pin 11 of the IC. MSBFIRST means the most significant bit is sent to Q7 and the least to Q0. So B11010001 means Q7 is 1, Q6 is 1, Q5 is 0 … Q0 is 0. The first bit always go to Q7. B_number is the number between 0 to 255 that will be converted to a binary number.

VIII. Using the serial monitor

Serial.begin(speed); // write this in setup(). Speed is the bauds rate. You usually use speed=9600 (bits/ seconds). But you can use another speed. https://www.arduino.cc/en/Serial/Begin
Serial.end();// deactivate the communication with the serial port. So we can use pin 0 and 1 again.
Serial.available); // read number characters available in the buffer (128 max)
Serial.Read();// read the first character in the buffer. If it returns -1 , that means there is no character. The character read is removed from the buffer and the next one can be read.
Serial.flush();// flush the buffer
Serial.print(); and **Serial.println();**// to print on serial monitor. Examples: Serial.print(75,DEC) Serial.print(75,BIN); Serial.print(75,HEX); Serial.print(75,BYTE); Serial.write(); // write in binary

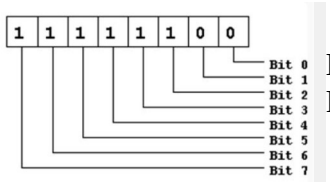
IX. C programming for arduino basics

A) logic
x==y
x!=y (not)
x<y smaller
x<=y smaller or equal
(expression 1) || (expression 2) means OR
(expression 1) && (expression 2) means AND
!(expression) means NOT
 	
C) loops
while (condition) {}
while(1) {} //never stop
for (byte i=0; i<Max; i++) {}
break; // leave the loop
continue(); // stay in the loop but skip the rest of instructions
goto label1; // with label1: ….;
	B) if , switch()
	if (condition) {}
	if (condition) {} else {}
	
	switch(val) {
	case value1:
	…
	break;
	case value2:
	…
	break;
	default:
	…
	break;
	
	D) arrays
	int my_array[4]; // the array has 4 spaces.

E) anatomy of functions
int myFunction(int x, int y) {
int result;
result=x+y;
return result;}
// if the function does not return anything:
void myFunction(int x) {
digitalWrite(3,HIGH);
delay(x) ;}
	my_array[0] is the first element
	float temperatures[]={49. 60., 30., 40., 35.,25.};
	//initial values are given
	temperatures[1]=30.;

X) bitwise operations – tuning hardware registers
PART A
Operators: & (and) | (or) ^ (xor / exclusive) ~ (not) << (shift bit left) >> (shift bit right)

See here for more: http://playground.arduino.cc/Code/BitMath



Note: In a 8 bits register (memory cell of 1 byte) the bits are labeled 0 to 7. From right (least significant bit) to left (most significant bit) .

example 1: & (and)
if a = B01101100; b=B11000111; Find val such as val = a&b ; Answer: val=B01000100

You multiply bit by bit:
a= B01000100
b= B11000111
_____
val=01000100 (multiply bit by bit such as 0x1=0;1x1=1;0x0=0)

Note: Usually we use hexadecimal notations when coding
https://www.rapidtables.com/convert/number/binary-to-hex.html
so a=0x44; b=0xC7 .

Example 2: ~ (not)
if a = B01101100; Find val such as val=~a; Answer: val=B10010011 (0x93)
You flip each bit. 1 becomes 0 and 0 becomes 1.

Example 3: | (or)
if a = B01101100; b=B11000111; Find val such as val=a|b; Answer :val=B 01000111 (0x47)
You add bit by bit:
a= B01000100
b= B11000111
_____
val=01000111 (you add bit by bit such as 0+1=1;1+1=1;0+0=0)

Example 4<< (shift bit left)

if a = B001101011; Find val=a<<4; Answer: val=B010110000
This means that 4 zeros enter from the right and push all the bits to the left. Bits that overflow at the left are lost (they fall).

Important If a = 00000001 we can write a=1 (in decimal form).
Then 1 <<4 means 00000001<<4 and we get 00010000 which means that bit # 4 is set to 1.
This is important when coding uC. It is used to set a bit to 1 (Each bit is a switch set to on or off).
1 <<3 = 00001000. 3 zeros pushed the 1 to the left.
(1<<6) = 01000000 (0x40 or 64)

Example 4>>(shift bit right)

if a = B001101011; Find val a>>4; Answer: val =B000000110.
4 zeros enter from the left and 4 bits (at right) fall.
Note: if the most significant bit is 0 then the zeros enter from left. if most significant bit is 1 then ones enter from left (because if the number is signed then 1 means a negative number).

(1<<6) means 01000000 with 00000001=1 (decimal form)

Example 5<< ^ (xor) → toggling a bit

a=B01101101 ; How to toggle bit 4? ; Answer: a ^ B00010000 (=01111101)
The bit 4 is toggled. This is useful to blink a pin of a uC.
We write instead (01101101) ^ (1<<4) for short to toggle bit 4
(01101101) ^ (1<<6) to toggle bit 6 (=00101101)

PART B: tuning hardware registers

Those operators are used to tune the hardware registers of a microcontroller (uC). Hardware registers are series of 8 bits (so a byte) (in a 8-bits uC) that are used to control the peripherals of a uC. Think of them as a series of switches. Each bit (each switch) controls a function of a peripheral (like turning on or off the pin13. Enabling the I2C serial communication. Starting an analog to digital conversion etc…). A 1 means means a voltage of 5V (HIGH). A 0 means 0V.

The operators are used to:
clear a bit (bit=0) in a register. **Set a bit** (bit=1) in a register. **Test a bit** (is it 0 or 1). **Toggle a bit** (0 to 1 or 1 to 0).

testing a bit use &:

The operation (B01101101 **& B00010000**) is testing if the bit # 4 is on (1) or Off (0).
If the statement is True then the bit is on. Usually we write: B01101101 **& (1<<4)**
because (1<<4) = 00010000. For example. If a register's name is PORTC and you want to check if the bit 4 is on (could be attached to an arduino pin) then write: **if (PORTC & (1<<4) {do whatever}**

clearing a bit use &:

(B01111101 **& B11101111**) is turning off bit 4. Answer = B01101101 (multiply bit by bit)
Usually we write: B01111101 **&** (~1<<4) because (~1<<4) = B11101111 with 1<<4 = B00010000
If a register name is PORTD then write: PORTD= PORTD **&** (~1<<4) that will clear bit 4/

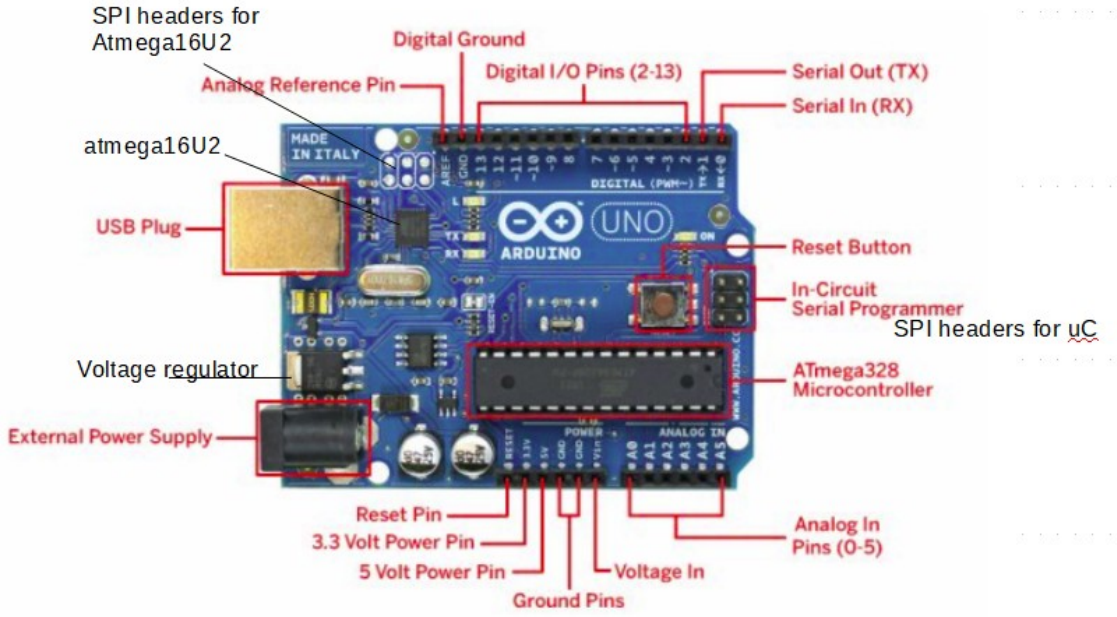
setting a bit use |:

B01101101| **B00000010** = B01101111 is setting the bit # 1 to 1 (or HIGH). To switch on bit 1.
If you want to set bit #3 to 1 in a register called PORTB then write: PORTB= PORTB | (1<<3)
Remember (1<<3) =00001000

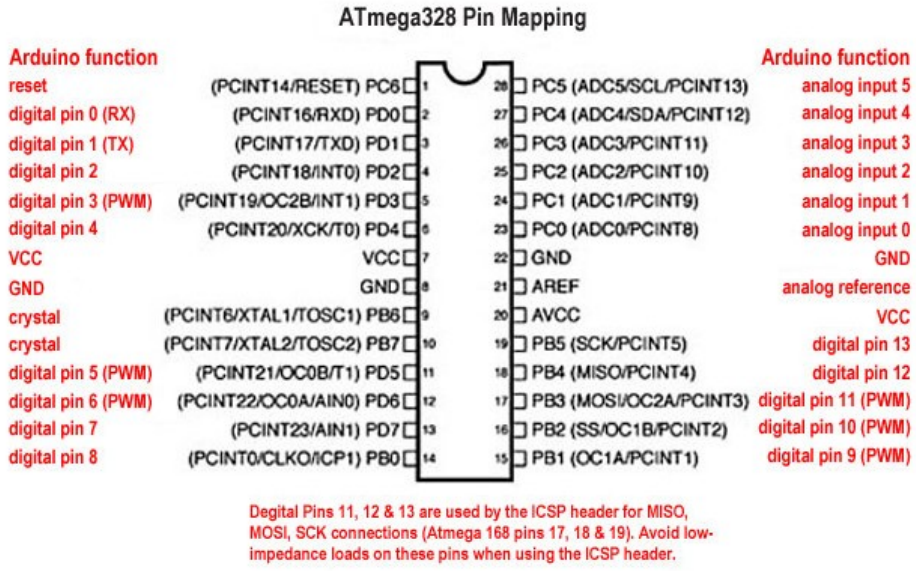
toggling the bit use ^:

B01101101 ^ B00010000 = B01111101 to toggle bit 4
If you want to toggle bit #3 in register PORTB with XOR (exclusive OR)
PORTB = PORTB ^ (1<<3) So if the bit was 1 it will be 0 and if it was 0 it will be 1

APPENDIX B: Arduino Uno R3 – Atmega328 – Characteristics



https://www.arduino.cc/en/Main/ArduinoBoardU



pin mapping for the microcontroller ATmega328
Source: http://www.jameco.com/jameco/workshop/jamecobuilds/arduinocircuit.html. They will show you how to build your own development board and will provide the list of the parts including the uC with a bootloader installed (bootloader by arduino).

Arduino Uno is a open source development board. The hardware is open source and the IDE (software) is open source. The development board includes the uC, I/O connectors and additional circuitry (fuse to protect the PC, voltage regulators, circuitry to select source of power, LEDs, decoupling capacitor (low pass filter) to absorb noise generated by the processor, diode to protect the barrel connector, singnaling LEDs.., more capacitors, resistors)

Below we will cover:
- ATmega328 – memory - LOW vs HIGH – Power – max current - I/O pins – IDE – USB to UART bridge – Bootloader vs ISP headers – clock – Hardware registers – I/O PORT registers – ADC – Interrupts – timers – PWM – serial communication – reference for the pins

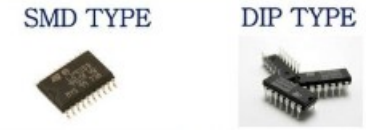
ATmega328 the microcontroller (uC) on Arduino Uno.
2KB RAM memory (data memory) – 32KB flash memory – 2KB EEPROM
Speed is 8MHz if the uC is used as a standalone. 16MHz on the platform Arduino Uno.

The microcontroller (uC) is the chip 8-bits ATmega328. From Atmel (now microchip)
https://www.microchip.com/wwwproducts/en/ATmega328

The uC is a computer system in a small chip. It has a CPU, clock, peripherals. The uC has several peripherals/modules. This includes 3 modules for serial communication with the outside world (SPI, I2C/TWI, UART), 3 timers (two 8-bits timers and one 16-bits timer), one 10-bits analog to digital converter (ADC), a watchdog timer, an Interrupt module and I/0 module for the digital/analog pins.

AVR has different families but they all have the same architecture (harvard architecture). AVR stands for Alf Vegard Risc processor). Alf and Vegard are the names of the two students who developed the uC at the Norwegian Institute of technology (NTH). Risc is a type of processor (architecture for processor) that runs one instruction per clock cycle. It is a faster architecture but can not deal with a lot of instructions (https://www.studytonight.com/computer-architecture/risc-cisc-processors) The harvard architecture allows the CPU to exchange data with the flash memory and with the data memory at the same time. The data don't use the same bus.

The packaging for the chip can be DIP (dual in line package) . Or it can be a surface mounted device (SMD package). You can remove a DIP chip and replace it with another one. If you make your own development board then you buy a DIP chip and experiment with it.



With the SMD there are 4 extra pins. 2 for extra power and Gnd pin and 2 extra analog pins out (ADC6 and ADC7) but they are not pinned out on the Arduino board.

Arduino UNO hasan embedded attached to LED 13 that you can blink to check your arduino board. (see File→examples→basics→blink). Pin 13 is already connected to a 1K resistance so no additional resistance is necessary.

Memory

- **The data memory (SRAM) is 2K**. It includes 64 I/O registers (8-bits cells) to control the peripherals (including the I/O pins). 32 general purpose registers used by the ALU (personal use). The rest of ther egister is for the use of the code uploaded (heap/stack). Each register in the RAM can be accessed by an address.

I/O registers are also called Special Functions registers. are used to control the peripherals (like tmers, pins, ADC .., I/O pins) and the pins I/O. Some of those registers are also exclusively used by the ALU (stack pointer, program counter, status register)

The 32 general pupose registers are used by the ALU. The registers are names R0 to R31. The last 6 registers are special pointers (called X,Y,Z) 16 bits long used by the compiler and point to parts of the SRAM.

- **The flash memory is 32KB**. The registers in the flash memory are 16 bits long. 32 X 1024 bytes /2 = 16,384 addresses. The flash memory is accessed by blocks. 1024 registers are used for the bootloader (2KB) if you are using arduino (the board). The bootloader is a firmware installed in the uC by Arduino (see below). The firmware simulates a programmer that " catched" the code coming from the computer through the USB interface. The flash memory is accessed by blocks. 42 registers are reserved for configuration of interrupt instructions. 3 of the registers are called fuses (It's just a name. They are not blowing up). The fuses set the clock speed, which clock to use (internal or external), set the brownout detection, set the watchdog timer ... Those fuses can not be changed by software. They were configured by the bootloader burnt by Arduino in the uC. You need to use a programmer and use the SPI headers to reconfigure the fuses but you would lose the bootloader. This is advanced stuff.

- **1KB EEPROM** memory. To save data even when the power is off.

LOW versus HIGH

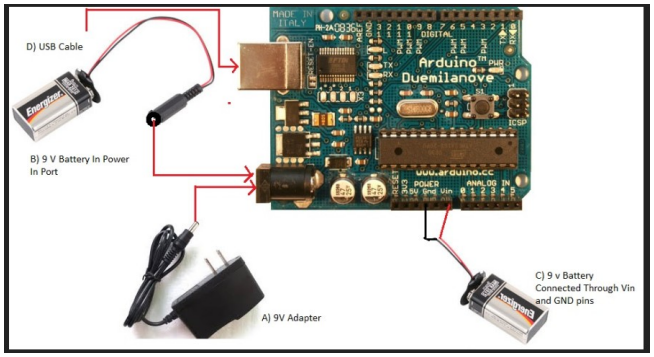
When reading an input a voltage smaller than 1.5V is a LOW (digital 0) for the 5V board (Arduino Uno) or smaller than 1V for a 3.3V board. A High is 60% of the Vcc. For arduino uno Vcc is 5V so a HIGH is a voltage greater than 3V. (1.98V for a 3.3V board).

Power

The uC can be powered with a voltage between 1.8V and 5.5V and should not exceed 6V. The power affects the speed of the clock (maximum is 8MHz for the uC alone without external clock). The higher voltage supplies allow for the faster clock rates. To get the 20MHz you need an external clock and a voltage of at least 4.5V. Arduino runs at 16MHz (to be compatible with previous versions). The development board Arduino has a voltage regulator (LM7805) that will turn voltage between 7V and 12V to a regulated 5V (+/- 5%). The voltage can be provided using the barrel connector (for a 9V battery/ AC to DC charger) or using the pin Vin. You can by pass the voltage regulator by providing directly 5V to the 5V pin but this is not recommended. You can power the board through the USB cable instead. And the USB cable is supposed to provide 5V to the board. Although sometimes the voltage varies. The board has a resettable fuse to make sure the current drained from the PC is limited to protect the board (the USB driver in the PC is supposed to limit the current as well but not all of them do). If there is too much current, the fuses heat upand and it resistance increases limiting the current. The fuse has a limited life time. The board has also a 3.3V voltage

regulator and can source 3.3V at one of its pin/ The USB cable is also used for Serial communication with the PC. The board has a smart circuitry to compare the voltage coming from the barrel connector and the voltage coming from the USB cable. If the barrel provided enough power (5V) then the board uses it and not the USB cable. The hardware used a transistor as a switch and an analog comparator to compare the voltages.

Here is a picture that shows different ways to power Arduino. You can also use the pin Vin and Gnd to power it. Source: https://www.codeproject.com/Articles/845211/Stage-Complete-Beginners-Guide-For-Arduino-Hardware. Do not use the pin Vcc. It reads the Power provided to the board.



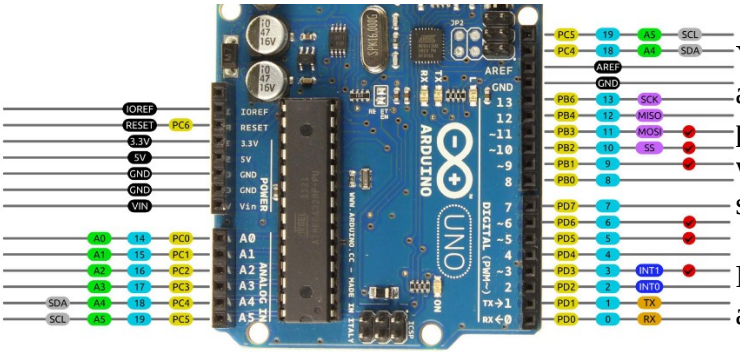
Maximum current

The maximum current that can be drained from the pins is 40mA. The maximum for the whole board is 200mA. The maximum current from the 5V power pin is 400mA if Arduino is connected to the USB cable. The 3.3 V power pin can provide 150mA. Input pins behave like a sink and can not sink more than 40mA. All the input pins working together can not provide more than 100mA. Actuators (like motors) that need more current need another power source and you can use arduino as a switch using a transistor.

Source:http://electronics.stackexchange.com/questions/67092/how-much-current-can-i-draw-from-the-arduinos-pins

I/O pins

Arduino has 20 pins. 14 digital input/output pins (of which 6 can be used as PWM outputs), 6 analog inputs**.** http://playground.arduino.cc/Learning/Pins has a very useful quick reference / table that shows the properties of each pin. Which pin is analog, which pins are used for the serial communication I2C (TWI) or SPI.



You will find this table at the end of this appendix. Some pins are used for different purposes or are duplicated. You have to know which pin is doing what. Below is another schematic.

Pins 18 and 19 are duplicate of A4 and A5. They are used for the I2C protocol.

Pins 0 and 1 are used for serial communication with the computer through the USB cable (UART

serial communication) You can not use them if data are transferred. Pins 10,11,12,13 are used for the SPI protocol.

IDE

editor → sketch in C++ → cross-compiler → avrdude → bootloader → flash memory

We will use the IDE provided by arduino to program the microcontroller. The IDE is a package including the editor, the cross-compiler (gcc developed by the GNU project) and a serial monitor. The IDE installer installs a driver so data can be sent through to the USB cable to the **Atmega16U2** (the chip to bridge between USB cable and the uC UART module). With the IDE you also bypass the flash programmer (that uses the SPI headers) and it simplifies a lot the compiling and uploading process. An Arduino board has a chip (uC) with a bootloader (written by Arduino) and this firmware will emulate a flash programmer like the STK500. The bootloader "pretends" to be a programmer that uses a programming protocol (avrdude) and will collect the code sent through the USB cable and will flash the code into the memory. Compiling is done by avr-gcc, a cross-compiler (C++ compiler) for the AVR chip (the IDE takes care of that). The sketch is linked to the libraries and to other files (see arduino\hardware\arduino\avr\cores\arduino) during the compiling process and files are converted to an executable code in an hexadecimal form later unpacked into a binary form by Arduino. Loading the program to the AVR uC is done by avrdude a very handy but not friendly utility (program). Avrdude is a software uploader that feeds the compiled machine to the flash programmer. The software specifies the communication PORT you are using, the target (type of uC), the speed of transmission.. All those steps are not easy and this is done under the hood by the IDE. https://github.com/arduino/Arduino/wiki/Build-Process

A flash programmer (bootloader here but could be a programmer) works by grounding the RESET line which halts the CPU and signals the AVR to start listening on the serial line (or SPI bus if you are using a real programmer). The bootloader takes 500 bits of memory but make the uploading of the sketch very easy. It listens on the serial port and process any sketch uploaded. It also sets the clock speed (16MHz).

You can do the flashing in more advanced ways using an editor for the C code, a compiler (avr-gcc), a programmer + avrdude. In this case you can even read the data back out of the AVR's memory.

The IDE provide the user with many pre-built commands and makes it very easy to program without knowing C or registers management. So the Arduino IDE make it easy to control the peripherals (to interact with the outside world) without dealing with the registers. The language is C++. Also many libraries have been developed for the IDE by an enthusiastic community (SD card, clock, EEPROM, sensors etc..). The IDE has a serial monitor attached to it so you can send data from the monitor or send data to the monitor. You can also use a different serial monitor like CoolTerm or TeraTerm You can also use python to "catch" serial data sent by the the uC using the library pyserial.

Incidentally, you can use a protocol called Firmata that let you control Arduino from the host computer using Python, Java, Processing or a windowns 10. You just need to upload a Firmata program in Arduino using the IDE (example → firmata). Arduino becomes an extension of the host computer and you can control it with another language.

Note: Instead of using the arduino IDE you can switch to Atmel studio 7 to program the chip via the USB cable using the bootloader. Without a flash programmer. See: https://www.youtube.com/watch?

v=zEbSQaQJvHI&t=4s. A previous video by the same author explains why he switched. If you use the Atmel studio7, then you have to learn registers management (switching bits in registers to control peripherals). A Register is a memory unit of 8 bits and each peripherals of the chip has registers to control it.

ATmega16U2 a USB to UART converter / USB bridge (the FTDI chip was replaced by ATmega16U2)
The chip emulates a serial port. The development board of Arduino has a chip (Atmega16U2) in addition to the uC so that you don't need an additional programmer in order to upload a sketch into the uC. That chip works with the bootloader (firmware in the uC that simulates a programmer) in order to "catch" the data from your editor (IDE) and send it to the uC through a serial bus. The bus has 2 lines. Tx and Rx. Data can be bused the other way and sent to the serial monitor included in the IDE. The lines connect the bridge to the UART peripheral included in the uC.

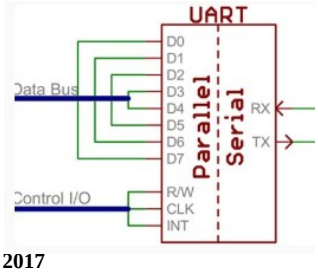
Data are bused through the lines Tx/Rx:
(IDE running on computer)→ 16U2 → (UART module of the uC Atmega328)
(UART module of the uC) → 16U2 → (IDE)

This chip is a bridge between the USB PORT and the UART module (peripheral) included in the uC. The UART module processes the data coming in or coming out through the Tx/Rx buses.

The 16U2 firmware uses the standard **USB** COM drivers, and no external driver is needed. However, on Windows, a ArduinoUno.inf file is required (inf will be installed automatically now).

The first Arduino had a nine-pin serial RS-232 connector like the old PC with a RS232 (+/-13V) to TTL (0 to 5V) converter.

UART module (asynchronous serial communication) inside the uC
Serial data are transmitted from PC USB PORT → USB – UART bridge → UART module in the uC
This peripheral is a serial port in the uC and is used in asynchronous mode. It does not require or provide a clock signal. The USB cable is used to upload the code (called sketch) / into the arduino and to exchange data through the serial bus. The serial bus has 2 lines. One line to send data from computer to arduino (RX line) and one line to send data from arduino to the computer (TX line). When the computer communicates with the micro controller, it uses the pins 0 and 1. pin 0 is called the RX pin (receiving serial data) and pin 1 is called the TX pin (transmitting data). When pin 0 and pin 1 process the serial data they can not do something else like reading sensors. Data are sent through the USB cable by 8-bits chunks. The serial bus has a buffer of 128 bytes. The USB cable contains also a power line (5V) and the Gnd line. The interface between the USB cable and the microcontroller is controlled by the USB to UART (Universal Asynchronous Receiver/Transmitter) bridge Atmega16U2 (see next section). So the data travels from the IDE to the bridge to the UART peripheral of the microcontroller. The UART module/peripheral (inside the microcontroller) acts as an intermediary between parallel and serial interfaces. On one end of the UART is a bus of eight-or-so data lines (plus some control pins), on the other is the two serial wires - RX and TX connected to the bridge: https://learn.sparkfun.com/tutorials/serial-communication

to CPU of uC 

to Atmega16U2

Uploading a code - Bootloader (508 bytes of the memory) vs ISP (In System Programming) headers (sometimes called ICSP headers)

It is a small program (firmware) flashed by Arduino in the uC (you can buy a standalone chip already flashed with the bootloader). The bootloader and the IDE is what makes Arduino so easy to use. You can program the uC without using a programmer. The bootloader is there so you can flash the code (sketch) through the USB cable through serial communication (Tx Rx lines) instead of using a special programmer that uses the SPI bus (protocol to send data through serial lines). Arduino has an USB serial converter chip (Atmega16U2). It also set the speed of the processor and set other registers (https://www.baldengineer.com/arduino-bootloader.html). It is the easiest way to flash code in the chip.

You can bypass the bootloader and flash code using the ISP headers using the SPI bus (a protocol to send serial data) and a programmer. In that case you don't need a USB cable and a USB to serial converter (Atmega16U2 chip) . But you need a programmer (see below). The ICSP headers are connected to the MOSI MISO SCK (for SPI protocol) RESET GND and VCC pins. When using those pins you don't need a bootloader flashed in the chip. You use those headers (SPI bus) if you need to flash a bootloader in the uC. Warning: If you flash a code using the SPI pins instead of using the USB connection then you **overwrite the bootloader** and wont be able to flash a code with the USB connection anymore (voice of experience). A programmer (from atmel) can be used to flash a code through the headers but there is also a way to turn an arduino into a programmer in order to program a standalone https://www.arduino.cc/en/Tutorial/ArduinoToBreadboard. This method is used to flash a code in a standalone uC. You can buy a uC chip with the bootloader already installed by Arduino. Then you can easy build your own development board an add a breakout to bridge between the computer and the uC through an USB cable (using a FTDI breakout).

Clock

The system click is the heartbeat of the entire uC. The development board Arduino uno has an external 16 MHz quartz crystal (connected to the uC). A ceramic can be used instead when building your own board (less accurate and cheaper). The pins XTAL2 and XTAL1 of the uC are used for that. Two capacitors of 22pF are part of the set up to make the crystal works. The start the oscillator circuit at power-up because they store energy. That means it can process data at a rate of about 16 million data per seconds about. The crystal is used for the timers or as counters. Timers are, for example, necessary to convert digital data to analog data using the pulse width modulation (PWM). Arduino boosts the uC to 16MHz with a 5V power supply. If the power supply is less the clock slows down.

The chip (uC) has also an internal clock (RC circuit) of 8MHz (to be adjusted in software) if you use it has a standalone without an external clock. But you can not use this clock with Arduino unless you upload a different bootloader. The bootloader is for a 16MHz external clock. There is also another internal RC circuit with frequency of 128KHz that is attached to the watchdog (deals with interrupt and with resetting the uC). You can use this clock but not with the bootloader of ARduino. Note that when you buy the uC chip alone the clock is at 1MHz to save power and some sketch has to be uploaded in order to change it to 8MHz (you use prescalers).

With arduino, you don’t have to worry about that. The clock is set at 16MHz and it is easy to slow it down using prescalers (between 1, 2,4,8,16,32,64,128,256). If you set a prescalers to 256 then you get a 62.5KHz clock.

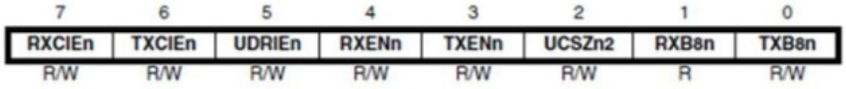
Reset

There is a reset button to start over the execution of the code. Usually, it has a bar over it because its an active low input. The RESET function occurs when the input is taken low. You can connect the reset pin (there is a reset pin) to gnd with a jumper to restart the program ir you can use the push button. The reset pin of the uC is connected to a 10K pull-up resistor (so reset is HIGH) so the reset pin does not float and the uC does not reset randomly. It is connected to the DTR line from the serial port (USB to UART) through a 0.1uF capacitor. So when the reset is HIGH the DTR connection does not pulls it down. When the RESET is connected to LOW, the DTR line is LOW to signal the bootloader to listen and to upload a new sketch. When the uC is reset, the registers are cleared, pins are 0V. Without bootloader, the function causes the chip to enter In-System Programming (ISP) mode, or low Voltage Serial Programming mode (LSVP) mode. The uC is reset when you power the uC. After reset, the logic state of all port pins is undefined, since they are inputs with no internal pull-up and high impedance. It is more susceptible to be damaged by static discharge than other pins.

Hardware registers. The IDE hide the register manipuation behind the hood. Beginners don't have to deal with setting the registers. Arduino has many commands that hide the registers tuning.

AVR has macro defined in io.h to help the user with register manipulation
https://www.microchip.com/webdoc/AVRLibcReferenceManual/ch20s22s02.html

A register is a memory unit of 8 bits or byte. They are the building blocks of memory. But you have special function registers or hardware registers (part of the flash memory) and they control the AVR’s built-in peripherals (inside the uC). Those registers I/0 pins, the clock, the serial communication modules (I2C, SPI, UART), the ADC (analog to digital converter)... Registers are slots in the AVR’s memory. Each bit in a register (byte) is a switch that enables or disables some functions of the peripherals. Setting up a hardware to do its job is a matter of figuring out which switch you need to set as a 0 or 1. Those hardware registers are special memory locations. The locations are hidden from you in a io.h file and are given mnemonic names like DDRB or “ data-direction register B”. That register is to control the pins I/O from PORT B (pins 8 to 13 of arduino) and sets if the pin is an input or output. You can treat registers like variable.
For example DDRB=4 (00000100 in binary form). So the register sets pin 8 and pin 9 as input (bit=0), pin 10 as an output (bit=1) etc.. .
Another example are the registers for the serial communication module (UART) that process serial communication (with 2 lines Tx and Rx) between the chip and the computer or another device. Two registers UCSR0B and UCSR0C) are configuration registers. UCSR0A controls reception and transmission of data and checks if the hardware is ready to transmit/receive. Another register contains the data received/sent UDR0. Another has the baud rate etc.. Each bit of the register has its own mnemonic name. The registers are listed in the datasheet of AVR uC. Here is an example below. UCSR0B This is a configuration register for UART. n=0 for arduino uno that has only one UART peripheral.



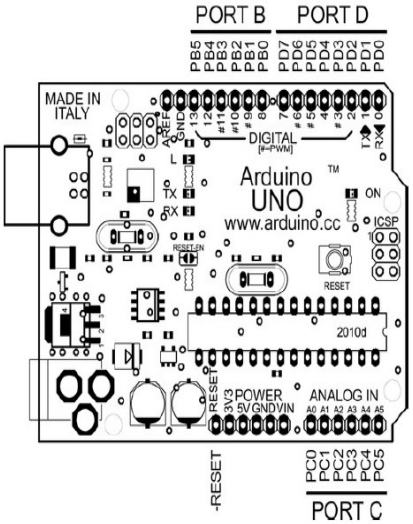
7	6	5	4	3	2	1	0	
RXCIEn	TXCIEn	UDRIEn	RXENn	TXENn	UCSZn2	RXB8n	TXB8n	UCSRnB
R/W	R/W	R/W	R/W	R/W	R/W	R	R/W	

Each bit is a switch and has a special function.

The bit 7 (RXIE0) enable reception of data. The bit 6 (TCIE0) enables transmission.
The register UCSR0A (another control register) checks if the hardware is ready to receive or transmit. The bit UDRE0 (from UCSR0A) is 1 when it is ready to transmit. The bit RXC0 is 1 when the hardware is ready to receive. The baud rate is a 16-bit number so it takes 2 registers of 8 bits to store it (UBRR0L and UBRR0H). The register UBRR0 contains the divisor to divide the CPU clock (runs at 16Mhz) so it can run between 9,600Hz and 115,200Hz. UCSR0C has bits to set 8 data bits to be send and to enable the 1 stop bit.
So the good news is that the IDE hides all that for us. Here is an example of what the IDE hides from us when transmitting/receiving data through the serial bus: https://appelsiini.net/2011/simple-usart-with-avr-libc/

I/O pins and the parallel ports – port manipulation and bitwise operations (see APPENDIXA)
Again this is hidden when using the IDE. However, using PORTS instead of the commands save memory.

Hardware registers DDRx, PORTx, PINx (x=B,C,D) control the I/O pins: Are they input or output? (DDR registers) If they are output, are they writing a 0 or a 1 (PORT registers) ? Are they reading a 0 or a 1 (PIN registers) ? If they are input pins are you setting up the pull-up resistors?



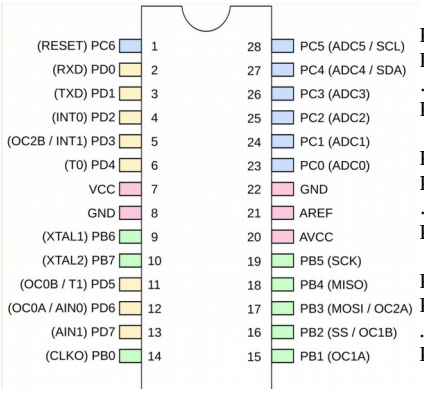
Reference picture: Arduino internals – Dale Wheat

The pins are arranged in 3 ports (below). They are parallel ports. Data are processed as packs of 8 bits.

https://www.arduino.cc/en/Reference/PortManipulation. See also below the pin mapping diagram. The pins are grouped in ports (Port C, port B and port D) of 8 bits (the µC is a 8 bits computer).They are parallel ports. PORT B (control arduino digital pin 8 to 13 + pins for the crystal) , PORT C (control analog input pins A0 to A5 + 1 pin for reset), PORT D (digital pins 0 to 7).

Pins A0 to A5 can be referred as pins D14-D19

AVR uses mnemonic names for each pin.
Reference for picture below: AVR programming – Elliot Williams.



PD0→ pin0
PD1→pin 1
…
PD7 → pin7

PC0→ pin A0
PC1→ pin A1
….
PC5→ pinA5

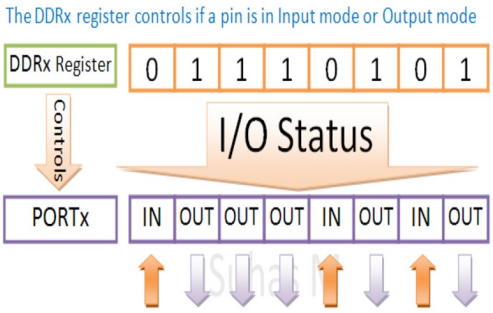
PB0→ pin 8
PB1→ pin 9
...
PB5→ pin13

Every I/O pin of the board is connected to a pin of the chip atmega328.

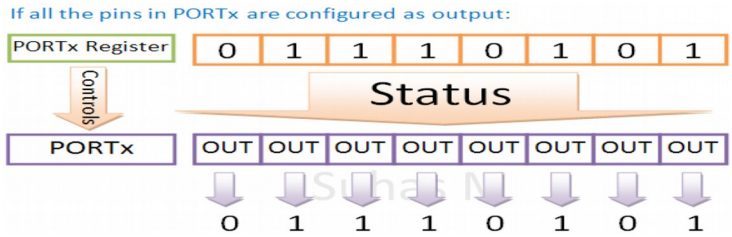
Each bank of pins (B,C,D) called PORT has 3 hardware registers associated with it. If you don't use the arduino IDE functions (appendix A) then you have to work with those registers. You can work with registers using the IDE editor which save memory.

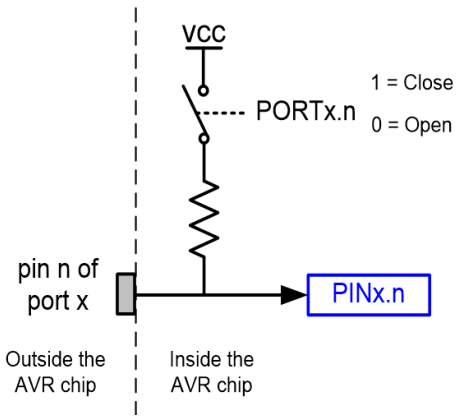
The registers to control the I/O pins are:

DDRx (x is B, C, D) → to set pins as input or output. 0 is input and 1 is output. By default the pins are input (so the register contains only 0). For example. DDRD=0x00110000 means pin 5 and 6 are output and the pins 0, 1, 2, 3, 4, 7 are input. But we write instead DDRD= DDRD| 0x00110000 (or even better DDRD=DDRD|(1<< 5)|(1<<6) so we don't disturb the other bits. (see appendix A bitwise operation). Because some bits are not attached to I/O pins. For example port B has 2 pins attached to the clock so you don't want to mess with it. DDRB=DDRB|0x00010001 means pin 8 and pin 12 are output.



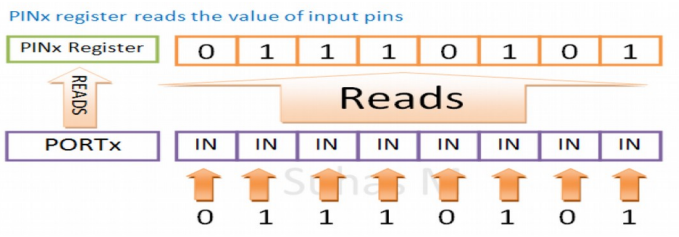
PORTx → A) if the pin is an output (as set by DDRx). We can set a pin HIGH(1) or LOW (0) using this register. For example, to lit an LED at pin 2: PORTD=PORTD | 0b00000100 (or PORTD=PORTD | 1<<2)



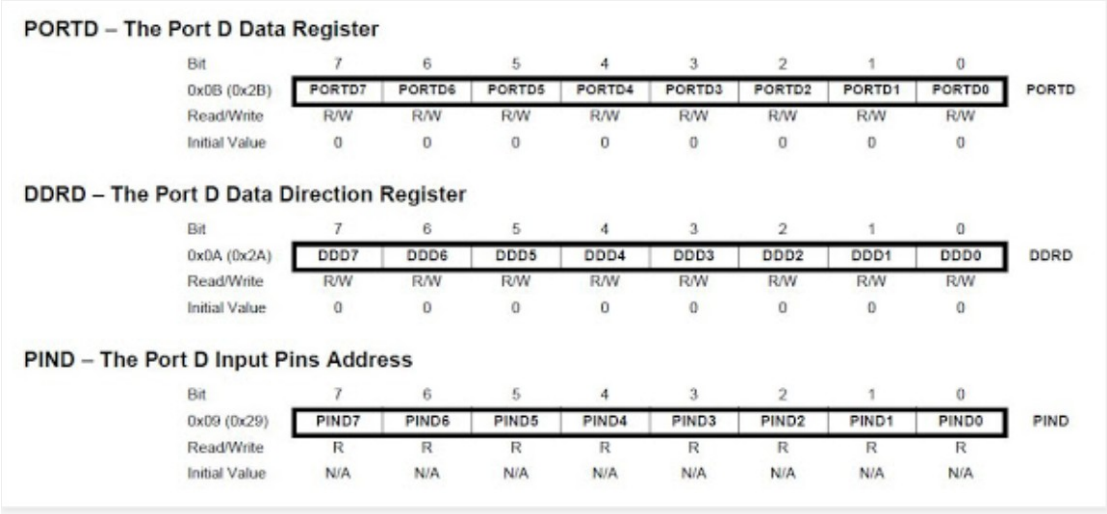


→ B) if the pin is an input (according to DDRx) then 0 means deactivate pull up resistor. 1 means activate pull up resistor.
A 1 means that the input pin is attached to 5V through an internal resistor (about 30K) so it does not float. So by default the pin is a HIGH. (5V- resistor – pin)→ outside world. A 0 means that the input pin is let to float when nothing is connected to it. A pull-up resistor allows the signal to remain in a known state when no external circuit is attached.

PINx → to read the value of an input pin 0 or 1. For example. PINC=PINC&0b00001000 means testing the pin A3. If this is true then the input is 1 (maybe you pushed a button). We can also write
PINC=PINC & (1 <<3)



Here are the registers that control PORTD (from datasheet). Each bit can be 0 or 1.



PORTD – The Port D Data Register

Bit	7	6	5	4	3	2	1	0	
0x0B (0x2B)	PORTD7	PORTD6	PORTD5	PORTD4	PORTD3	PORTD2	PORTD1	PORTD0	PORTD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

DDRD – The Port D Data Direction Register

Bit	7	6	5	4	3	2	1	0	
0x0A (0x2A)	DDD7	DDD6	DDD5	DDD4	DDD3	DDD2	DDD1	DDD0	DDRD
Read/Write	R/W	R/W	R/W	R/W	R/W	R/W	R/W	R/W	
Initial Value	0	0	0	0	0	0	0	0	

PIND – The Port D Input Pins Address

Bit	7	6	5	4	3	2	1	0	
0x09 (0x29)	PIND7	PIND6	PIND5	PIND4	PIND3	PIND2	PIND1	PIND0	PIND
Read/Write	R	R	R	R	R	R	R	R	
Initial Value	N/A	N/A	N/A	N/A	N/A	N/A	N/A	N/A	

For example consider the PORT D.

DDRD=B11111110; means only pin 0 is an input and pins 2 to 7 are output. PORTD=B10101000; means pins 3,5,7 are high. DDRD=DDRD|B01000000; means don't change anything except for pin 6 that needs to be an output. It is the same as DDRD=DDRD|(1<<6); PIND=PIND&(1<<6) is to test if the pin 6 is on or off. If TRUE it is on (HIGH).
DDR=0x00; means all pins in PORT A are input
PORTD=PORTD &B10111111 means means pin 6 is LOW (output). Don't touch the other pin.
(same as PORTD=PORTD & (~ 1<<6))
See more example here: http://www.elecrom.com/avr-tutorial-2-avr-input-output/
See also http://thepiandi.blogspot.com/2016/01/programming-atmega328p-registers-from.html

analog to digital converter ADC
The microcontroller has an 10 bits ADC peripheral (analog to digital converter) and 6 analog pins (A0 to A5) to read analog values. The analog value is between 0 and 1023 (10 bits number). We provides the ADC with a reference value. If the reference value is 5V (default for IDE) then 1023 is 5V and 512 is 2.5V for example. We can pick a different reference value (see appendix A). The IDE function analogRead() take care of tuning the registers for us. See appendix A for more details on how to read analog values with the

IDE. It takes about 100 microseconds for a value to be converted in a digital value by default. You can change the frequency but it will affect the accuracy. The frequency for the ADC is supposed to be between 50KHz and 250KHz (datasheet). The clock is 16MHz and the prescaler is 128 so the clock is reduced to 125KHz. But it takes 13 cycles of ADC clock to take a sample or 9600Hz. Which corresponds to 100us/sample for a conversion. So a coversion is done every 100us. Two sample per cycle of an analog signal is required (Nyquist theorem). So the cycle has to be at least 200us for the signal to be sampled. Or the frequency of the signal to be digitalized (sampled) needs to be less than 4.8KHx (1//200us). So the ADC of Arduino can not digitlized signal with a frequency less than 4.8KHz. We can change the prescaler in one of the hardware register (ADCRA) but the accuracy gets worse. Prescalers can be 2, 4, 16, 32, 64, 128. You use the bits ADPS0, ADPS1, ADPS2 to set the prescalers in the register ADCRA (see http://maxembedded.com/2011/06/the-adc-of-the-avr/). You can select another prescaler but if the sampling frequency increases (larger than 125KHz) the accuracy decreased (the binary number has to be between 0 and 1023 with a step of 1 no matter the sampling rate).

The ADC of the uC is complex. First the analog value is stored in a capacitor. For that the capacitor is charged until it reaches the input value to be converted (a comparator is used). This is done to keep a snapshot of the value before the next sampling. This is called a “sample and hold” circuit. Then the ADC uses a technique called successive approximations. In Math it is the equivalent of the bisection method. A digital value (512 corresponding to 2.5V) half way between 0 and 1023 (10 bits) is first selected. That digital value is converted to an analog value (using a DAC) and compared to the input value using a comparator (op-amp). If the input is greater than 2.5V so 512 is stored (10000...). Then the input is compared to half of the half (512+256 or 3.75V). if it is greater we have 768 otherwise we keep 512 and try 640 and so forth. The result is a 10 bits number between 0 and 1023. It is actually a number spread over 2 registers of 8 bits. But we can just access the 10 bits number without worrying of the 2 registers used. The uC uses 2 registers. ADCL with the 8 least significant bits and ADCH and the 2 most significant bits.

If several analog inputs are to be sampled a multiway switch is used. This is called a multiplexer. If you don’t use the IDE then hardware registers have to be tuned to control the ADC. If you use registers then you have access to other modes. Like “ free-running mode” versus the “ single conversion mode” described above. In that case, the ADC is constantly reading the analog pins and output an analog number without waiting for instructions after a conversion. This is convenient if you are passing the numbers through the serial bus and plot them using python for example.

The AREF pin (analog reference pin) is by default 5V (that means 1023 is 5V). You can measure the voltage between this pin and the Gnd and you get 5V. You can connect this pin to a different reference voltage like 3V (1023=3V). This increase the resolution for the analog voltage (3/1024) V for each step. You can also use a command to select the reference 1.1V. It is called the interal voltage reference. Doing either, connect a 0.1uC between the AREF pin and the Gnd for stability. There is a capacitor between AREF and the Gnd to stabilize the analog reference voltage.

Interrupt

It is a section in the hardware so the uC can stop what it is doing and run a short routine (called Interrupt Service Routine or ISR for short). AVR uCs have only one core so it can not do 2 things at the same time. Thanks to the interrupt mode the uC will save what is is doing (execution backed up in the stack pointer) and the program counter is loaded with the routine (Interrupt Service Routine or ISR) address which is stored in the interrupt vector space address. Luckily, the arduino IDE make interrupts user friendly (see appendix A). For example, the command attachInterrupt(0, routine, FALLING) means if the voltage falls at

pin PD2 then run the routine. This command is to use the external interrupts. External interrupts attached to pin PD2 and PD3 (digital 2 and digital 3) also called pins INT0 and INT1 (AVR terminology) and those pins can tell when a voltage (from a button for example) changes, rises or fails a routine is run (ISR). We can have an external interrupt at the other pins as well but it can only react to a change in voltage. It can not tell a rising voltage from a falling voltage. If you use the other pins (not INT0/INT1) then you can not tell which pin " saw" a voltage change. Three banks of 8 pins can be configured as pin-change interrupts. But each bak shared the same vector so the same routine will be called if there is a voltage change at either one.

You need to keep the ISR as short as possible. While it runs nothing in the loop will be run and no other interrupt routine can be run. So the command delay() and millis() will stop counting because they use timer interrupts (they counts time since reset). However, you can use delayMicroseconds. You can not use Serial.print() or try to read from Serial. Serial communication also use interrupts.

Also you can not pass data through the ISR. To pass data, you need to use global variables of type volatile.
Volatile boolean flag=True; The ISR will recognize that variable.

You can explicitly turn interrupts on or off.

Special registers have to be tuned in order to use the other pins as interrupt pins.
If you are not using the IDE you have more flexibility but hardware registers have to be set and it get more complicated. First we enable one of the pin as an interrupt pin in the register EIMSK (bit INT0 means PD2). The uC will check its flag the flag for INT0 constantly. If the flag is on (change in voltage) then the ISR routine is run. This routine does not take any variable and does not return any variable but takes a "vector or pointer" (like INT0_vect) which points to the memory address where the ISR is found. The vector contains a command that makes the uC " jumps" to the location of the ISR. The hardware keeps a " vector space" or map with all the ' vectors' that points to an interrupt handler. This map is found in the datasheet. In the sketch you need to include the interrupt routine: ISR(INIT0_vect) { what to do if interrupt occurs }. When the ISR is run a global flag is turned off so no other interrupt can occur. If you don't use the arduino IDE, you need to set that flag on before, in the setting of the registers. When the ISR is done running, the global flag is on again and the INIT0 flag (for example) is cleared. You also have to switch a bit in another register to tell the uC to react to a voltage change, voltage rise or fail (only for PD2/INIT0 and PD3/INIT1).

You also have internally triggered interrupts that respond to the internal AVR hardware peripherals. For example, the interrupts that can run the ISR code when the ADC / serial communication peripherals have got new data They can also be triggered by timers. The datasheet gives a map with the priority of the interrupts. The uC constantly checks the flags of the interrupts. Is it set? If yes it runs the corresponding ISR. Then the uC clears the interrupt flag again. You can clear the flag manually. The uC runs the interrupt with the highest priority according to the map. INIT0 has the highest priority.

Three timers (counters)
Timers run parallel to the CPU and are part of independent modules. Timers are also peripherals and arduino has 3 timers. Each timer has 2 pins attached to it so they can be used to produce waveform with a given frequency (the timer toggles the pin after a given cycle). Timers have three modes. They can be used as counters and the value of the counter is found in a register (TCNT0 for timer 0). This way you can time events. The CTC mode is clear timer on match. We can set a top value and when the counter reaches the value, it overflows and starts counting again. A pin can be toggled each time the timer overflows. A radio

wave can produced this way and square wave. Timers are used to produce modulated waveform (PWM) with the PWM mode. This mode is used to simulate analog outputs.

The highest frequency between each tick of the timer is 16MHz (with an external quartz crystal). But the frequency can be decreased (clock can be slowed down) by prescalers or divisors.

You can attach a timer to an interrupt routine so when the counter reaches a value then something is done (routine ISR is run).

Timers are used for the Arduino functions tone(), delay(), millis(), micros(), servo().

timer0. This is 8-bits counter. So it counts one bit at a time from 0 to 255. The divisor set by default by Arduino is 64 which means the frequency of the counter is 64/16,000,000 = 250,000 Hz. So the time between each click is 1/250,000= 4 microseconds. If we let the counter counts from 0 to 255 then the period of 1 cycle is 256 x 4 us = 1ms about. Or about 980Hz. This is the frequency used for the PWM (see below) pins 5 and 6 (also called PD5, PD6 or OC0A, OC0B). This is because those pins are " attached" to the timer0. We can force the timer to click slower or faster by changing the prescaler (or divisor). The frequency for the PWM pins can vary from 62,500Hz (divisor is 1) to 61Hz (divisor is 1024). The timer0 controls the functions millis(), delay(). If you mess with the timer0 then it will mess those functions. Micros() has a resolution of 4us because of the above. Special registers control the timers. For example TCNT0 stores the timer value. TCCR0B is the control timer to configure the prescaler. The register TCCR0A is also used for configurations. Other registers are also involved. Codes to change the divisors are found on line. see http://playground.arduino.cc/Main/TimerPWMCheatsheet. Usually we don't touch the timer 0 but we touch timer1 when using the arduino IDE. Delay(), micros(), millis() are attached to timer0. Arduino has libraries to change the prescalers.

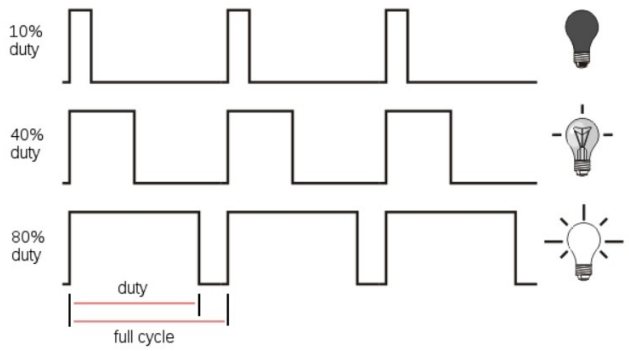
timer1. It is a 16-bits timer but it controls the PWM frequency for pins 9 and 10. In that case, only 8 bits is used and the mode is phase-correct. This is the setting by Arduino. It means the counter counts up and down the 8-bits (0 to 255 then down to 0), one bit at a time. So the total number of bits is 510 (255 x 2) .The divisor is again 64 by default (by Arduino) or a frequency of 250,000Hz and a resolution of 4 us. 4us x 510= 0.00204 of about 2ms for 1 cycle of the clock. The frequency of the clock is therefore 490Hz. This is the PWM frequency for pins 9 and 10. You can change the prescaler to get a frequency of 30Hz for example. (see previous website). You could turn timer1 into a 16 bits counter and use it to count up to 8s. The counter runs up and down $(2^{16} - 1)$ x 2 bits, one bit at a time. It can count from 8ms to 8s depending of the prescaler. With a prescaler of 256 we get a counter of 2s period. You can change the frequency when using motors for a robot for example. Timer1 controls the servo() library. See http://playground.arduino.cc/Code/Timer1 for prescalers (divisors) allowed.

There is a useful function provided by arduino to change the PWM frequencies. Example: setPwmFrequency(3, 1024); The pin 10 has a frequency of (31372/1024 = 30)
see http://playground.arduino.cc/Code/PwmFrequency

timer2. It is a 8-bits counter and it controls the PWM frequencies of pins 11 and 3. It controls tone(). It offers more prescalers than timer1. See http://playground.arduino.cc/Main/TimerPWMCheatsheet. By default the prescaler is 64.

Reference: https://arduino-info.wikispaces.com/Timers-Arduino

PWM (pulse width modulation) instead of DAC controlled by timers.
There is no DAC (digital to analog converter) for say but the uC uses PWM (pulse width modulation) method to simulate digital to analog conversion. PWM describes a square wave where you modulate the amount of time it stays on in ratio of the total period. So if you want to generate 1V the square wave is on for 20% of the period and off for 80% of the period. It is a digital signal (see below the diagrams).

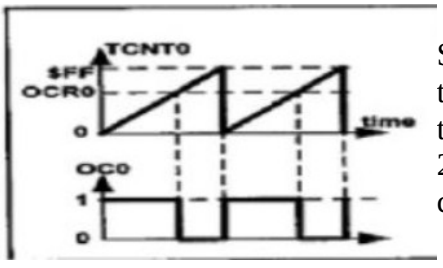


If the wave is 50% on and 50% off the analog output is 2.5V. If it is 10% on and 90% off the analog output is 0.5V etc..
A 10% duty cycle would read 0.5V with a voltmeter and would give out a very dim light. With an Arduino command you would write analogWrite(pin,26). 26 is about 10% of 255. A 40% duty cycle would give 2V and 80% 4V. A voltmeter can not pick up the change in voltage and get the average. It will work fine for a light bulb or a motor. But it would not work with an oscilloscope that " would see" the waveform and not the average. It would not work with sensors with a quick response to voltage. In which cases, you can add a DAC to Arduino or attach a R-2R network (simple R-2R DAC) if you need to.

See: https://learn.sparkfun.com/tutorials/pulse-width-modulation/all for more.

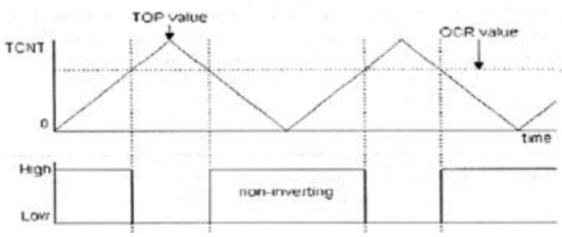
PWM signals are generated using timers. You get two PWM pins per timer so 6 for the Atmega328. PWM can be generated at pins 3,5,6,9,10 and 11. By default the frequencies of the signal are 490Hz or 980Hz (see table). This can be changed using prescalers. Arduino has libraries that makes it easy to change the frequencies. Otherwise, you need to tune registers.

pins	base frequency	default frequency	divisors	timers	functions attached
3	31372	490	1,8,32,64,128,256,1024	timer2	tone()
5	62500	980	1,8,32,**64**,128,256,1024	timer0	delay()..
6	62500	980	1,8,32,**64**,128,256,1024	timer0	delay()…
9	31372	490	1,8,**64**,256,1024	timer1	servo
10	31372	490	1,8,**64**,256,1024	timer1	servo
11	31372	490	1,8,32,**64**,128,256,1024	timer2	tone()
		divisor is 64			



So timers are used to generate PWM waveform. For example, if you use timer0 in fast PWM mode (tuned by registers and set by default by Arduino) the counter will count from 0 to 255, overflows and starts counting from 0 to 255 again (sawtooth signal here). The register TCNT0 has the value of the counter.

If you need a 40% duty free signal then write 102 in the a register (OCRO) and when the counter (TCNT0) reaches this value the signal is off until the counter is back to 0. This is a non-interting mode because the signal is high for 40% ot the time and off for 50%.



Another option is to count up and down this is called Phase direct. In this case, the frequency is twice are much. With Arduino, the timer 1 counts from 0 to 255 then backward to 0. See http://maxembedded.com/2011/08/avr-timers-pwm-mode-part-i/

Synchronous data buses – they require a clock line – faster than the asynchronous bus line (UART)
I2C and SPI protocols

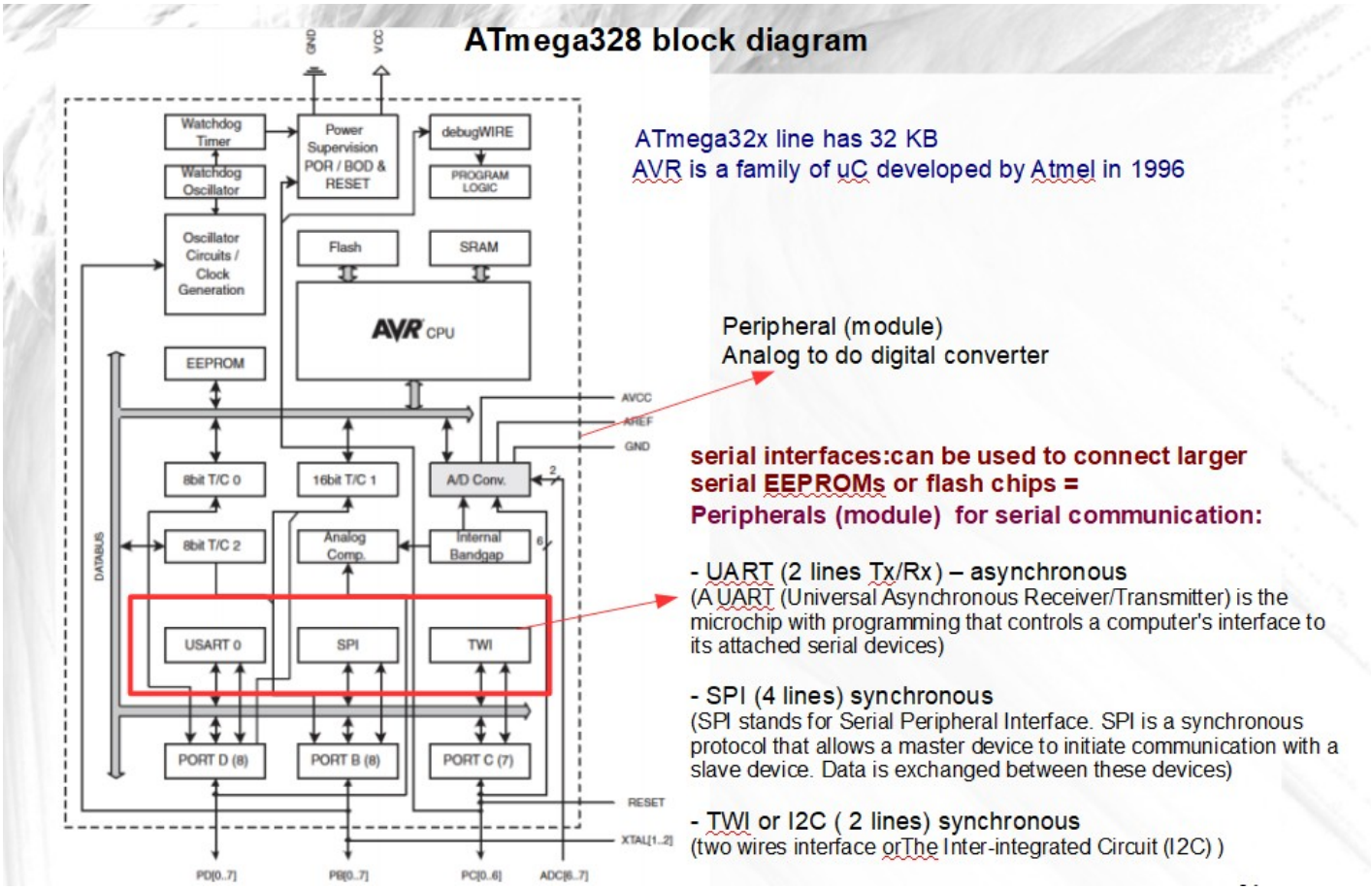
Arduino can communicate with other ICs with the protocols I^2C and SPI. The communication is via a data bus, a system of connections that allow two or more devices to exchange data in an orderly manner. The I^2C bus also known as the two wires interface (TWI). Arduino is the master and the IC the slave. Data sent and received are using the same wire. So you can not send / receive at the same time. You need the wire.h library and you need to use the functions that go with it. Each IC has a given address so arduino knows where to send the data. https://www.arduino.cc/en/Reference/Wire and https://learn.sparkfun.com/tutorials/i2c

The SPI bus can send and receive data at the same time and at different speed. Arduino is the master and the IC is the slave. Arduino communicate with one slave at a time. The library to use is SPI.h. https://www.arduino.cc/en/Reference/SPI. https://learn.sparkfun.com/tutorials/serial-peripheral-interface-spi

The quick reference below is from: http://playground.arduino.cc/Learning/Pins

		I/O	PWM duty	10bit ADC	TWI / I^2C	High Speed	triggers	USB / FTDI	USB / FTDI	port (3)	PCINT (4)	PCMSK (5)
-Pin-	Alias	pinMode() digitalWrite() digitalRead()	analog Write()	analog Read()	Wire .send()	Spi .transfer()	attach Interrupt()	Serial .print()	Serial .read()			
0	RX	YES	-	-	-	-	-	-	YES	D	16	PCMSK2:0
1	TX	YES	-	-	-	-	-	YES	-	D	17	PCMSK2:1
2	IRQ0	YES	-	-	-	-	YES *	-	-	D	18	PCMSK2:2
3	IRQ1	YES	YES (2)	-	-	-	YES *	-	-	D	19	PCMSK2:3
4	-	YES	-	-	-	-	-	-	-	D	20	PCMSK2:4
5	-	YES	YES (0)	-	-	-	-	-	-	D	21	PCMSK2:5
6	-	YES	YES (0)	-	-	-	-	-	-	D	22	PCMSK2:6
7	-	YES	-	-	-	-	-	-	-	D	23	PCMSK2:7
8	-	YES	-	-	-	-	-	-	-	B	0	PCMSK0:0
9	-	YES	YES (1)	-	-		-	-	-	B	1	PCMSK0:1
10	SS	YES	YES (1)	-	-	YES **	-	-	-	B	2	PCMSK0:2
11	MOSI	YES	YES (2)	-	-	YES **	-	-	-	B	3	PCMSK0:3
12	MISO	YES	-	-	-	YES **	-	-	-	B	4	PCMSK0:4
13	SCK	YES	-	-	-	YES **	-	-	-	B	5	PCMSK0:5
14	A0	YES	-	YES *	-	-	-	-	-	C	8	PCMSK1:0
15	A1	YES	-	YES *	-	-	-	-	-	C	9	PCMSK1:1
16	A2	YES	-	YES *	-	-	-	-	-	C	10	PCMSK1:2
17	A3	YES	-	YES *	-	-	-	-	-	C	11	PCMSK1:3
18	A4	YES	-	YES *	YES **	-	-	-	-	C	12	PCMSK1:4
19	A5	YES	-	YES *	YES **	-	-	-	-	C	13	PCMSK1:5

APPENDIX C: Architecture of a 8-bits microcontroller (ATmega32 in Arduino)



source: http://ww1.microchip.com/downloads/en/DeviceDoc/Atmel-8271-8-bit-AVR-Microcontroller-ATmega48A-48PA-88A-88PA-168A-168PA-328-328P_datasheet_Complete.pdf p.6

see also:
https://www.allaboutcircuits.com/technical-articles/understanding-arduino-uno-hardware-design/