



# Measurements, Sensors and Data Logging Course

Week 4

# Simple Datalogger

## Lesson 8

# Arduino Shields

## Lesson 8: Simple Datalogger

- What is an Arduino Shield?
  - An Arduino shield is a circuit board that follows the basic shape and pinout of the Arduino board. It plugs into the top of the Arduino Board to extend the capabilities of the Arduino with the circuit on the shield.
- How do we use an Arduino Shield?
  - Arduino shields are plugged into the top of the Arduino board and can be stacked together.
  - Libraries and code can then be used to access the additional capabilities of the shield.
- More Information:
  - <https://learn.sparkfun.com/tutorials/arduino-shields>
  - <https://www.arduino.cc/en/Main/arduinoShields>

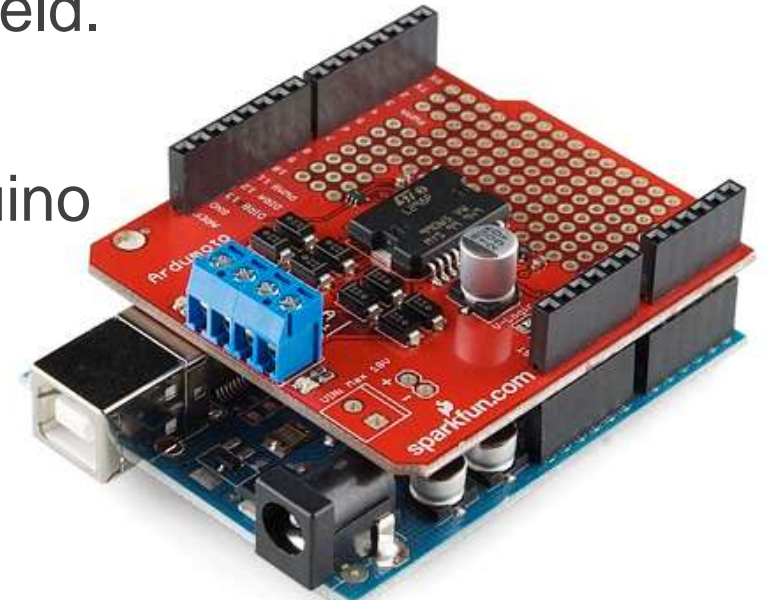
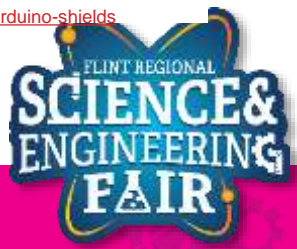


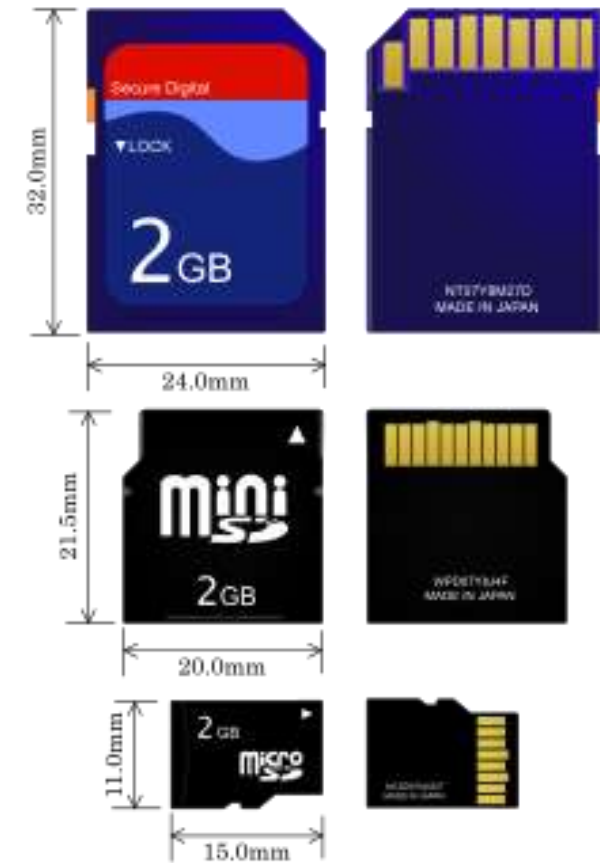
Image copied from <https://learn.sparkfun.com/tutorials/arduino-shields>



# SD Card

## Lesson 8: Simple Datalogger

- What is an SD Card?
    - Memory storage device
    - Stands for Secure Digital
    - Multiple Sizes: physical form factors and memory capacity
  - Where is Flash data storage used?
  - How do we use an SD card with an Arduino?
    - SD card shield!!
    - SD card library
    - SPI communication (digital communication)
- <https://www.arduino.cc/en/reference/SD>



# Digital Communications

## Lesson 8: Simple Datalogger

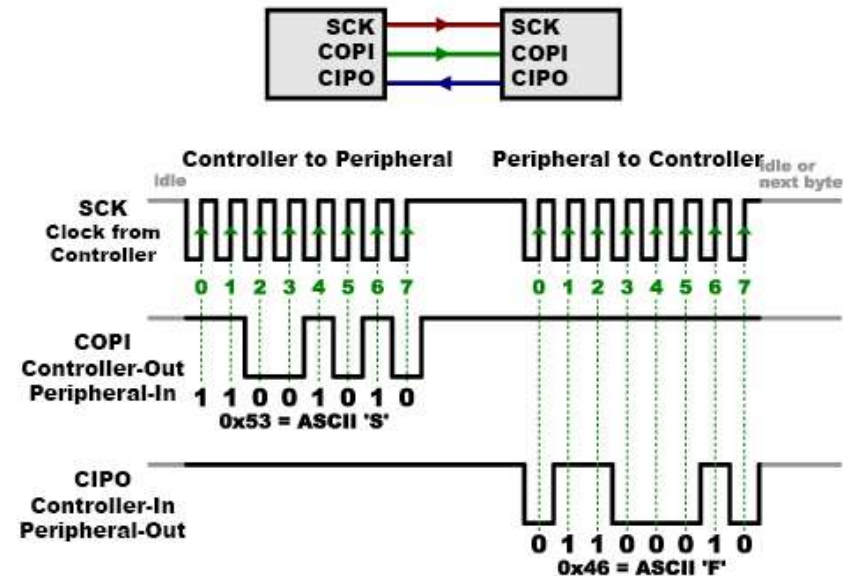
- Digital Communications
  - Digital signals used to communicate between devices
  - Ability to combine multiple signals together
  - Many different protocols
- Example protocols
  - CAN (2 wires)
    - Ability to have multiple devices communicating to each other (3+)
  - RS232 (3 wires)
    - Only 2 devices can communicate together



# SPI

## Lesson 8: Simple Datalogger

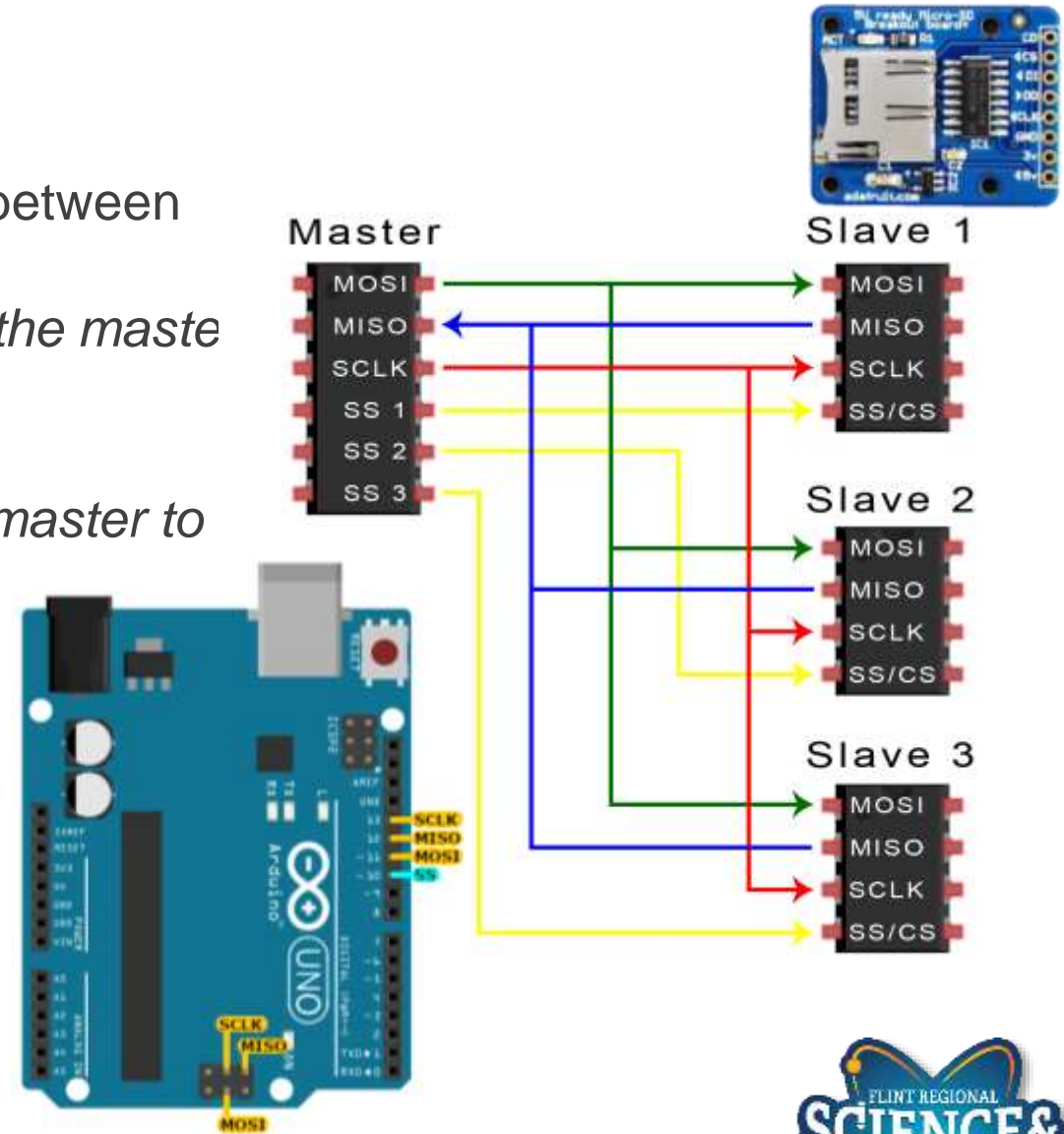
- SPI (Serial Peripheral Interface), multiple devices, short distances
  - Synchronous Communication: clock signal shared between master and slave
  - COPI: Controller Out / Peripheral In, *sends data to the master from the slave device*
    - MISO: Master In Slave Out
  - CIPO: Controller In / Peripheral Out, *data from the master to the slave device*
    - MOSI: Master Out Slave In
  - SCK: Serial Clock
  - CS: Slave Select (individual for each device)
  - <https://www.arduino.cc/en/reference/SPI>
  - <https://www.corelis.com/education/tutorials/spi-tutorial/>



# SPI

## Lesson 8: Simple Datalogger

- Synchronous Communication: clock signal shared between master and slave
- COPI: Controller Out / Peripheral In, *sends data to the master from the slave device*
  - MISO: Master In Slave Out
- CIPO: Controller In / Peripheral Out, *data from the master to the slave device*
  - MOSI: Master Out Slave In
- SCK / SCLK: Serial Clock
- CS / SS: Slave Select (individual for each device)



# Lesson 8 Hardware

## Lesson 8: Simple Datalogger

- What hardware will we need for this Lesson?
  - Potentiometer on pin A0
  - Grove Light Sensor on pin A6
  - Grove Sound Sensor A2
  - Seeeduino Lotus (Arduino Uno compatible board)
  - SD Card Shield
    - Multiple options are available
  - SD Card
    - Formatted as FAT16

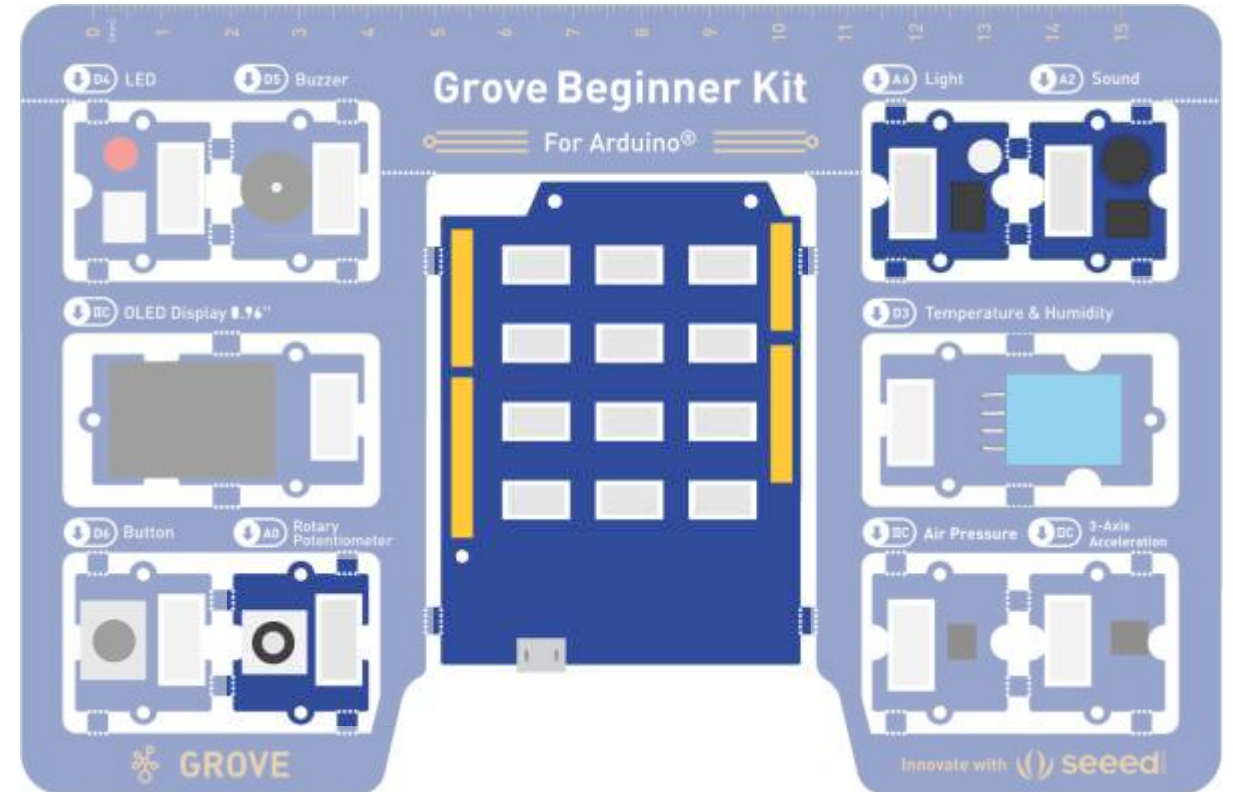


Image modified from <https://files.seeedstudio.com/wiki/Grove-Beginner-Kit-For-Arduino/res/Grove-Beginner-Kit-For-ArduinoPDF.pdf>



Image copied from <https://www.amazon.com/HiLetgo-Logging-Recorder-Logger-Arduino/dp/B00PI6TQWO/>



Image copied from <https://www.microcenter.com/product/485234/micro-center-64gb-microsdxc-class-10-uhs-1-flash-memory-card>





# Assemble the Datalogging Hardware

## Lesson 8: Simple Datalogger

1. Carefully align the pins of the shield with the sockets in the Seeeduino Lotus and press down until seated
2. Place micro-SD card into SD card adaptor and insert into SD Card Shield
3. Plug in USB cable between Seeeduino Lotus and Computer

# Open and Upload Sketch

## Lesson 8: Simple Datalogger

4. Open Simple Datalogger Sketch
  - **File → Sketchbook → FRSEF\_Crash\_Course → Week\_4 → L8\_Simple\_Datalogger.ino**
5. Upload the sketch to your Arduino by clicking the Upload Button.
  - The sketch should compile, and then upload to your Arduino.
6. Open the serial monitor.
  - **Tools → Serial Monitor (Ctrl+Shift+M)**
7. Observe the output in the Serial Monitor for a few seconds
  - Cover the light sensor, rotate the potentiometer, and make some noise to change the value of the sensor readings
8. Unplug the USB connector



# SD Library

## Lesson 8: Simple Datalogger

```
#include <SD.h>
```

- The SD Library allows us to communicate with SD cards formatted with FAT16 and FAT32 filesystems.
- It is built into the Arduino IDE, so there is no need to install it.
- It takes advantage of the SPI interface of the SD card, so the SPI library must also be included (`#include <SPI.h>`) before the SD library.
- It requires the chip select (sometimes called slave select) pin from the SD card to be connected to a digital output of the Arduino.
  - The SD Card Shield we chose uses pin D10 for the chip select
- More information:
  - <https://www.arduino.cc/en/Reference/SD>

# Open Logged Data on Computer

## Lesson 8: Simple Datalogger

9. Remove SD card from Shield
10. Insert into computer
11. Open “datalog.csv” in excel or other spreadsheet program
12. Bonus Activity: Graph the data!





# CSV File

## Lesson 8: Simple Datalogger

- What is a CSV file?
  - Comma Separated Value
  - Typically used as a generic spreadsheet, can be opened in many programs.
    - Excel, Google Sheets
  - Also used for exporting and importing data with specialized programs.
- How do we use a CSV file?
  - Can be opened in Excel and used similar to a regular excel file
    - Excel features (math, etc) cannot be saved
    - File can be saved as a regular .xls / .xlsx
  - Can also be opened in google sheets, notepad++, etc

# SD Library – Appending data to a file

## Lesson 8: Simple Datalogger

- We can append data to a file by:
  - Opening the file for writing
  - Checking that the file was opened
  - Writing a String to the file (we cover creating this String on the next slides)
  - Closing the file

```
File dataFile = SD.open("datalog.csv", FILE_WRITE);  
if (dataFile)  
{  
    dataFile.println(dataString);  
    dataFile.close();  
}
```

# Strings

## Lesson 8: Simple Datalogger

```
String example = "This is an example of a string.";
```

- Create a String class instance named `example` containing the text "This is an example of a string."
- Strings are representations of **text** instead of numbers, and as such they do not behave in the same way as our previous datatypes.
- We have used string constants before in our sketches. Look for text encased in quotes, usually inside our `print()` and `println()` statements.
- More information:
  - <https://www.arduino.cc/reference/en/language/variables/data-types/stringobject/>
  - <https://www.arduino.cc/en/Tutorial/BuiltInExamples#strings>



# Converting other datatypes to Strings

## Lesson 8: Simple Datalogger

- Syntax:

```
String(val)
```

```
String(val, base)
```

```
String(val, decimalPlaces)
```

- `val` – variable to format as a string
- `base` - (optional) if `val` is an integer, what base should be used for the string representation. Options – DEC (default), BIN, or HEX
- `decimalPlaces` – (optional) if `val` is a float, how many decimal places should be used. 2 is default

- For example:

- |                                   |             |   |
|-----------------------------------|-------------|---|
| – <code>String(42)</code>         | → “42”      | // convert integer value to DEC           |
| – <code>String(42, BIN)</code>    | → “101010”  | // convert integer value to BIN           |
| – <code>String(42, HEX)</code>    | → “2A”      | // convert integer value to HEX           |
| – <code>String('x')</code>        | → “x”       | // convert character value to String      |
| – <code>String(0.12345)</code>    | → “0.12”    | // convert float value to DEC to 2 places |
| – <code>String(0.12345, 5)</code> | → “0.12345” | // convert float value to DEC to 5 places |

- This operation is called type casting





# String Concatenation

## Lesson 8: Simple Datalogger

- Concatenation – combining 2 strings into one larger string
- There are several way to concatenate strings in Arduino
- Syntax:

```
string1 += string2;
```

– Appends `string2` to the end of `string1`

- Example:

```
String string1 = "1";
```

```
String string2 = "2";
```

```
string1 += string2; // string 1 now equals "12"
```

```
Serial.print(string1); // prints 12
```

- More information and more ways to concatenate strings:


- <https://www.arduino.cc/reference/en/language/variables/data-types/string/functions/concat/>
- <https://www.arduino.cc/reference/en/language/variables/data-types/string/operators/concatenation/>
- <https://www.arduino.cc/reference/en/language/variables/data-types/string/operators/append/>

# Creating a CSV Row with a String

## Lesson 8: Simple Datalogger

```
String dataString = "";  
dataString += String(val1);  
dataString += ",";  
dataString += String(val2);  
dataString += ",";  
dataString += String(val3);
```

val1, val2, val3



	A	B	C
1	val1	val2	val3

# Activities

## Lesson 8: Simple Datalogger

- Read the temperature and humidity and light sensors overnight
  - Read the data and open it in a program (excel) to analyze and plot

# Real Time Clock + Time Library

Lesson 10



# Hz + Logging Rates

## Real Time Clock + Time Library

- What is Hz?
  - A unit of frequency. aka: how often something is happening each second
- How do we calculate Hz?
  - # of occurrences / second or 1 / time event takes (sec)
    - $\frac{\text{\# of occurrences}}{1 \text{ (sec)}}$  ex. 5 occurrences / 1 sec = 5 Hz
    - $\frac{1}{\text{time event takes (sec)}}$  ex. 5 seconds / event  $\Rightarrow$  1 occurrence / 5 sec = 0.2 Hz
- What rate should we log our data at?
  - Depends!
  - Ideal: 10 times faster than the signal changes
  - Minimum: 2 times faster than the signal changes

# Real Time Clock

## Real Time Clock + Time Library

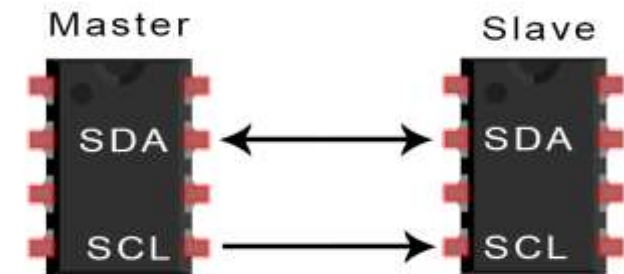
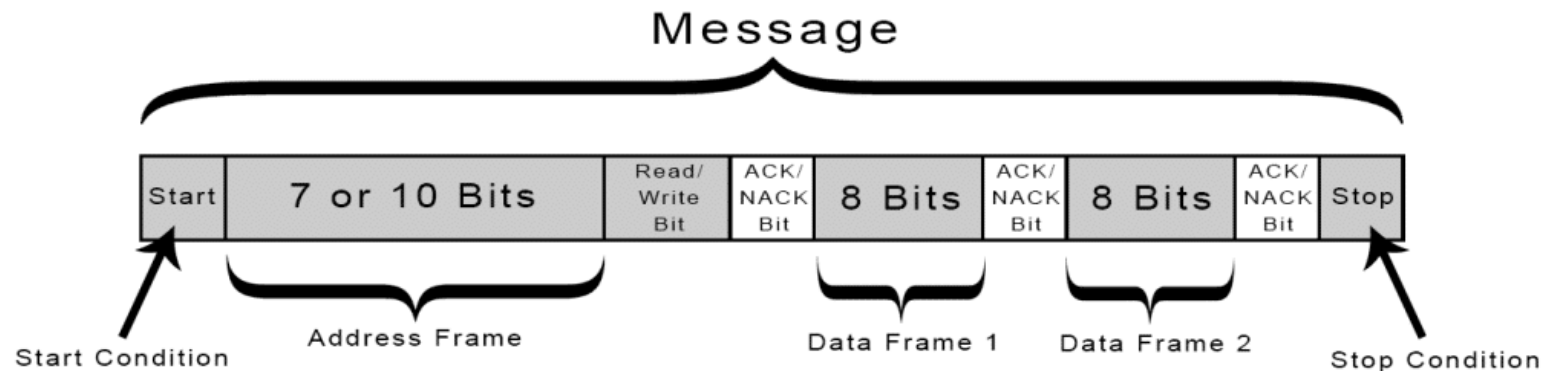
- What is a Real Time Clock (RTC)?
  - A clock that keeps track of the current time, independent from the Arduino.
    - On-board crystal oscillator.
    - Battery keeps the RTC powered even when the Arduino is powered down.
  - Arduino can be used to read this time and program.
- How do we use the RTC?
  - Time and PCF8523 Libraries
  - I2C communication
  - Need to set the time on the RTC the first time it is powered (already set on your hardware)
- More Information:
  - [https://www.pjrc.com/teensy/td\\_libs\\_Time.html](https://www.pjrc.com/teensy/td_libs_Time.html)
  - [https://www.pjrc.com/teensy/td\\_libs\\_DS1307RTC.html](https://www.pjrc.com/teensy/td_libs_DS1307RTC.html)



# I<sup>2</sup>C (Inter-Integrated Circuit)

## Real Time Clock + Time Library

- I<sup>2</sup>C, sometimes I2C or IIC
  - Two wires and robust, but slower transfer rate than SPI
  - Library: Wire.h
  - SDA (serial data): The line for the master and slave to send and receive data.
  - SCL (serial clock): The line that carries the clock signal.

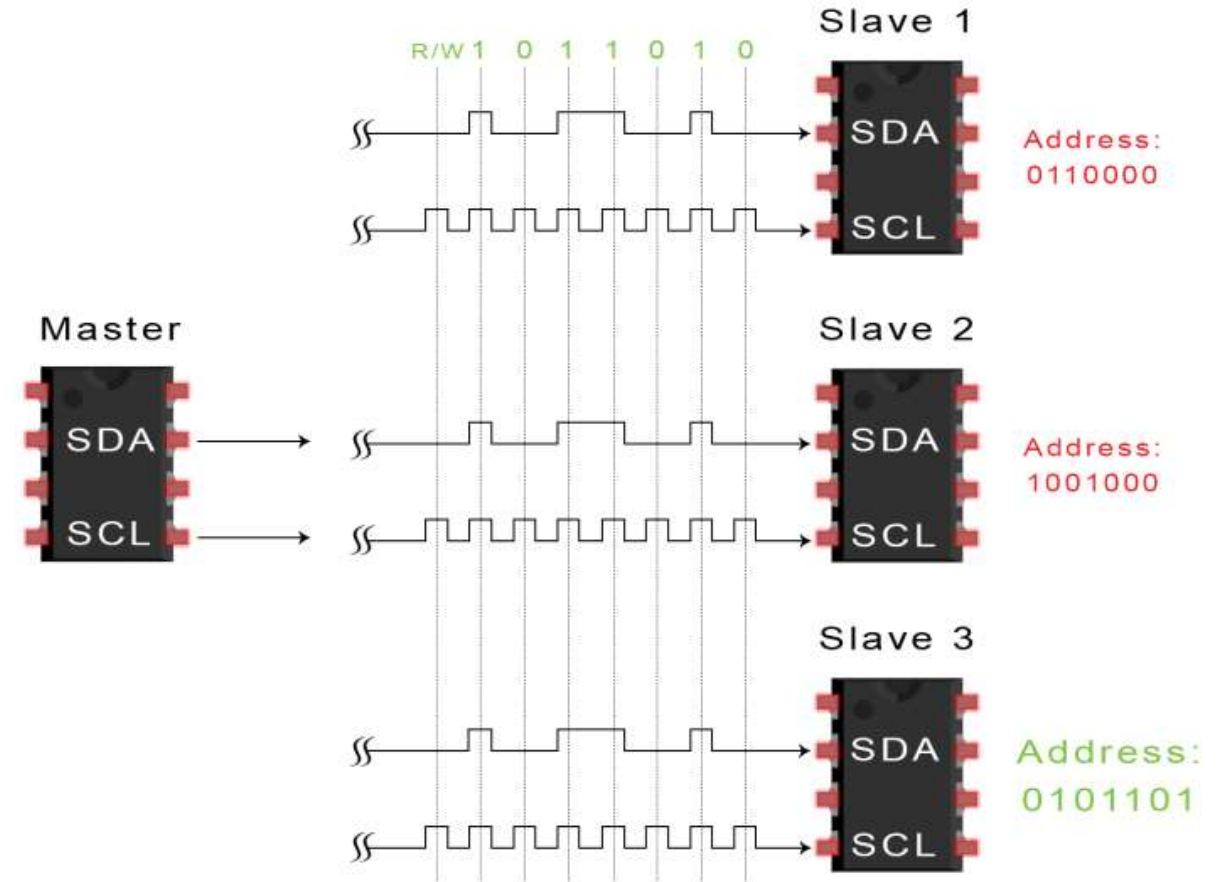
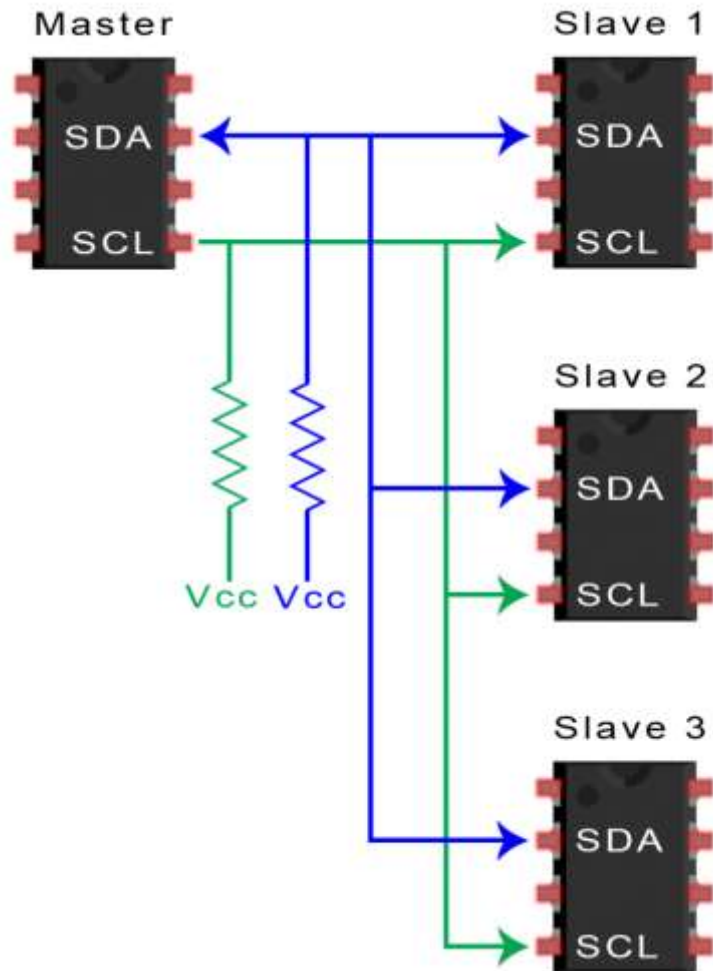


- <https://www.circuitbasics.com/basics-of-the-i2c-communication-protocol/#:~:text=I2C%20is%20a%20serial%20communication,always%20controlled%20by%20the%20master>



# I<sup>2</sup>C (Inter-Integrated Circuit)

Real Time Clock + Time Library





# Lesson 9 Hardware

## Real Time Clock + Time Library

- What hardware will we need for this Lesson?
  - Potentiometer on pin A0
  - Grove Light Sensor on pin A6
  - Grove Sound Sensor A2
  - Seeeduino Lotus (Arduino Uno compatible board)
  - SD Card Shield (HiLetGo) w/ battery
    - Multiple options are available
  - SD Card
    - Formatted as FAT16

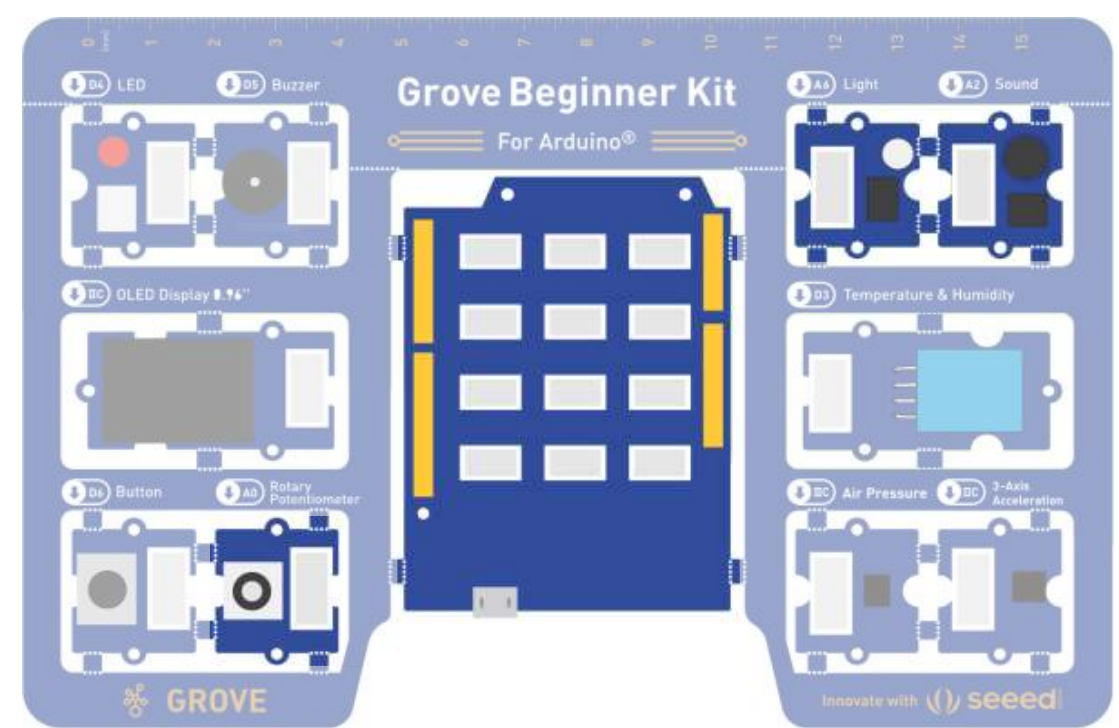


Image modified from <https://files.seeedstudio.com/wiki/Grove-Beginner-Kit-For-Arduino/res/Grove-Beginner-Kit-For-ArduinoPDF.pdf>



Image copied from <https://www.amazon.com/HiLetgo-Logging-Recorder-Logger-Arduino/dp/B00PI6TQWO/>



Image copied from <https://www.microcenter.com/product/485234/micro-center-64gb-microsdxc-class-10-uhs-1-flash-memory-card>



# Install the Time Libraries

## Real Time Clock + Time Library

1. Install the Time library from the Library Manager
  - a. Sketch → Include Library → Manage Libraries...
  - b. Search for “**RTCLib**”
  - c. Install **RTCLib** by Adafruit
  - d. Search for “**TimeAlarms**”
  - e. Install **TimeAlarms** by Michael Margolis
  - f. Install **Time** by Michael Margolis



# Open and Upload Sketch

## Real Time Clock + Time Library

2. Open Simple Datalogger Sketch
  - **File → Sketchbook → CrashCourse\_Jan → L9\_Timed\_Datalogger.ino**
3. Upload the sketch to your Arduino by clicking the Upload Button.
  - The sketch should compile, and then upload to your Arduino.
4. Open the serial monitor.
  - **Tools → Serial Monitor (Ctrl+Shift+M)**
5. Observe the output in the Serial Monitor for a few seconds
  - Cover the light sensor, rotate the potentiometer, and make some noise to change the value of the sensor readings
6. Unplug the USB connector



# Open Logged Data on Computer

## Lesson 8: Simple Datalogger

7. Remove SD card from Shield
8. Insert into computer
9. Open “datetime.csv” in excel or other spreadsheet program. Note the date and time recognized by the spreadsheet
10. Bonus Activity: Graph the data!

# Time Library

## Real Time Clock + Time Library

- What is the Time library?
  - An Arduino library that helps in keeping the time and provides functions to deal with seconds, minutes, hours, days, months and years.
  - The Time library can be synced with a RTC or other time and date services (GPS, NTP, Serial or other service that provides a standard Unix `time_t` time)

```
#include <TimeLib.h>
```

- Syntax:

`year()` – return the current year  
`year(t)` – return the year of `t`  
`month()` – return the current month (1-12)  
`month(t)` – return the month of `t` (1-12)  
`day()` – return the current day of the month (1-31)  
`day(t)` – return the day of `t` (1-31)

`hour()` – return the current hour (0-23)  
`hour(t)` – return the hour of `t` (0-23)  
`minute()` – return the current minute (0-59)  
`minute(t)` – return the minute of `t` (0-59)  
`second()` – return the current second (0-59)  
`second(t)` – return the second of `t` (0-59)  
`now()` – return the current time as a `time_t` number

`t` is a `time_t` number

`setSyncProvider(getTimeFunction)` – configure Time library to periodically call a user specified function to sync the clock.  
`getTimeFunction` is the name of the function that gets called.

- More Information

- [https://www.pjrc.com/teensy/td\\_libs\\_Time.html](https://www.pjrc.com/teensy/td_libs_Time.html)
- <https://github.com/PaulStoffregen/Time>

# RTCLib Library

## Real Time Clock + Time Library

```
setSyncProvider (RTC.get) ;
```

- What is the RTCLib library?
  - An Arduino library to interface with the DS1307 RTC over I<sup>2</sup>C (I2C or IIC).
  - It provides lower level access to the get (read) and set (write) the time to and from the RTC chip.

```
#include <RTCLib.h>
```

- Syntax:

`rtc.now()` – reads the current date and time from the RTC and returns it as a `DateTime` number. (need to convert from `DateTime` to `time_t` to use inside the `setSyncProvider()` function from the time library.)

- More Information

- <https://github.com/adafruit/RTCLib>





# RTCLib Library Setting the Time

## Real Time Clock + Time Library

- How to set the time of the RTC (we have already done this for you)
  1. Open the SetTime example:
    - a. File → Examples → RTCLib → pcf8523
  2. Upload to the Arduino
- This example utilizes the compile time to determine the current time and set it into the RTC.
- When would you have to do this:
  - After turning on the RTC after any power loss (ex battery died, first power on)
  - If the time is significantly off (remove and replace the battery in the shield while Arduino is unpowered to reset the clock)
  - Adjusting to a different time zone

# TimeAlarms Library

## Real Time Clock + Time Library

- What is the TimeAlarms library?
  - An Arduino library designed to work with the Time library to run functions at specific times.
  - Can work similar to an alarm clock or a timer and can trigger these alarms once or repeatedly.

```
#include <TimeAlarms.h>
```

- Syntax:

`Alarm.delay`(milliseconds) – check if alarm or timer should run and then delay for specified milliseconds

`Alarm.alarmRepeat`(dayofweek, hours, minutes, seconds, function) – create repeating alarm that calls function at a particular time. Optional dayofweek can be used to only repeat on a certain day.

`Alarm.alarmOnce`(dayofweek, hours, minutes, seconds, function) – create one off alarm to call function at a particular time. Optional dayofweek can be used to only trigger on a certain day.

`Alarm.timerRepeat`(seconds, function) – create a timer that calls function at seconds interval.

`Alarm.timerOnce`(seconds, function) – create one off timer to call function at a particular time once.

- More Information

- [https://www.pjrc.com/teensy/td\\_libs\\_TimeAlarms.html](https://www.pjrc.com/teensy/td_libs_TimeAlarms.html)
- <https://github.com/PaulStoffregen/TimeAlarms>



# Activities

## Real Time Clock + Time Library

- Read the temperature and humidity and light sensors at 1 minute intervals overnight
  - Read the data and open it in a program (excel) to analyze and plot
- Turn on the LED for 2 minutes at 8:00 AM and 8:00 PM each day



# Review

## Real Time Clock + Time Library

- What unit do we use to describe how fast we are logging a signal?
  - If we are logging a signal 20 times / sec, we are logging at \_\_\_\_\_
  - If something takes 10 seconds to occur, it is happening at \_\_\_\_\_
- What is a library?
  - What command do we use in our sketch to reference a library?
  - What do we use to install a library on our computer?

# Two Point Calibration

# Calibration

## Sensor Calibration

- What is calibration?
  - Calibration establishes a known relationship between a measurement (ex. voltage) and a standard (ex. temperature, position).
- What types of calibration are there?
  - Offset (add or subtract out error)
  - Slope [Sensitivity] (multiply by a correction factor)
  - Slope and Offset (combination of prior two, think  $y = mx + b$ )
  - Two Point (linear interpolation between two known points)
  - Best Fit Equation [Characteristic Equation] (algebraic equation of line of best fit)
  - Look Up Table [LUT or Characteristic Curve] (linear interpolation between point on the table)
- More information:
  - <https://learn.adafruit.com/calibrating-sensors?view=all>
  - <https://us.flukecal.com/literature/about-calibration>





# Open and Upload Sketch – At Home

## Sensor Calibration

1. Open Calibration Sketch
  - **File > Sketchbook > CrashCourse\_Jan > L14\_Calibration.ino**
2. Upload the sketch to your Arduino by clicking the Upload Button.
  - The sketch should compile, and then upload to your Arduino

# Two Point Calibration of the Potentiometer

## Sensor Calibration

- Activity:
  1. Adjust the potentiometer such that the slot in the knob is vertical. Record at the raw value.
  2. Rotate the knob by  $180^\circ$  (half turn) and record the raw value.
  3. Update `deg90Count` and `deg270Count` values and re-upload.
  4. Note how the Degrees output is now much closer to the actual rotational value.
- When setting up logging – record the new calibration value and the raw value off of the sensor.
- Calibration can also be done during data analysis.

# Code Analysis: map () function

## Sensor Calibration

```
map(potValueRaw, deg90Count, deg270Count, 90, 270);
```

- Linearly interpolate `potValueRaw` between points defined by `deg90Count`, `deg270Count`, 90, and 270.
- Re-maps a number from one range to another.
  - A value of `fromLow` would get mapped to `toLow`,
  - A value of `fromHigh` to `toHigh`,
  - Values in-between to values in-between, etc.
- Syntax:

```
map(value, fromLow, fromHigh, toLow, toHigh)
```

  - `value`: number to map or interpolate
  - `fromLow`: lower bound of input range (range of value)
  - `fromHigh`: upper bound of input range (range of value)
  - `toLow`: lower bound of the output range (range of calibrated value)
  - `toHigh`: upper bound of the output range (range of calibrated value)
- More information:
  - <https://www.arduino.cc/reference/en/language/functions/math/map/>



# Analysis Tools + Logging Trigger

# Options

## Analysis Tools + Logging Trigger

- Endless number of tools, some of the most commonly used:
  - Excel
  - Power BI
  - Matlab
  - MathCAD
  - Google Data Studio
  - Python Scripts (write your own)
- Random Other Tools
  - WinDarab (by Bosch Motorsport)

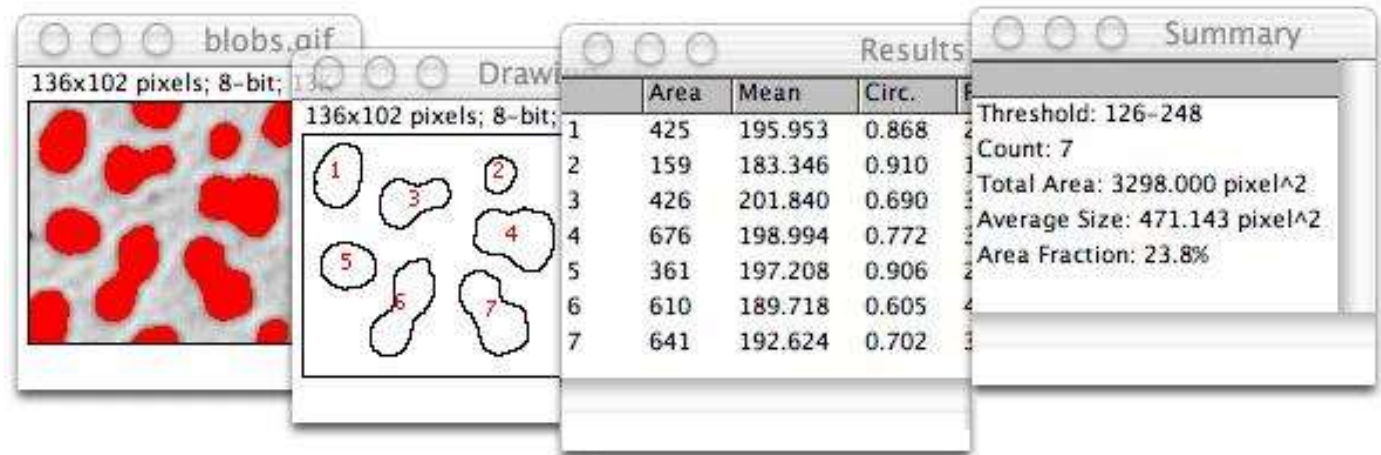


# Resources

## Analysis Tools + Logging Trigger

- Other Tools

- Imagej <https://imagej.net/Welcome>
  - Image processing tool



- Resources to Learn More

- Google Analytics Academy
  - <https://analytics.google.com/analytics/academy/>



# Triggers

## Analysis Tools + Logging Trigger

- What is a Trigger?
  - A trigger starts an action when it detects an event.
  - We can program a trigger to detect a specific event and “trigger” a function of our choosing
  - We will use a trigger to detect an event such as dropping our accelerometer, and begin data capture
  - Triggers form an important and powerful part of modern test equipment data collection strategy.

[https://en.wikipedia.org/wiki/Event-driven\\_programming](https://en.wikipedia.org/wiki/Event-driven_programming)

[https://en.wikipedia.org/wiki/Triggering\\_device](https://en.wikipedia.org/wiki/Triggering_device)

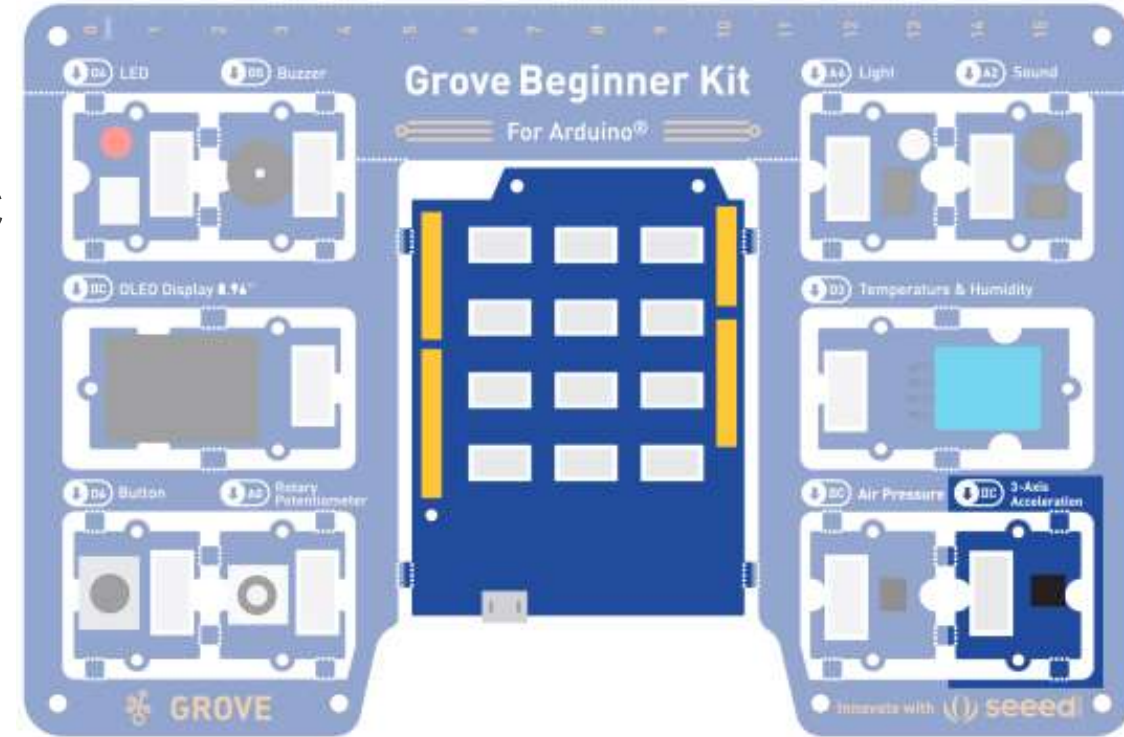
<https://www.electronics-notes.com/articles/test-methods/oscilloscope/oscilloscope-trigger.php>



# Hardware

## Accelerometer

- What hardware will we need for this Lesson?
  - Grove 3-axis Accelerometer Module on IIC
  - Seeeduino Lotus (Arduino Uno compatible board)



# Open and Upload Sketch

## Lesson 11: Logging Trigger Round 1

1. Open Simple Datalogger Sketch
  - File → Sketchbook → FRSEF\_Crash\_Course → Week\_5 → L11\_Trigger.ino
2. Upload the sketch to your Arduino by clicking the Upload Button.
  - The sketch should compile, and then upload to your Arduino, assuming you have the correct
3. Open the serial monitor
4. *Gently drop your board (from a couple inches)*
5. See the data stream to the serial monitor
6. Copy the data to your spreadsheet program
7. Look at the data in your spreadsheet program



# Activity

## Analysis Tools + Logging Trigger

- Import and plot x,y,z on a graph (use a scatter plot or line graph)
  - What data is present?
    - Beginning should show values very close to 0 for all axis
    - Then a large spike when the grove board hits a surface,
    - Secondary smaller spikes or oscillation may be seen as the board comes to rest with the magnitude around 1g

# Calculating the Magnitude of Acceleration

## Accelerometer

- What is the Magnitude?
  - Magnitude is the “length” or “size” of a vector.
  - For acceleration it is the amplitude of the acceleration
  - It has no direction
- How do we calculate the magnitude of our 3-
  - $Magnitude = \sqrt{X^2 + Y^2 + Z^2}$
  - Note the similarity to the Pythagorean Theorem

[https://en.wikipedia.org/wiki/Euclidean\\_vector](https://en.wikipedia.org/wiki/Euclidean_vector)

<https://byjus.com/maths/magnitude-of-a-vector/>

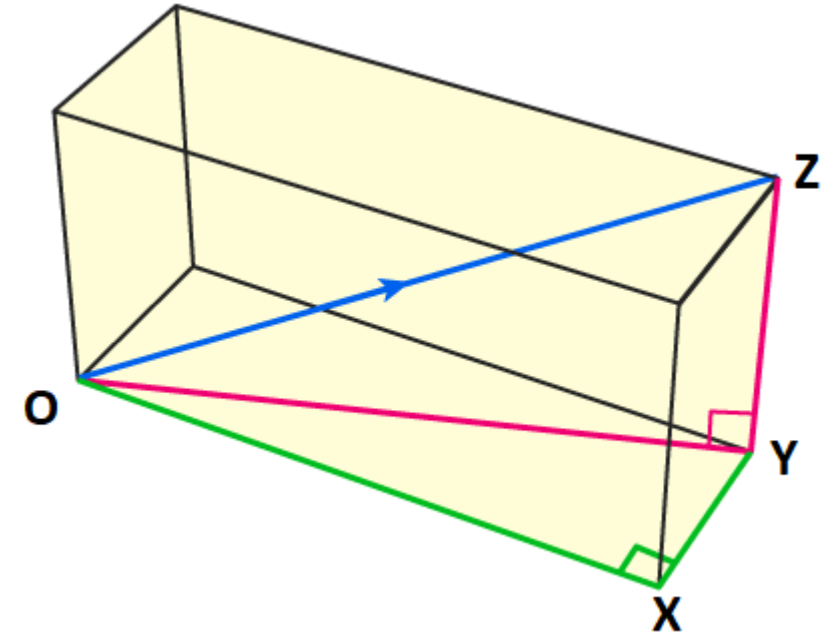


Image modified from <https://byjus.com/maths/magnitude-of-a-vector/>

# Code Analysis: vectorMagnitude () function(s)

## Accelerometer

```
float aMag = vectorMagnitude(float aX, float aY, float aZ);
```

- Functions written for you and included in code to calculate the magnitude.

```
// Calculate the magnitude of a 3D vector
float vectorMagnitude(float x, float y, float z)
{
    return sqrt(x*x + y*y + z*z);
}

// Calculate the magnitude of a 2D vector
float vectorMagnitude(float x, float y)
{
    return sqrt(x*x + y*y);
}
```





# Code Analysis: Drop detection

## Analysis Tools + Logging Trigger

- Drop detection is done by looking for the magnitude of acceleration to be close to 0g.

```
const float fallTrigger = 0.2; // in g, less than

// Detect falling and set trigger flag
bool triggerFall(float mag)
{
    if(mag < fallTrigger)
    {
        return true; // fall or drop detected
    }
    return false; // fall or drop not detected
}
```

# Code Analysis: Movement detection

## Analysis Tools + Logging Trigger

- Movement detection is done by looking for the magnitude of acceleration to be significantly greater than 1g or significantly less than 1g.

```
const float moveTriggerHi = 1.2; // in g, greater than
const float moveTriggerLo = 0.8; // in g, less than

// Detect movement and set trigger flag
bool triggerMove(float mag)
{
    if((moveTriggerHi < mag) || (mag < moveTriggerLo))
    {
        return true; // movement detected
    }
    return false; // movement not detected
}
```

# Code Analysis: Impact detection

## Analysis Tools + Logging Trigger

- Impact detection is done by looking for the magnitude of acceleration to be much greater than 1g.

```
const float impactTrigger = 4.0; // in g, greater than

// Detect impact and set trigger flag
bool triggerImpact(float mag)
{
    if(mag > impactTrigger)
    {
        return true; // impact detected
    }
    return false; // impact not detected
}
```



# Edit Sketch

## Analysis Tools + Logging Trigger

- Modify the sketch so we use the pushbutton as our logging trigger

### – Original

```
if(!triggered) // if not triggered, check for trigger event
{
    if(triggerFall(aMag) == true)
    //if(triggerMove(aMag) == true)
    //if(triggerImpact(aMag) == true)
    //if(triggerButton() == true)
```

### – New

```
if(!triggered) // if not triggered, check for trigger event
{
    //if(triggerFall(aMag) == true)
    //if(triggerMove(aMag) == true)
    //if(triggerImpact(aMag) == true)
    if(triggerButton() == true)
```



# Open and Upload Sketch

## Lesson 11: Logging Trigger Round 2

1. Open Simple Datalogger Sketch
  - File → Sketchbook → FRSEF\_Crash\_Course → Week\_5 → L11\_Trigger.ino
2. Upload the sketch to your Arduino by clicking the Upload Button.
  - The sketch should compile, and then upload to your Arduino, assuming you have the correct
3. *Push the button and then gently drop your board (from a couple inches)*
4. See the data stream to the serial monitor
5. Copy the data to your spreadsheet program
6. Look at the data in your spreadsheet program

# Activity

## Analysis Tools + Logging Trigger

- At Home Activity
  - Modify the sketch to start recording when the board is moved
  - Modify the accelerometer settings for logging rate and time logged



# Water Sensor + W3\_Chal

# Soil Moisture Types

## Sensor Calibration + Water Sensor

- Common Types

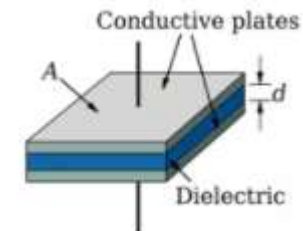
- Resistive

- Two probes
    - Current passes through the soil and the resistance value is calculated to measure soil moisture content.
    - Problems: sensor corrodes and resistance continually increases, fertilizer and nutrients can affect resistance
    - Quantitative measurement.

- Capacitive

- Single probe, soil contact with electrodes not required (less corrosion)
    - Soil + water form a dielectric, similar to a capacitor
      - Capacity of soil change with change of moisture content
    - Quantitative measurement

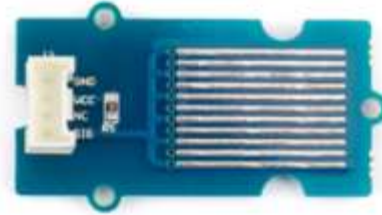
[https://ucanr.edu/sites/CE\\_San\\_Joaquin/files/35895.pdf](https://ucanr.edu/sites/CE_San_Joaquin/files/35895.pdf)



# Moisture Sensors

## Sensor Calibration + Moisture Sensor

- How it works: uses the conductivity of water to “short” between GND and the signal input
  - Digital Input: water is present or not
  - Analog Input: level of water present
    - Voltage divider with pullup resistor and water between the signal and ground lines.



# Hardware

## Sensor Calibration + Water Sensor

- What hardware will we need for this Lesson?
  - Grove LED, Buzzer and Display
  - Seeeduino Lotus (Arduino Uno compatible board)
  - Grove Water Sensor
    - Connect to **D2 Header**
    - Use the provided cable



# Open and Upload Sketch

Sensor Calibration + Water Sensor (This is the W3\_Chall)

1. Open Simple Datalogger Sketch
  - **File > Sketchbook > CrashCourse\_Jan > L13\_Water\_Alert.ino**
2. Upload the sketch to your Arduino by clicking the Upload Button.
  - The sketch should compile, and then upload to your Arduino, assuming you have the correct
3. Touch the water sensor, see what happens.
  - What happens if you place a drop of water on the sensor or dip the sensor into a cup of water?

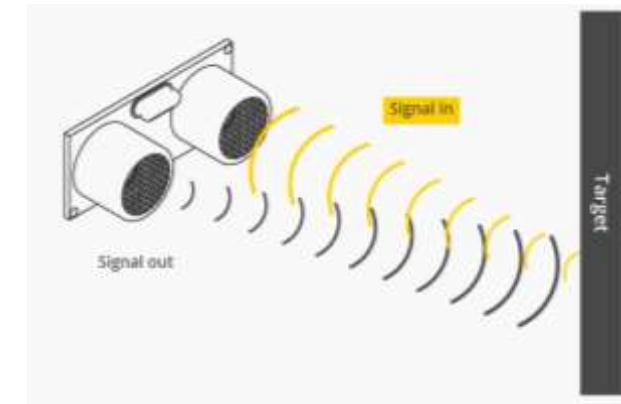


# Sensors & Applications



# Sensors & Applications: Ultrasonic Sensors

- Application using microphones: ultrasonic sensors
  - Ultrasonic Sensors
    - Simple range finders
    - Object position and tracking
      - Example: automotive active safety



- <https://www.seeedstudio.com/blog/2020/01/03/what-is-a-sound-sensor-uses-arduino-guide-projects/#:~:text=A%20sound%20sensor%20is%20defined,converting%20it%20to%20electrical%20signals>

# Sensors & Applications: Ultrasonic Sensors

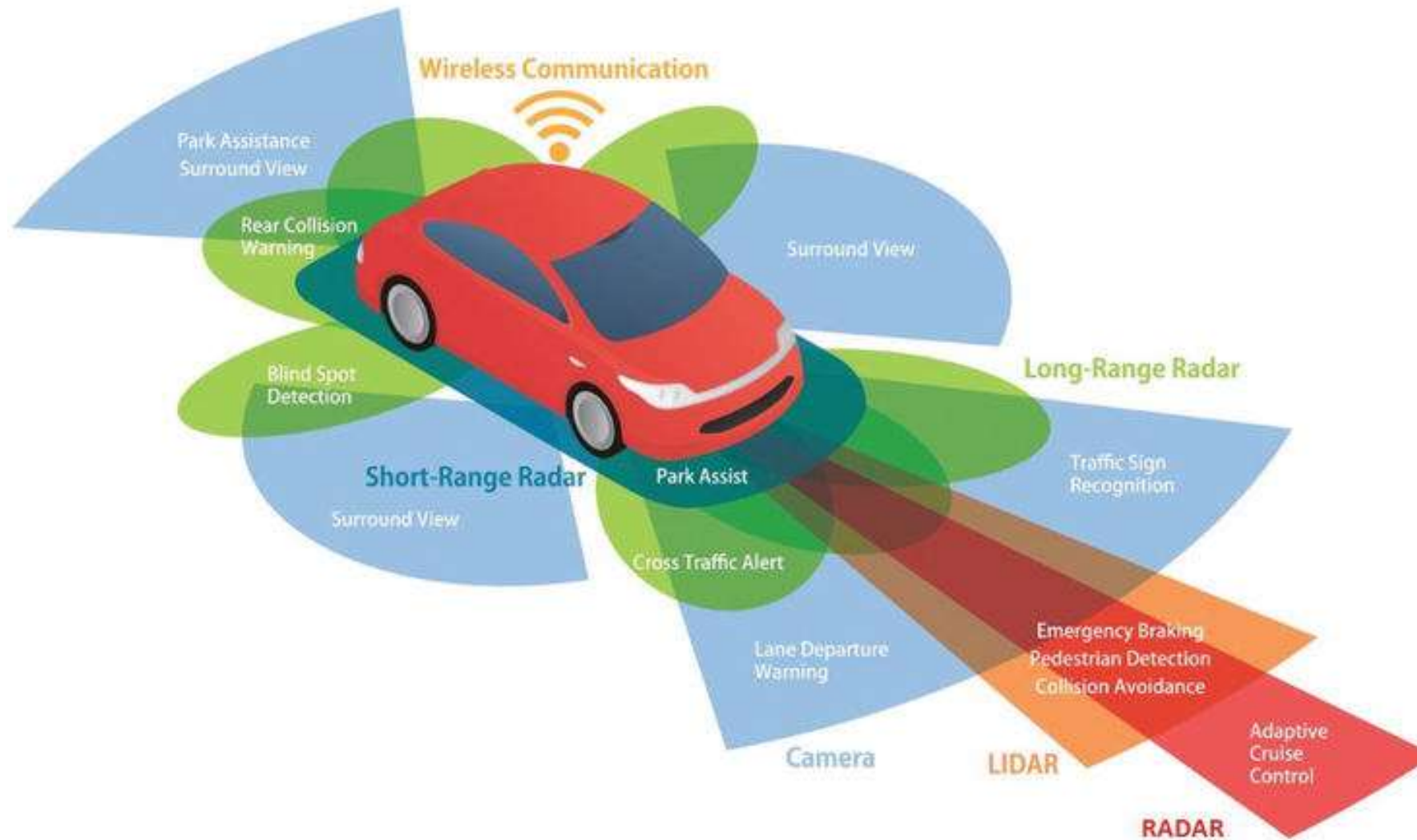
- Ultrasonic Sensors
  - Object position and tracking
    - Example: automotive active safety



# Sensors & Applications: Autonomous Driving

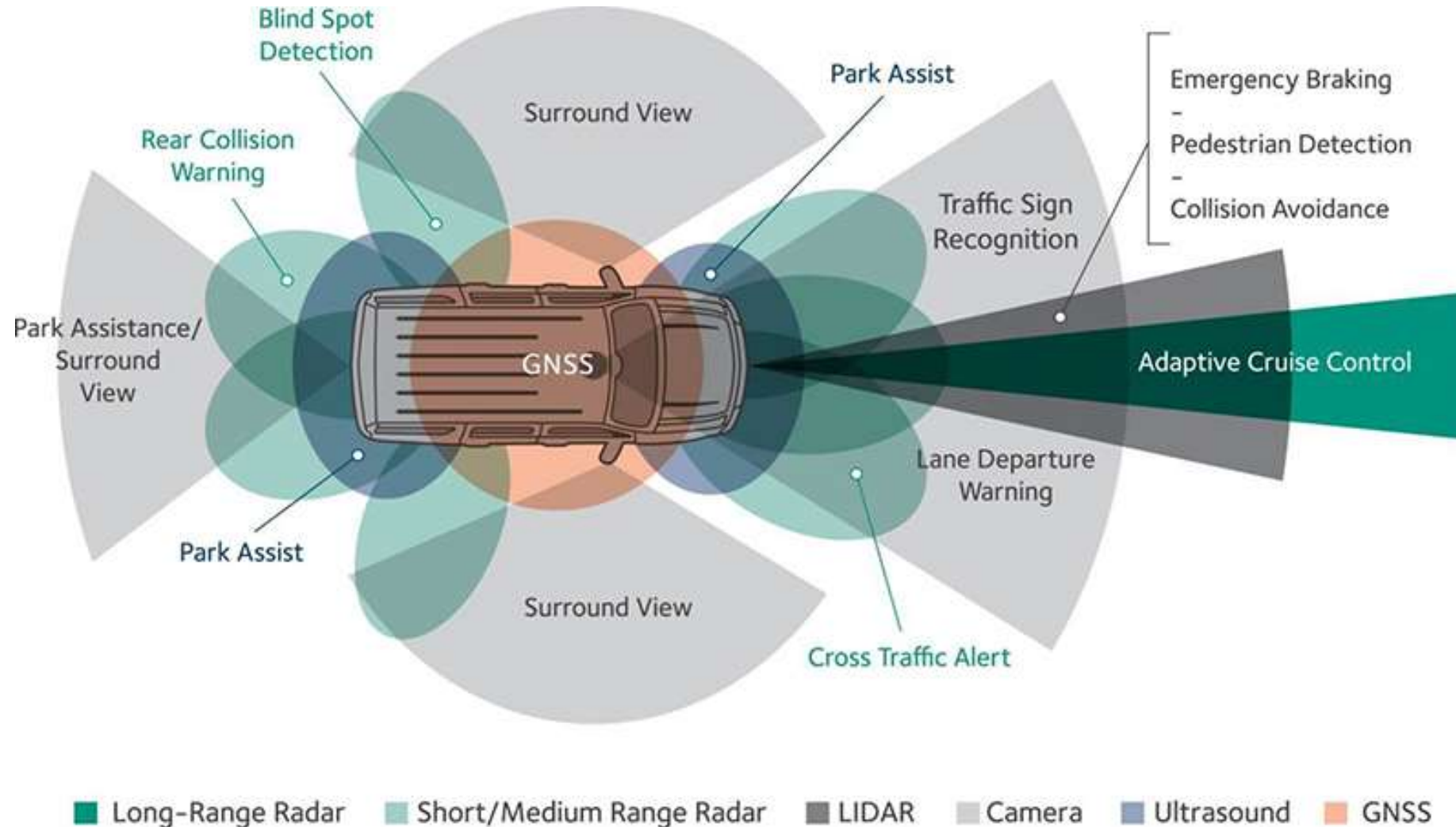
- Autonomous Driving
  - IMU (accel, gyro)
  - Location
    - GNSS (Global Navigation Satellite System) or GPS
  - External Object Detection
    - Radar
    - Lidar
    - Ultrasonic
    - Cameras
  - Vehicle Sensors
    - Wheel speed, steering angle

# Sensors & Applications: Autonomous Driving



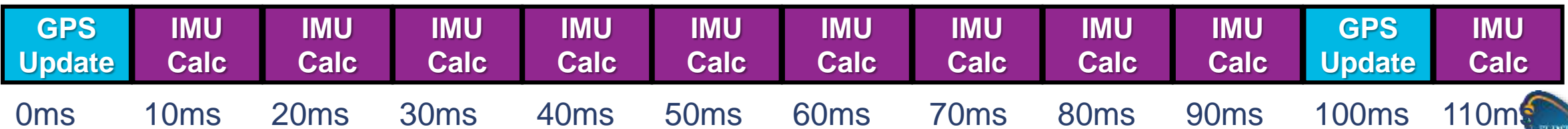


# Sensors & Applications: Autonomous Driving



# Sensors & Applications: Autonomous Driving

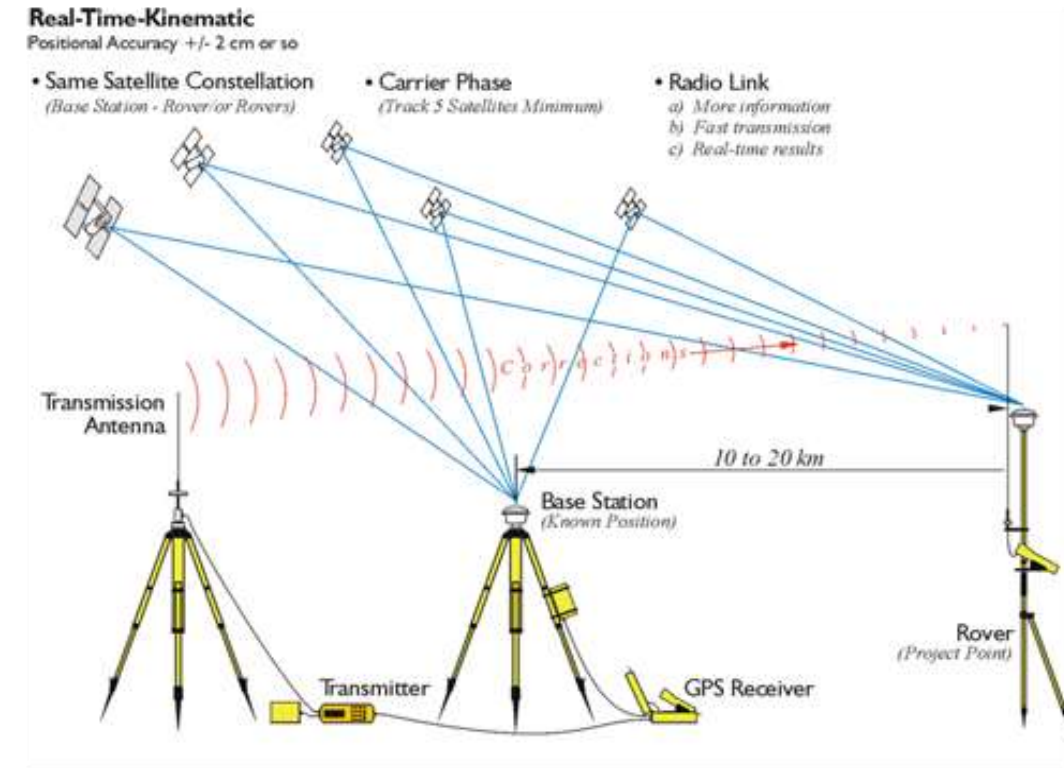
- GNSS (Global Navigation Satellite System)
  - Multiple systems in operation
    - GPS (US), Galileo (EU), GLONASS (Russia), Beidou (China)
  - Time clock
  - Update rate limitations
    - Common: 10 – 20 Hz
    - Mobile Phone: 1 Hz
    - Use IMU data to calculate location between GPS updates (gps imu fusion)
    - Location Calculation: integrate acceleration to get location, small errors start to add up (drift)





# Sensors & Applications: Autonomous Driving

- GNSS (Global Navigation Satellite System)
  - Correction services: clock drift, orbit errors, atmospheric errors
    - PPP: Precise Point Positioning – global correction
    - RTK: Real Time Kinematic – local base station that offset data
    - DGPS: Differential GPS – base station that provides offset data



<https://www.taoglas.com/centimeter-level-positioning-with-single-band-rtk/>

<https://www.e-education.psu.edu/geog862/node/1828>

# Sensors & Applications: Autonomous Driving

- Radar
  - Output radio waves and measure signals received back.
  - Short Range Radar: ~24 GHz
    - 1 to 30 m
  - Long Range Radar: ~77 GHz
    - 3 to 80 – 200m
  - Pros: Robust for weather, low cost
  - Cons: Lower resolution



# Sensors & Applications: Autonomous Driving

- Radar examples



# Sensors & Applications: Autonomous Driving

- Lidar
  - Illuminates a target with a laser and measures the characteristics of the reflected return signal.
    - Also called laser scanner, laser radar
  - Range: 0 – 200m
  - Pros: High resolution 3-D mapping, identify and classify objects
  - Cons: problems in rain, fog, snow; high cost
  - Mechanical Lidar: rotating assembly, 360° view, cost, size, robustness
    - “Orb” of google cars
  - Solid State Lidar: no spinning parts, multiple at front, rear and side combined together

[https://www.ti.com/lit/wp/slyy150a/slyy150a.pdf?ts=1607565385866&ref\\_url=https%253A%252F%252Fwww.google.com%252F](https://www.ti.com/lit/wp/slyy150a/slyy150a.pdf?ts=1607565385866&ref_url=https%253A%252F%252Fwww.google.com%252F)



# Sensors & Applications: Autonomous Driving

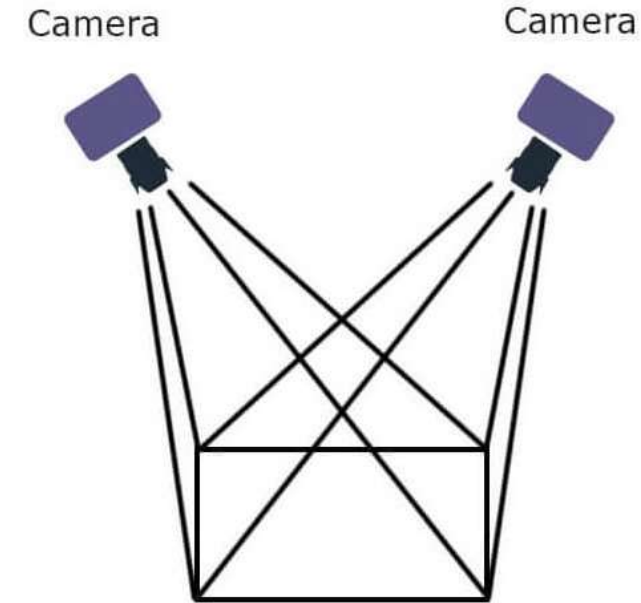
- Lidar
  - Mechanical Lidar: rotating assembly, 360° view, cost, size, robustness
    - “Orb” of google cars
  - Solid State Lidar: no spinning parts, multiple at front, rear and side combined together
    - Hundreds to thousands of lasers on each module
    - iPhone 12 Pro => better photos, 3D scanning, augmented reality



# Sensors & Applications: Autonomous Driving

- Camera

- Used to detect objects and classify objects
- Mono camera: single camera
  - Object detection and classification
- Stereo camera: two cameras at known offset
  - Spatial awareness: measure distance and improved size calculations
  - Pros: High resolution 3-D mapping, identify and classify objects
  - Cons: High cost, high computing and software requirements





# Operators

## Lesson 10

# Code Analysis – C++ Comparison Operators

```
for(unsigned int i = 0; i < 1<<filterConstant; i++)
```

– Is **i** less than  $2^{\text{filterConstant}}$ ?

==	Equal To	→ TRUE if the left side is <u>equal to</u> the right side
!=	Not Equal To	→ TRUE if the left side is <u>not equal to</u> the right side
<	Less Than	→ TRUE if the left side is <u>less than</u> the right side
<=	Less Than or Equal To	→ TRUE if the left side is <u>less than or equal to</u> the right side
>	Greater Than	→ TRUE if the left side is <u>greater than</u> the right side
>=	Greater Than or Equal To	→ TRUE if the left side is <u>greater than or equal to</u> the right side

- More Information:

- <https://beginnersbook.com/2017/08/cpp-operators/>
- <https://www.arduino.cc/reference/en/>



# Code Analysis – C++ Arithmetic Operators

## Operators

+	Addition	$A = 1 + 2 \rightarrow A = 3$
-	Subtraction	$B = 3 - 1 \rightarrow B = 2$
*	Multiplication	$C = 2 * 4 \rightarrow C = 8$
/	Division	$D = 6 / 3 \rightarrow D = 2$
%	Modulo (remainder)	$E = 7 \% 4 \rightarrow E = 3$

- More Information:

- <https://beginnersbook.com/2017/08/cpp-operators/>
- <https://www.arduino.cc/reference/en/>



# Code Analysis – C++ Auto-increment and Auto-decrement Operators

## Operators

```
for(unsigned int i = 0; i < 1<<filterConstant; i++)
```

- Increment `i` by 1 at the end of the for loop.

**++**    Auto-increment                      `i++`     $\rightarrow$     `i = i + 1`

- Increments the value of a variable by 1

**--**    Auto-decrement                      `j--`     $\rightarrow$     `j = j - 1`

- Decrements the value of a variable by 1

- More Information:

- <https://beginnersbook.com/2017/08/cpp-operators/>
- <https://www.arduino.cc/reference/en/>



# Code Analysis – for Loop

## Operators

```
for(unsigned int i = 0; i < 1<<filterConstant; i++)  
{ /* Do Something */ }
```

– Repeat code inside the curly braces  $2^{\text{filterConstant}}$  times

- **for ()** Loops are used to repeat code that appears between its curly braces

• Syntax:

Iterator Variable InitializationUpdate Iterator

```
for(initialization; condition; increment)  
{  
    // Do Something  
}
```

End Condition

- More Information:

- <https://www.arduino.cc/reference/en/language/structure/control-structure/for/>
- <https://beginnersbook.com/2017/08/cpp-for-loop/>



# Code Analysis – C++ Assignment Operators

## Operators

```
micValueLong += micValue;
```

- Add **micValue** and **micValueLong** then store the result in **micValueLong**

=	Equals	Assigns value of right side to the left side		
+=	Plus Equals	A += 2	→	A = A + 2
-=	Minus Equals	B -= 3	→	B = B - 3
*=	Multiply Equals	C *= 4	→	C = C * 4
/=	Divide Equals	D /= 5	→	D = D / 5
%=	Modulo Equals	E %= 6	→	E = E % 6

- More Information:

- <https://beginnersbook.com/2017/08/cpp-operators/>
- <https://www.arduino.cc/reference/en/>





# Code Analysis – Averaging Filter

## Operators

- What is a filter?
  - A filter is used to remove an unwanted component of a signal.
  - For sensor measurements a **low pass filter** is often used to reduce noise or some high frequency component.
  - There are many different types of filters, and numerous ways to implement filters.
- What is averaging?
  - Averaging is taking the mean value of a signal over the sampling period.
- More Information:
  - [https://en.wikipedia.org/wiki/Filter\\_\(signal\\_processing\)](https://en.wikipedia.org/wiki/Filter_(signal_processing))
  - <https://en.wikipedia.org/wiki/Average>
  - <https://www.mathsisfun.com/mean.html>

```
// Average filter
int average;
int sumSamples = 0;
for(int i = 0; i < numSamples; i++)
{
    sumSamples += analogRead(A2);
}
average = sumSamples / numSamples;
```