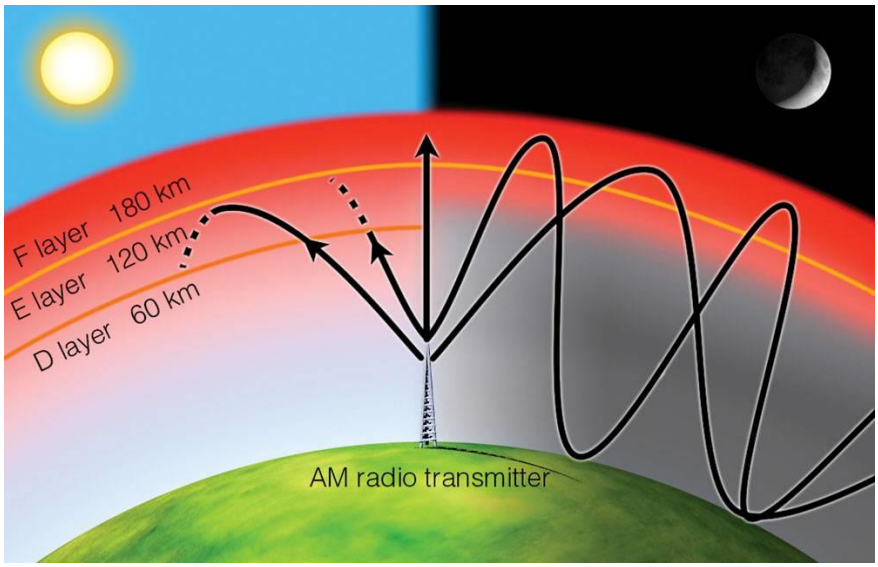
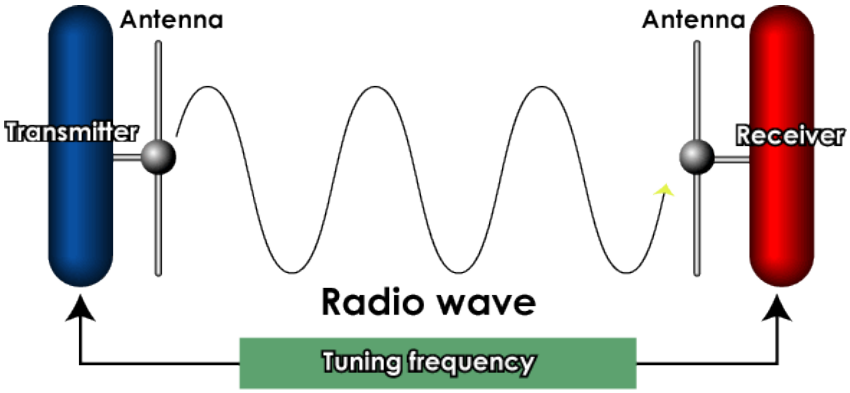
Radio Frequency (RF)

Radio antennas translate information into radio waves. A radio wave is literally electric radiation all around us, going through buildings and even people, at varying speeds and power. By sending information through radio waves, data is transmitted all over the world without wires. Different frequencies can be used to transmit many different types of data at the same time in the same area.



Most radio waves bounce around inside the atmosphere, though some escape into space. What are the aliens listening to right now?

The frequency (or speed) of the data, known as the Hertz (times per second), describes how many times the waves go up and down each second. To send data, a transmitter sends pulses of electromagnetic radiation at a specific frequency. The receiver is listening for data to come in at the assigned frequency, and if the right pulses of power are received, the receiving circuit can translate the pulses into bits of information.



Wireless phones and internet routers work at 2.4 million Hertz, the same frequency as microwave ovens.

Smart Car Version 4 - Wireless Control

Watching the robot run on its own is fun and all, but it's time to take control for ourselves! For that, we'll need a remote control system. To really ramp up our bots, we're breaking into the unlicensed 2.4 GHz radio spectrum, giving us fast and long distance communications!

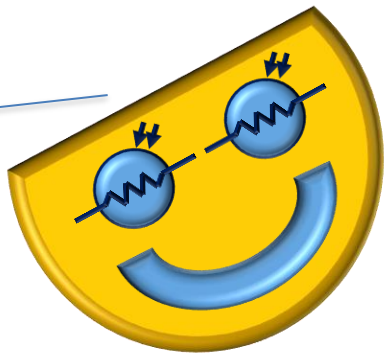
Bluetooth

Bluetooth is a short-range wireless technology using the 2.4 GHz wavelength, the same band as microwaves, wireless routers, and cell phones. Typical range is 33 feet, though some version 5 devices can reach 800 feet. Each Bluetooth "Master" device can communicate with up to 7 "Slave" devices at a time, and 79 channels are typically used to allow for multiple devices in close proximity. Bluetooth uses much less energy, and consequently is much slower and lower range than typical wireless found in wireless devices using 802.11, such as home routers.

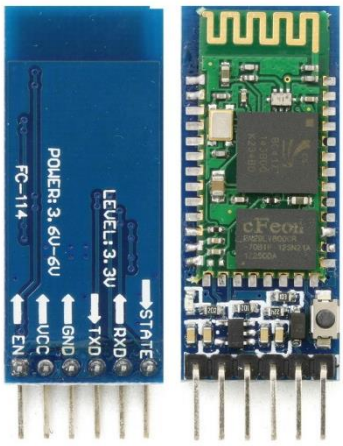
This technology is named after Nordic King *Haraldr blátǫnn Gormsson*, translated *Harald "Bluetooth" Gormsson* in English, who lived around the 10th Century and united the differing tribes of his region into a single kingdom. The idea behind the name was that the new wireless standard would help to bring technology together in a similar way.

The hardware does a lot of the work for us. By setting each Bluetooth pair of devices to be bound to one another, we can skip a lot of the usual software needs of finding devices, connecting, sending, and encrypting messages. However, deciphering the method of sending a receiving data is still on us, as you'll see soon.

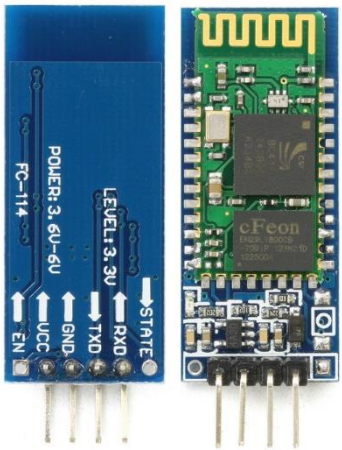
Nordic King "Bluetooth" and Italian King "Arduino" both lived in late 10th century Europe. Almost fitting these contemporary kings should meet again 1000 years later.



HC-05 FC-114



HC-06 FC-114

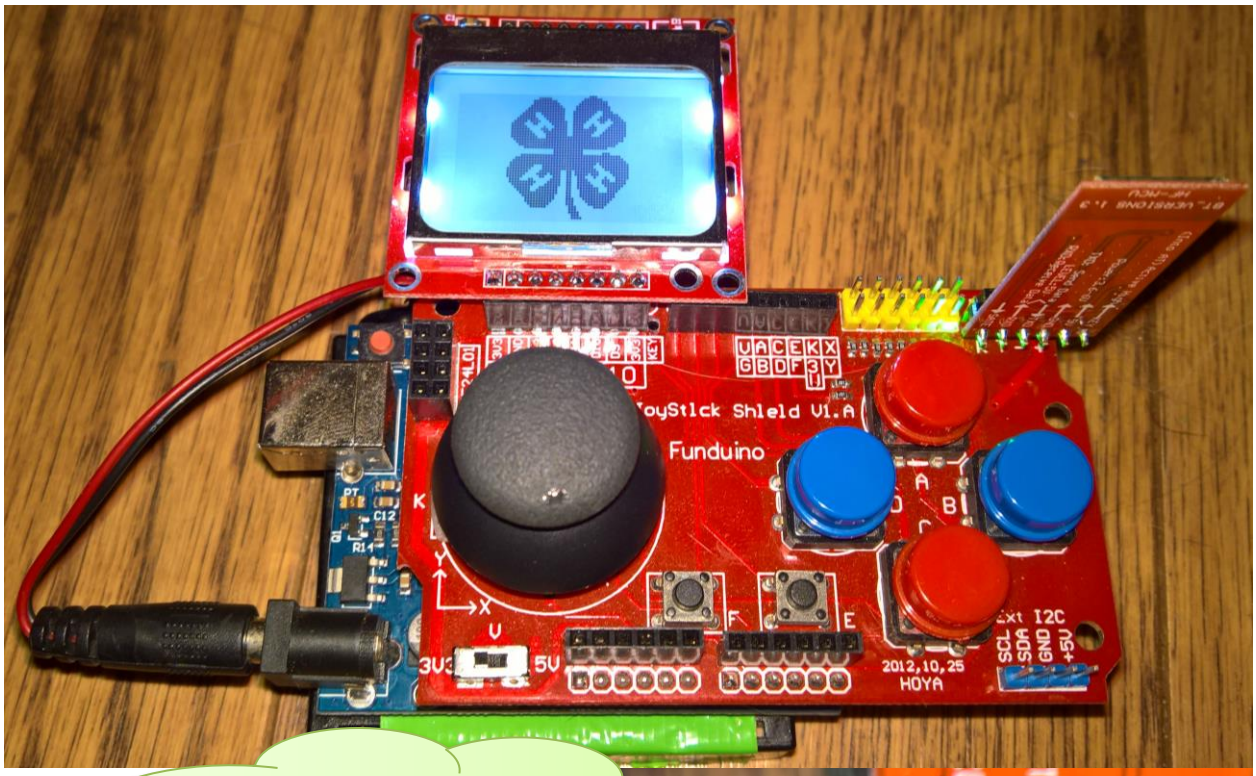


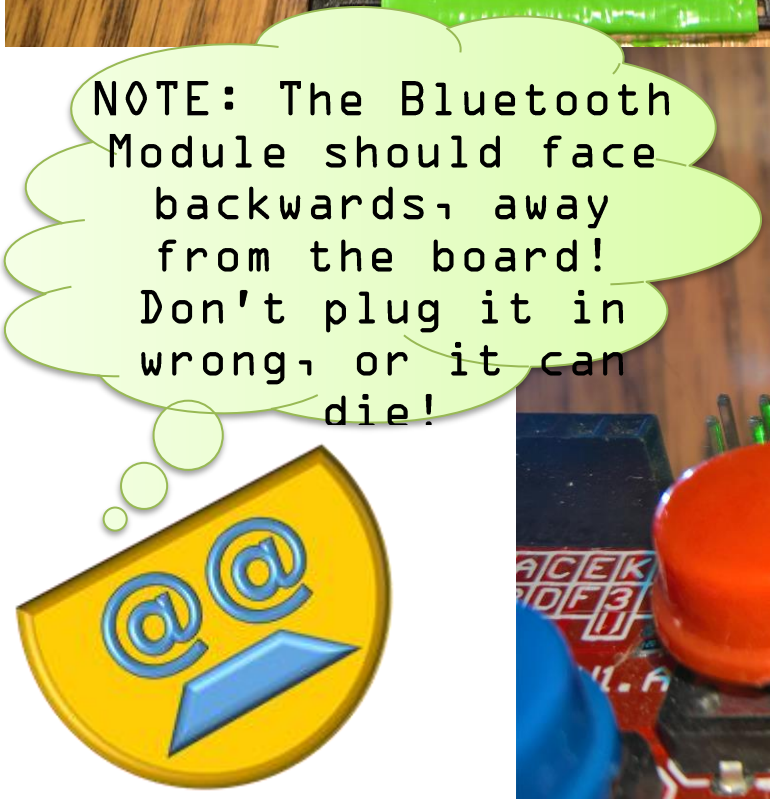
The HC-05 (Master/Slave) Bluetooth module controls the communication with the HC-06 (Slave) module, allowing for secure and constant communication without extra software commands. (We won't get into the pairing of these devices here, but if you want to see how to do it yourself, go to the site below.)

http://www.martyncurrey.com/arduino-to-arduino-by-bluetooth

Remote Transmitter Hardware

Our controller hardware is pretty straightforward: plug the Joystick Shield onto the Arduino, then plug in Bluetooth module onto the Joystick Shield (***make sure the pins go in the correct holes***). Add the 5110 LCD module, and give it power! (Note: we're using a red HC-05; it matches our remote!)





Liquid-Crystal Displays

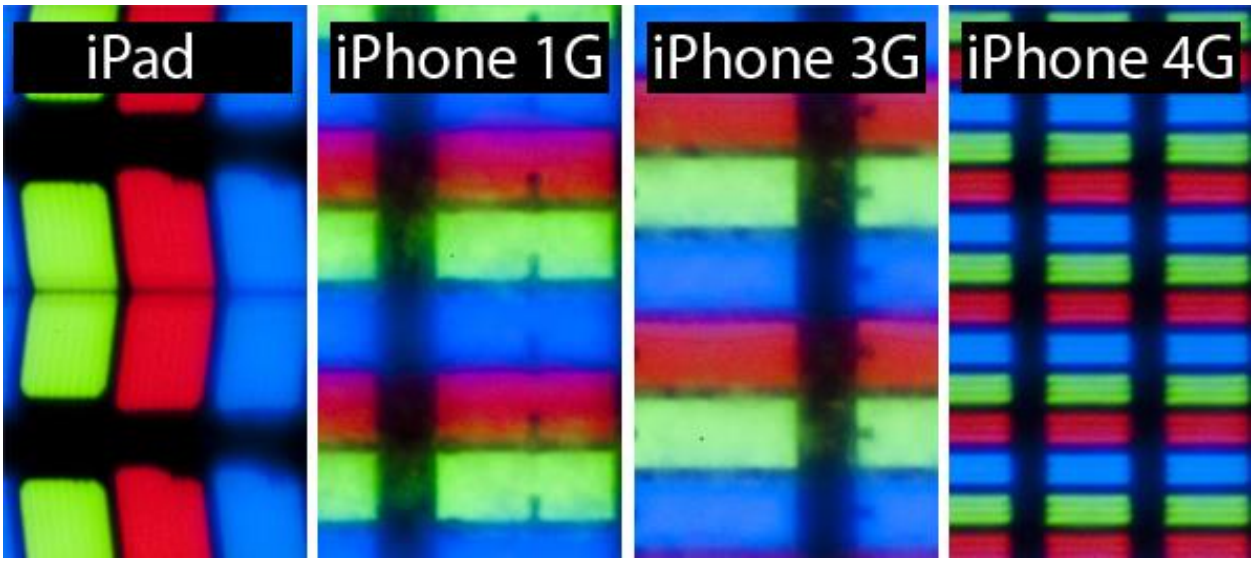
The "Liquid Crystal" in LCD isn't a liquid crystal. It's actually a mixture of liquid and crystals.

An LCD has two pieces of glass (or other appropriate material) that are both polarized and between them is sandwiched a liquid with crystals suspended in it. Polarized glass will only let through light that is oriented a certain way (horizontal vs. vertical, for the purpose of this explanation); if you have one horizontal and one vertical filter, then no light can get through the display.

Normally, the crystals will just align themselves however they like- what results is that the crystals line up with the polarized glass above and below, creating a spiral like structure that bends the light that comes through it. What this means is that you can have a horizontal filter on one side, the vertical filter on the other, and the crystals will act to bend the light from horizontal to vertical as it passes through it- effectively making the whole thing see-through.

When you apply a current, the crystals all line up to the current, which removes the spiral structure that it normally takes on. Since the light is no longer being bent from horizontal to vertical, this disables the see-through property.

So, how does this change color? Well, the basic liquid crystal is what you see in cheap calculators and digital watches. In a screen like for a computer, the individual LCD units are really really tiny, and each one controls a single color. What you end up with is a square with three LCD elements stacked in it, each one with a filter for red, blue, or green Each one controls its own color and combines them to create the color you see. For instance, for purple, the blue and red LCD elements are allowed to let light through, and the green element is energized to let no light through. The light itself is created by a light panel that is behind the entire screen and is generating white light. On something like a calculator, or older game systems like the original Gameboy, the system relies on light passing through the screen, bouncing off of a backdrop, and passing back through the screen again.



They're like little digital stain-glass windows with electric shutters.

Nokia 5110/3310

The LCD displays we're using today were used in old Nokia 5110/3310 cell phones (before the smart-phone fad turned every cell phone into a TV). It's a 84x48 pixel monochrome LCD display. These displays are small, only about 1.5" diagonal, but very readable and come with a white backlight. This display is made of 84x48 individual pixels, so you can use it for graphics, text or bitmaps. These displays are inexpensive, easy to use, require only a few digital I/O pins and are fairly low power as well.

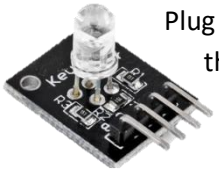
(Interesting fact: these screens went out of production years ago, but so many were made, that the market is still flooded with cheap monochrome LCDs that never got the chance to be in a phone!)

We're using our Ultrasonic Sensor robot from last session!

Leave the sensor and servo plugged in like last time.

Car Receiver Hardware

The wiring for the receiver on the car is only a touch more complicated. First, plug 4 male-to-female wires onto the Bluetooth module. Then plug in the Bluetooth 5V (this goes to the 3V pin; don't worry, this is fine for this module) and GND on the board, then the RX to pin 4, and TX to pin 5.



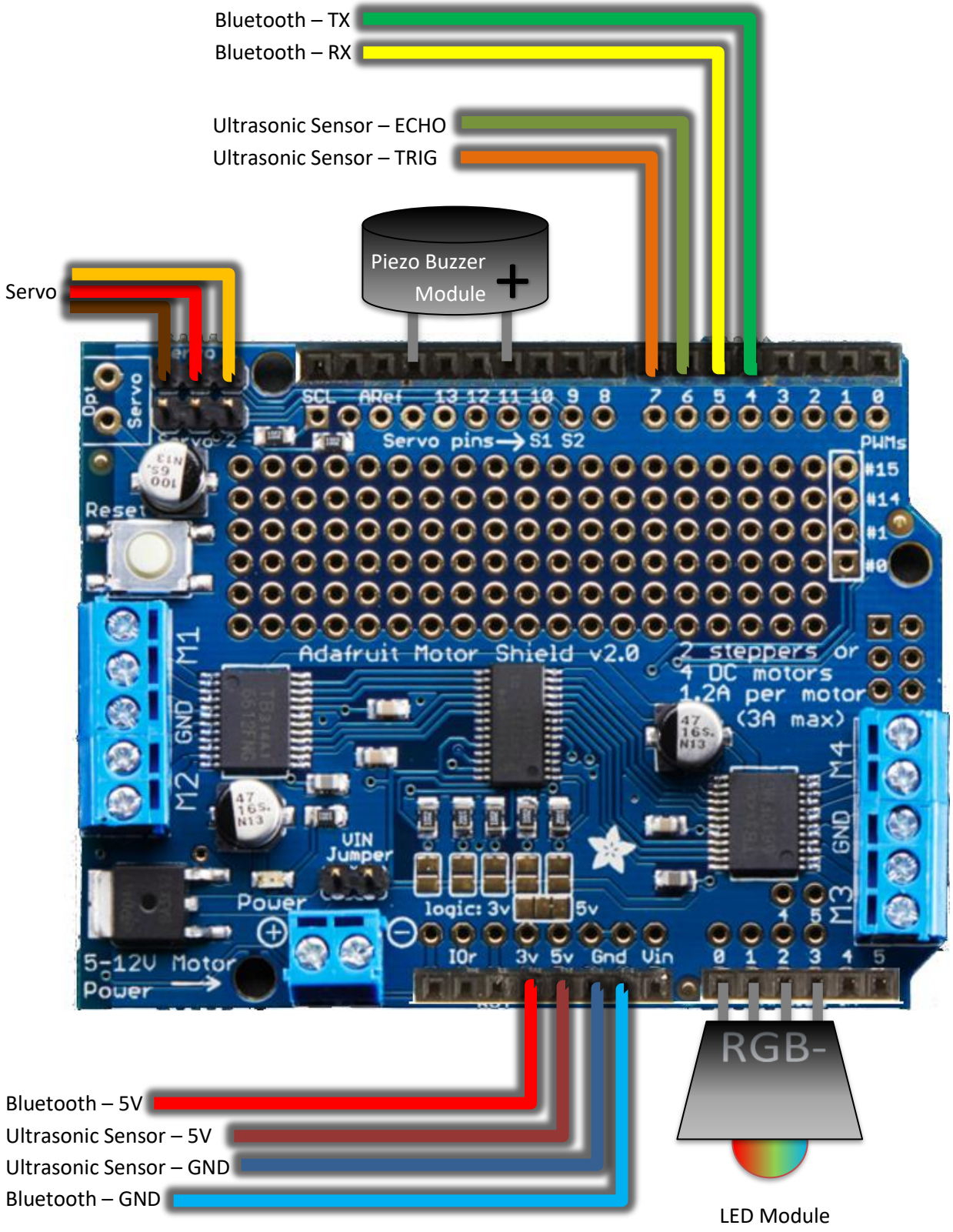
Plug the RGB LED module into the Analog pins A0-A3, making sure it is facing away from the board. NOTE: We're using pin A3 as a ground pin for this module; this is not something you would normally do, and only works because we're using it just to power an LED. Anything more on an IO pin like this could kill the Arduino.

Finally, plug in the Piezo buzzer module, with the "=" positive pin to pin 11 on the board and the shorter "-" negative pin to the GND pin on the same row. You may need to slightly bend these pins.

A diagram is shown on the next page.

(Reminder: we are still using the Adafruit Motor Shield and Ultrasonic Sensor setup from the last session. See the previous project for wiring assistance.)

Wiring Diagram



Program

This is going to be a bit long and complex looking, but bear with me. The robot we're building today has a lot of things it can do, and there's a lot that needs to happen to make each part work together. To better understand how this all works, it helps to create a "flowchart" with all the steps involved.

The two programs start much in the same way. First, we power on the Arduino and attached components. Then, we tell the Arduino the "Libraries" to use to make our coding job easier. Then any variables, pins, and components are assigned in the program. Finally, we initialize the sensors, screens, Bluetooth, and any other attached parts.

Now we can begin the main "Loop" of each of the programs.

Transmitter

In the Transmitter program, we start the Loop by clearing the display, then reading the inputs from the joystick and buttons.

Next, we check for incoming data on the Bluetooth serial connection, and if it is valid, we "parse" it, meaning divide it into chunks we can use. These data chunks are checked for error, and if they are all good, we store this received data in variables.

We are using an "Online Timer" to help determine if the connection is still working. If there is no valid data for over a second, then the LCD display will say "Offline..." However, every time valid data is received, the timer is reset, and the display shows "Online."

Included in the received data is the "Mode" the robot is in; Manual, Auto, or Demo. After displaying the Mode setting on the LCD, if in Manual Mode, the program will also display the current LED Color setting on the LCD.

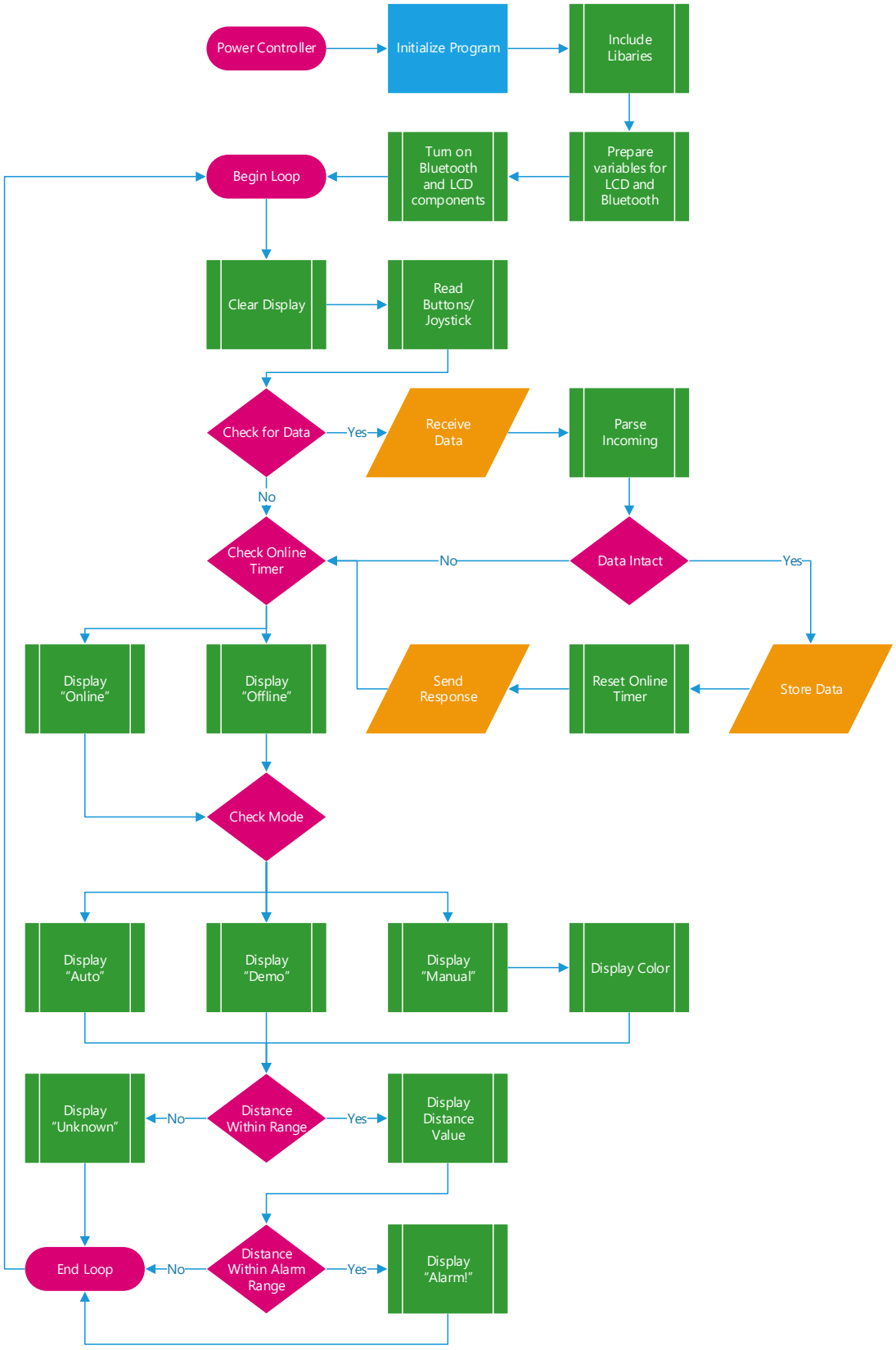
Next, the program determines if the Distance value received is within a valid range, and displays either the number of centimeters or "Unknown." If the value is within a valid range, and is too close, it then displays "Proximity Alert" as well.

Missing from the flowchart is the Servo information. The program checks the current angle the Servo head is pointing on the bot, and displays the degrees right or left along with an arrow indicating right, left, or forward.

When the program ends, it starts over at the Loop section again.

See the Flowchart on the next page for a visual representation of this.

Transmitter Flowchart



Receiver

The Receiver program starts much the same way as the Transmitter. The Libraries are included, the variables prepared, and the components initialized.

The Loop begins again with a check for incoming data, checking data integrity, parsing, and storing as variables for later use. The first check is the Horn, which plays a tone when the horn button variable indicates it was pressed.

Now, the Mode is set based on inputs. The E Button, where one might find the "Start" on a traditional Nintendo gamepad, switches the robot between Manual and Auto Mode. The F Button, where the "Select" button might appear, causes the robot to go into a short Demo mode, where it dances and plays music for a few seconds before returning to either Auto or Manual.

In Manual Mode, the program checks the Joystick input, and if it is higher than the default value from when the robot turned on, the program adjusts the values and recalibrates the Joystick automatically. After this, the angle of the Joystick is used to determine the speed and direction the motors turn. This Mode then checks if the Blue or Red buttons on the gamepad were pressed, that is buttons A, B, C, or D. Buttons B and D, the right and left Blue buttons, move the Servo head left and right, so long as it's not already at the minimum or maximum setting. A and C, the top and bottom Red buttons, change the Color value, which is then sent to change the LED color.

In Auto mode, the motor speed is first set to a default setting. Then the Ultrasonic Sonar sensor checks the distance between the robot and any obstacles. If nothing is detected in range, the Servo head sweeps right and left as the car drives forward, flashing the LED blue and red. If an obstacle is found, the LED turns purple and the angle of the head determines if the car backs up turning right, left, or straight back.

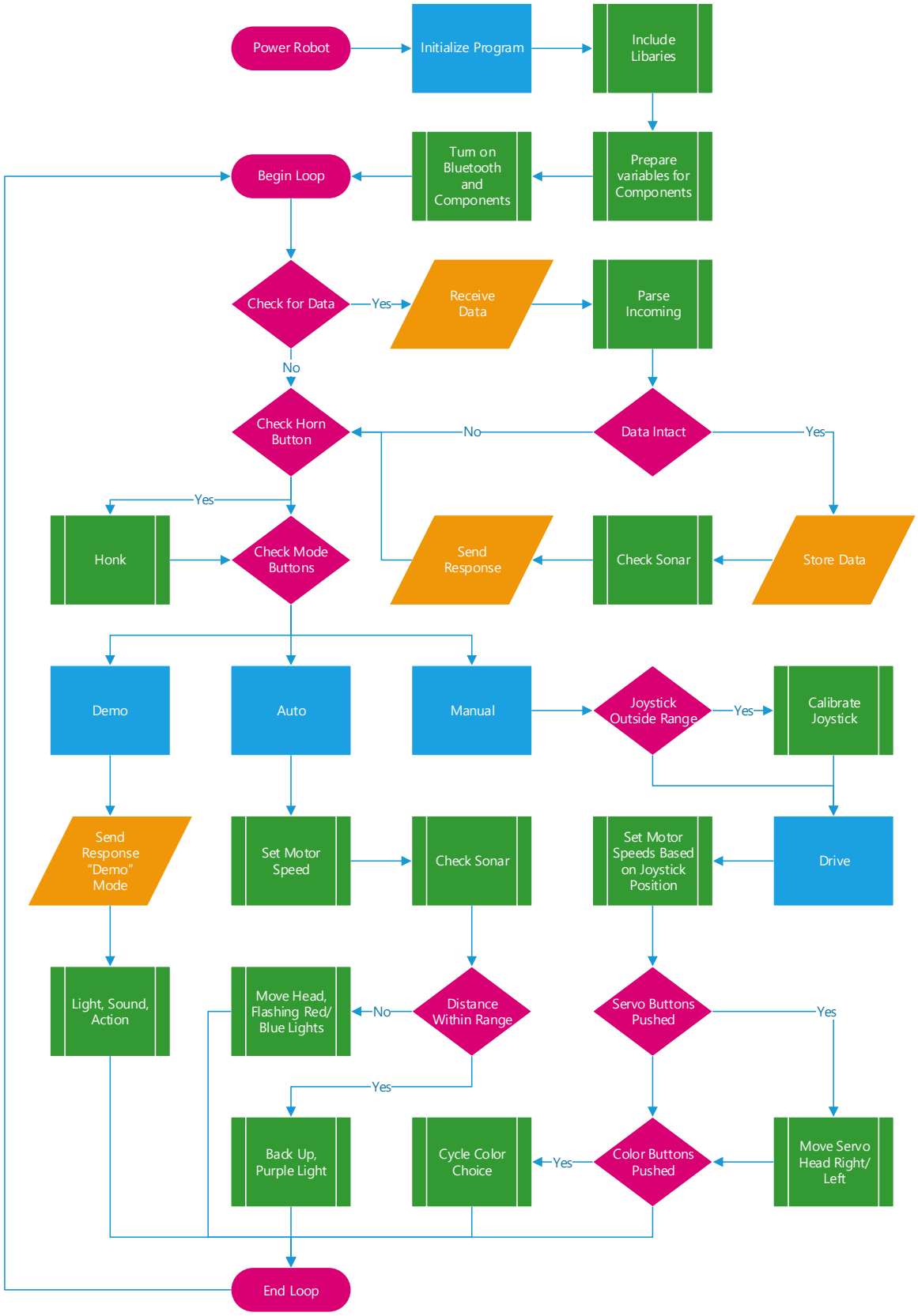
The program then completes by returning to the beginning of the Loop and repeating it over and over again.

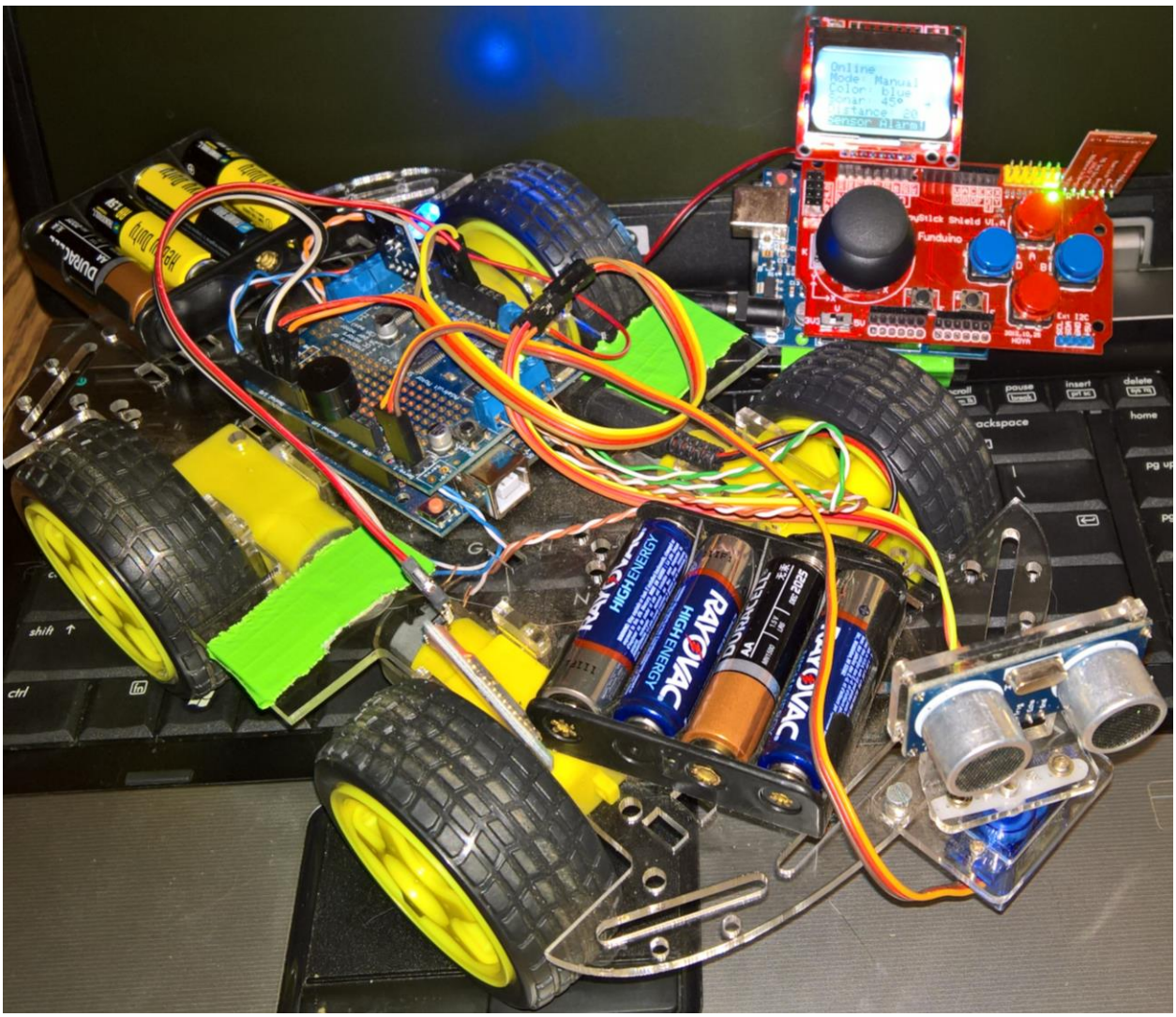
Code

Would you believe it? I made the code too long! 18 pages, to be exact! So, rather than kill a forest, I've put the code online!

Visit https://github.com/sagebrush4hrobotics/BluetoothCar for the complete code, links, and more.

Receiver Flowchart




Online
Mode: Manual
Color: blue
Funduino
HIGH ENERGY
RAYOVAC


Finally!
Let's Drive!

Sources

http://www.martyncurrey.com/arduino-to-arduino-by-bluetooth/

http://www.martyncurrey.com/connecting-2-arduinos-by-bluetooth-using-a-hc-05-and-a-hc-06-pair-bind-and-link/

http://jhaskellsblog.blogspot.com/2011/05/serial-comm-fundamentals-on-arduino.html

http://jhaskellsblog.blogspot.com/2011/06/parsing-quick-guide-to-strtok-there-are.html

http://techshorts.ddpruitt.net/2013/08/remote-controlled-robotank_99.html

https://github.com/adafruit/Adafruit_Motor_Shield_V2_Library

https://bitbucket.org/teckel12/arduino-new-ping/wiki/Home

https://bitbucket.org/teckel12/arduino-timer-free-tone/wiki/Home

http://forum.arduino.cc/index.php?topic=106043.msg1312639.html#msg1312639

http://tutorial.cytron.com.my/2015/09/04/softwareserial-conflicting-with-servo-interrupt/

https://learn.adafruit.com/adafruit-gfx-graphics-library

https://github.com/adafruit/Adafruit-PCD8544-Nokia-5110-LCD-library

This code can be found at https://github.com/sagebrush4hrobotics/BluetoothCar

Also check out the Sagebrush 4-H Robotics Blog for previous lesson plans:

http://sagebrush4hrobotics.blogspot.com/