Android controlled Arduino Robot Car

In this post I will write my hands-on experiences about how to use an Android phone to control a robot car, which is based on an Arduino Uno board. After a couple of weeks I assembled a robot car and programmed a very simple App that can send commands to the Arduino with the help of Bluetooth. Let's dive in!

Hardware Requirements:

- Arduino Uno board
- L298N Dual H-Bridge Motor Driver Shield
- Bluetooth Transceiver Chip (Serial Bluetooth Modem)
- Vehicle kit
- A breadboard and some jumper wires

After some googling I found a good choice from the DX website. Actually these components can be ordered on some other websites ex. Sparkfun, Adafruit or Amazon.

Software Requirements:

- Arduino IDE (http://arduino.cc/en/main/software)

The complete project will be explained step by step: in the first part I will introduce Arduino very shortly. L298N Motor Driver Shield will be explained in the second part. Next I will describe how to use our Bluetooth module. After we know how to program Motor and Bluetooth, we can put them together to achieve the final goal.

1 Introduction to Arduino

Arduino Uno Board

Arduino is essentially a small development board with a microcontroller that you can program. It interacts with the real world through LEDs, sensors, actuators, LCDs and so on. Take a moment to examine the Arduino Uno (see Figure 1.1). There are some important components, with which we need to be familiar:

- Atmel microcontroller
- USB programming interfaces
- Voltage regulator and power connections
- I/O pins
- Power and auxiliary pins

As Figure 1.1 shows, it has 14 digital input/output pins (of which 6 can be used as PWM outputs), which are located on the top row of Arduino Uno board. These tend to be used for reading on or off values, or switching actuators on or off. On the bottom left row, you can find two power pins: 3.3V or 5V supply. This is really useful, because many electrical components need 3.3V or 5V. Some pins labeled "GND" on the board are ground pins. On the bottom right row, there are 6 analog input pins labeled from A0 to A5. They can be used to measure things with a range of possible values, for example temperatures, moisture and so on.

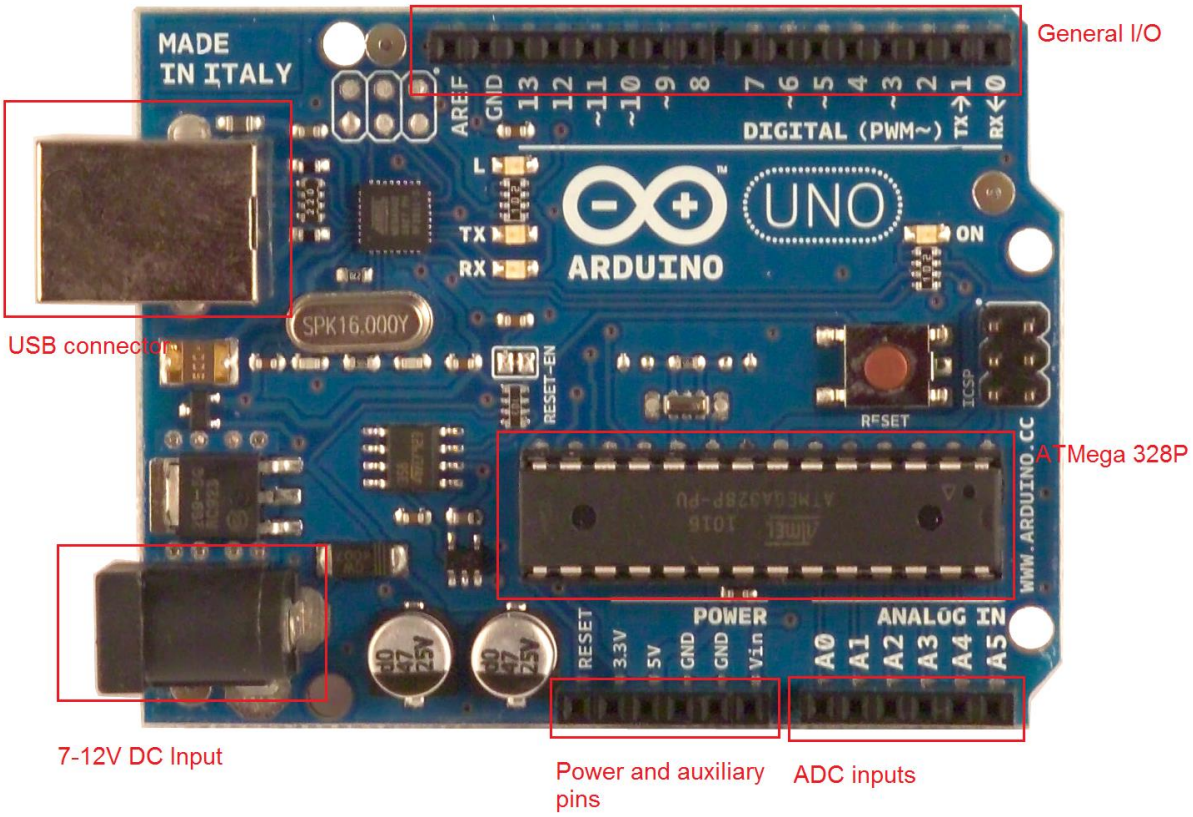


Figure 1.1: Arduino Uno Components

Arduino IDE

You write your C code in an Arduino IDE, which allows you to verify and compile your code, and then upload it to the Arduino Board through USB port. It is worth mentioning that the IDE also has a built-in serial monitor. With its help you can see what data the Arduino is sending back on your Desktop.

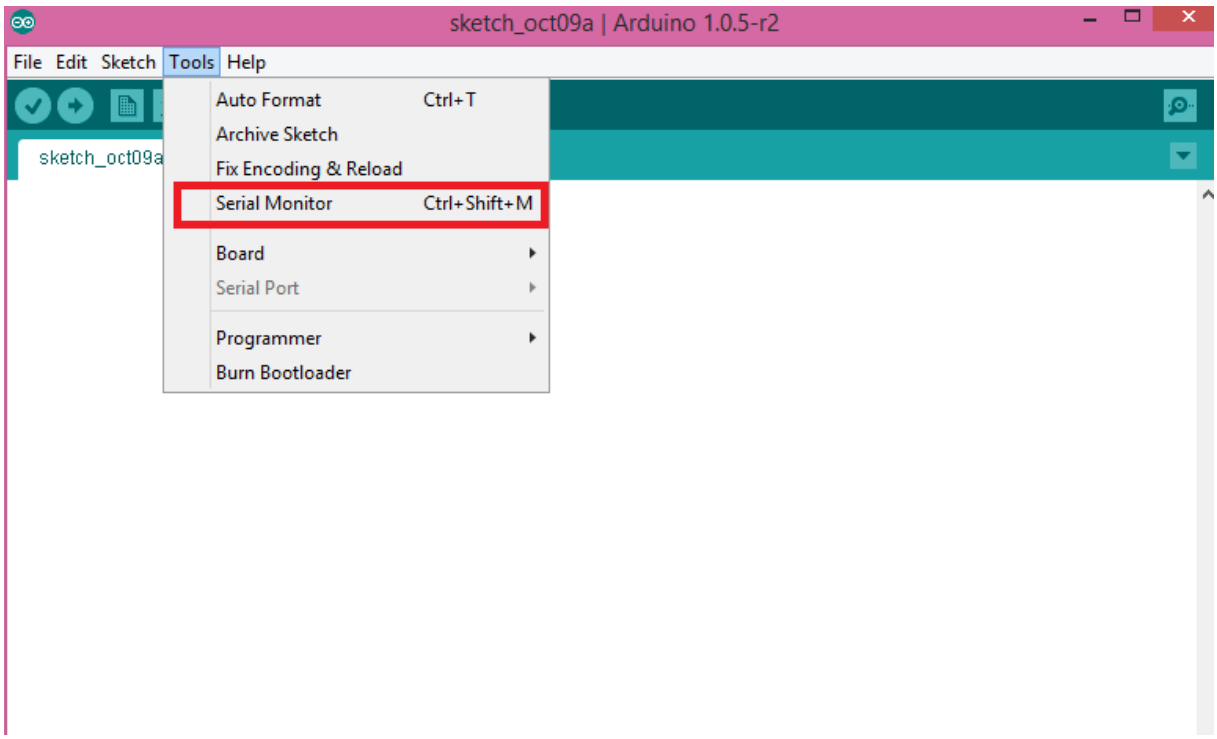


Figure 1.2: Arduino IDE

2 Controlling DC Motors

DC motors are widely used in numerous devices (e.g. DVD player) around your home. They rotate continuously when a DC voltage is applied across them. By adjusting the voltage you apply to them, you can change their rotation speed. Furthermore, by reversing the dire direction of the voltage applied to them, you can change direction of rotation. H-bridge is a useful module for controlling DC motors. The operation of an H-bridge can best be explained with the following diagram (see Figure 2.1).

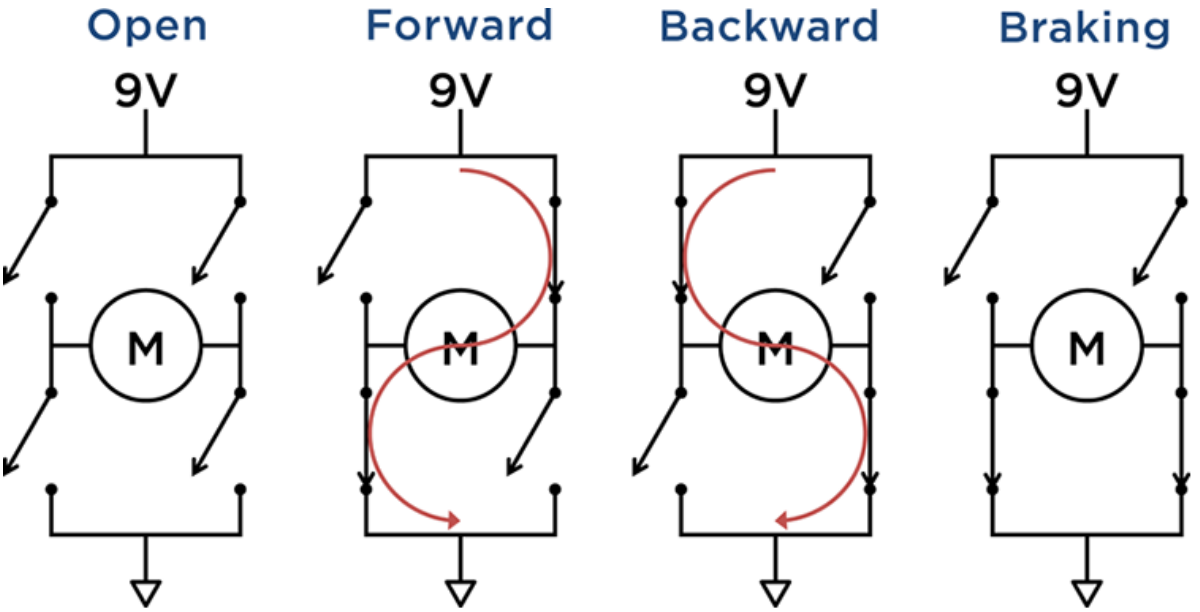


Figure 2.1 H-bridge Operation (Source: Exploring Arduino)

As you can see, the H-bridge has four states: open, forward, backward and breaking. In the open state, all the switches are open and the motor won't spin. In the forward state, two diagonally opposing switches are closed, causing a positive voltage through the motor. On the contrary, in the backward state this voltage is reversed, which leads to reverse operation of the motor. If the H-bridge is put in the braking state, all residual motion caused by momentum is ceased, and the motor stops.

For this robot car I use a Dual H-bridge motor diver shield, which is based on H-bridge driver chip L298N. As the Figure 2.2 displays, this motor driver can let us drive two independent DC motors, controlling the speed and the direction of each one.

On the hardware there are two motor connectors, which are labeled Motor A and Motor B. On this board it did not tell me which one is the positive or GND terminal, so I just hook it up the same way for both. In this picture, red is the positive end and black is the GND end for the motors.

In the middle of this board's right side, there are three pins: Vms, GND, and 5V. Vms and GND are used to connect external power source, which can be in the range from 6V to 35V. 5V pin can provide 5V voltage output, which can be used to power some other sensors, e.g. an ultrasonic sensor.

On the left side, there are eight pins:

- ENA (Input): 5V enable Motor A, 0V disable Motor A
- IN1 (Input): Motor A Direction 1 pin
- IN2 (Input): Motor A Direction 2 pin
- IN3 (Input): Motor B Direction 1 pin
- IN4 (Input): Motor B Direction 2 pin

- ENB (Input): 5V enable Motor B, 0V disable Motor B
- GND: Ground
- 5V (Input): power for this driver board

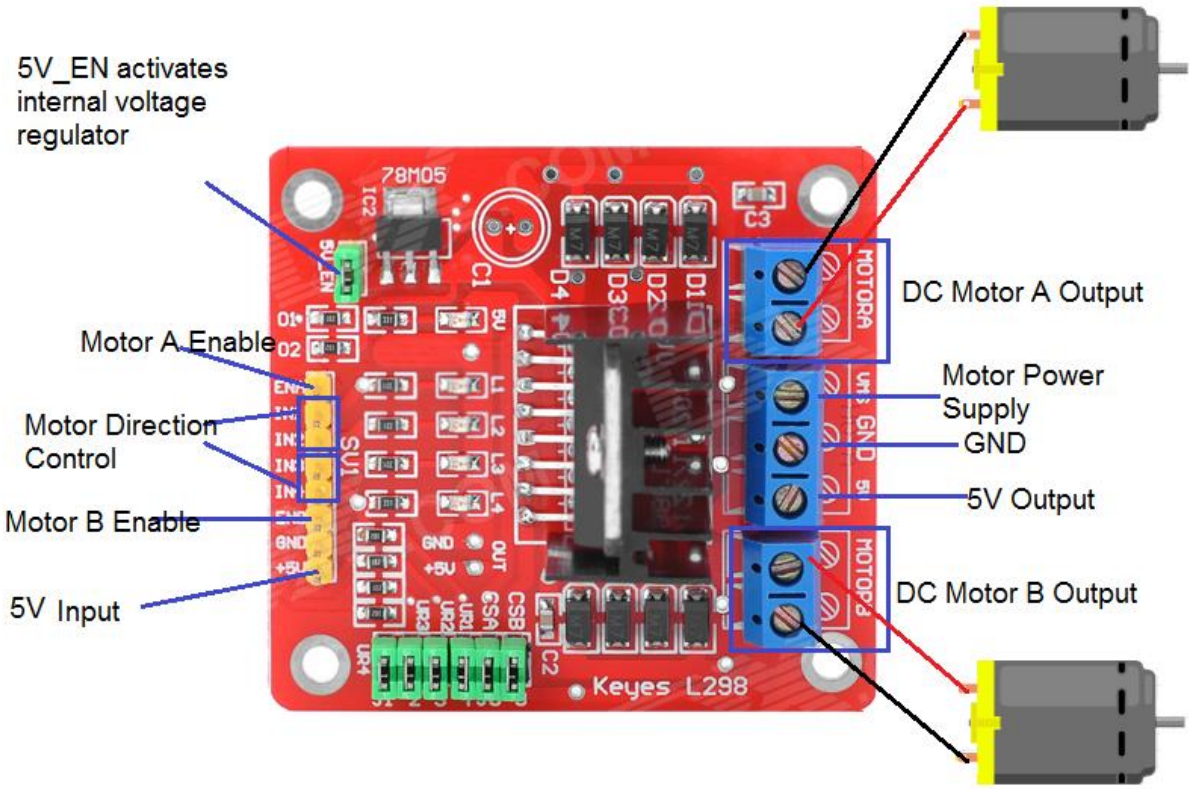


Figure 2.2: L298N Dual H-bridge DC Motor Driver Shield

The following table summarizes how to operate this shield:

Enable Motor	High – Enable	Low – Disable
Direction 1	IN1 – High	IN2 – Low
Direction 2	IN1 – Low	IN2 – High
Coasting	IN1 – Low	IN2 – Low
Break	IN1 – High	IN2 – High

Note: you may wondering what is Direction 1 and Direction 2. Direction 1 can be forward or backward, it depend on how you connect to your motor. It is better to get your own hands dirty. Do some experiments, which bring you some exciting experience.

Now I will show you a small demo. The following table describes how I bind Arduino with L298N.

Arduino Pin	**L298N**
5	ENA
6	ENB
2	IN1

3	IN2
4	IN3
7	IN4

The code is as follows:

```
/*##### Motor Shield (L298N) #####*/
/*
* Arduino Pin --> L298N
* 5 --> ENA
* 6 --> ENB
* 2 --> IN1
* 3 --> IN2
* 4 --> IN3
* 7 --> IN4
*/
const int ENA = 5;
const int ENB = 6;
/*
* IN1: HIGH; IN2: LOW --> Direction 1
* IN1: LOW; IN2: HIGH --> Direction 2
* IN3: HIGH; IN4: LOW --> Direction 1
* IN3: LOW; IN4: HIGH --> Direction2
*/
const int IN1 = 2;
const int IN2 = 3;
const int IN3 = 4;
const int IN4 = 7;

void setup()
{
  pinMode(ENA, OUTPUT);
  pinMode(ENB, OUTPUT);
  pinMode(IN1, OUTPUT);
  pinMode(IN2, OUTPUT);
  pinMode(IN3, OUTPUT);
  pinMode(IN4, OUTPUT);
  // Enable Motor A, Motor B: Constant Speed
  digitalWrite(ENA, HIGH);
  digitalWrite(ENB, HIGH);
  // Serial communication
  Serial.begin(9600);
}

void loop()
{
  Serial.println("Motor A & B: Direction 1");
  MotorAB_Direction1(1000);
  Serial.println("Motor A & B: Brake");
  MotorAB_Brake(1000);
  Serial.println("Motor A & B: Direction 2");
  MotorAB_Direction2(1000);
  Serial.println("Motor A & B: Brake");
  MotorAB_Brake(1000);
}

void MotorAB_Direction1(int milliseconds)
{
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, LOW);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, LOW);
  if (milliseconds > 0)
    delay(milliseconds);
}

void MotorAB_Direction2(int milliseconds)
{
```

```
  digitalWrite(IN1, LOW);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, LOW);
  digitalWrite(IN4, HIGH);
  if(milliseconds > 0)
    delay(milliseconds);
}

void MotorAB_Brake(int milliseconds)
{
  digitalWrite(IN1, HIGH);
  digitalWrite(IN2, HIGH);
  digitalWrite(IN3, HIGH);
  digitalWrite(IN4, HIGH);
  if(milliseconds > 0)
    delay(milliseconds);
}
```

Until now I have not explained how to control speed of DC motor. Normally this can be achieved through PWM (Pulse Width Modulation). What is PWM? The basic idea of PWM is to create a digital output wave of fixed frequency, but allow the microcontroller to vary its duty cycle. The system is designed in such a way that (HIGH+LOW) is constant, which means the frequency is fixed. The **duty cycle** is defined as the fraction of time the signal is high: duty cycle = HIGH / (HIGH + LOW). Some PWM signals with different duty cycles are shown in Figure 2.3.

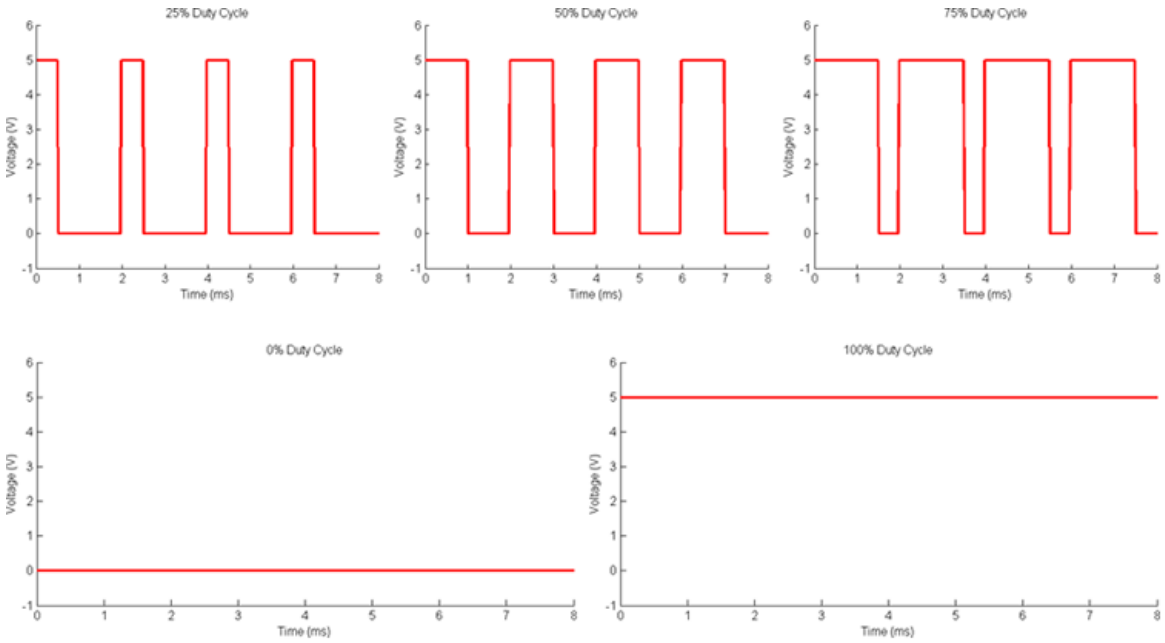


Figure 2.3 PWM signals with varying duty cycles (Source: Exploring Arduino)

On Arduino Uno board Pins 3, 5, 6, 9, 10 and 11 can be used as PWM pins. You can recognize them easily, because they are marked with ~ on the board. Furthermore, Arduino provide you a function named analogWrite() to generate PWM signals. The PWM output is an 8-bit value. In other words, you can write values from 0 to 255. Writing a value of 0 with analogWrite() indicates a square wave with a duty cycle of 0 percent (always low). On the contrary, writing a value of 255 indicates a square wave with duty cycle of 100 percent (always high).

I also made a small demo about how to control speed based on the above code. And I have upload these two examples to Github. If you have interest, please jump to this link: https://github.com/jiehou/ArduinoRobotCar.

3 Bluetooth Module (HC-06)

It is totally awesome that you are able to control your Arduino using Bluetooth. With help of HC-06 serial port Bluetooth module, this task can be easily achieved. It provides a wireless communication link between Arduino and any Bluetooth capable device. In this post the following HC-06 is used, which is shown in Figure 3.1.

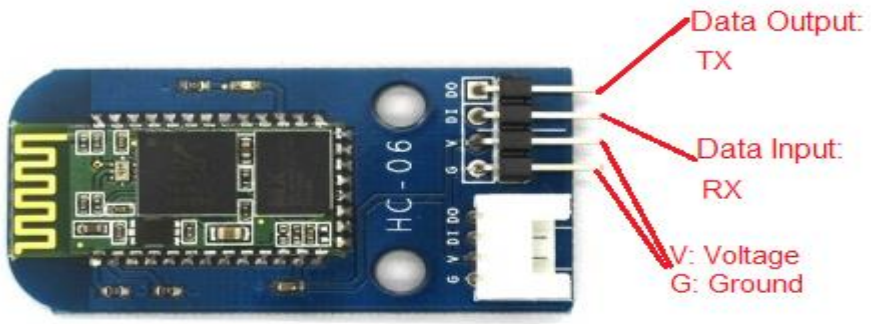


Figure 3.1 HC-06 Bluetooth Module

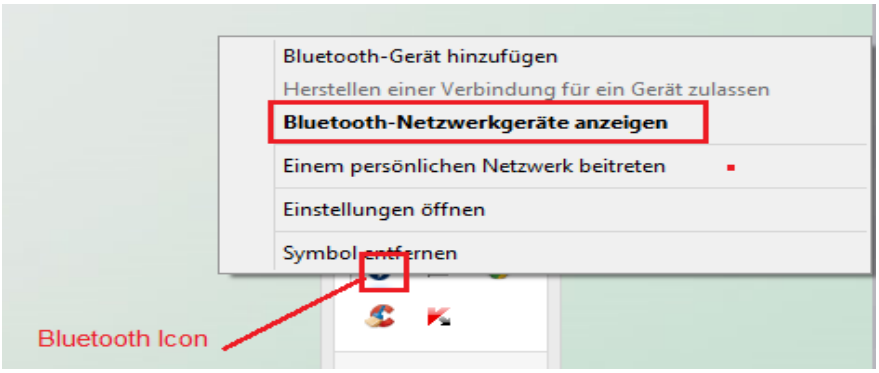
As Figure 3.1 shows, it has four pins: VCC, GND, TX and RX. It is not difficult to connect it to Arduino. The following table describes how to map these pins to Arduino Uno:

HC-06	**Arduino Uno Pin**
VCC	3.3V
GND	GND
TX (Data Output)	0 (RX)
RX (Data Input)	1 (TX)

Now we have finished connecting HC-06 to Arduino, it is time to test it. In this tutorial I will show you how to test HC-06 using your computer. It is assumed that your computer has Bluetooth feature. I used Tera Term (http://ttssh2.sourceforge.jp/index.html.en) on my laptop as the serial terminal.

After Tera Term was successfully installed on your computer, following steps help you set up connection between your computer and your HC-06 module.

Step 1: Pairing HC-06 with Computer



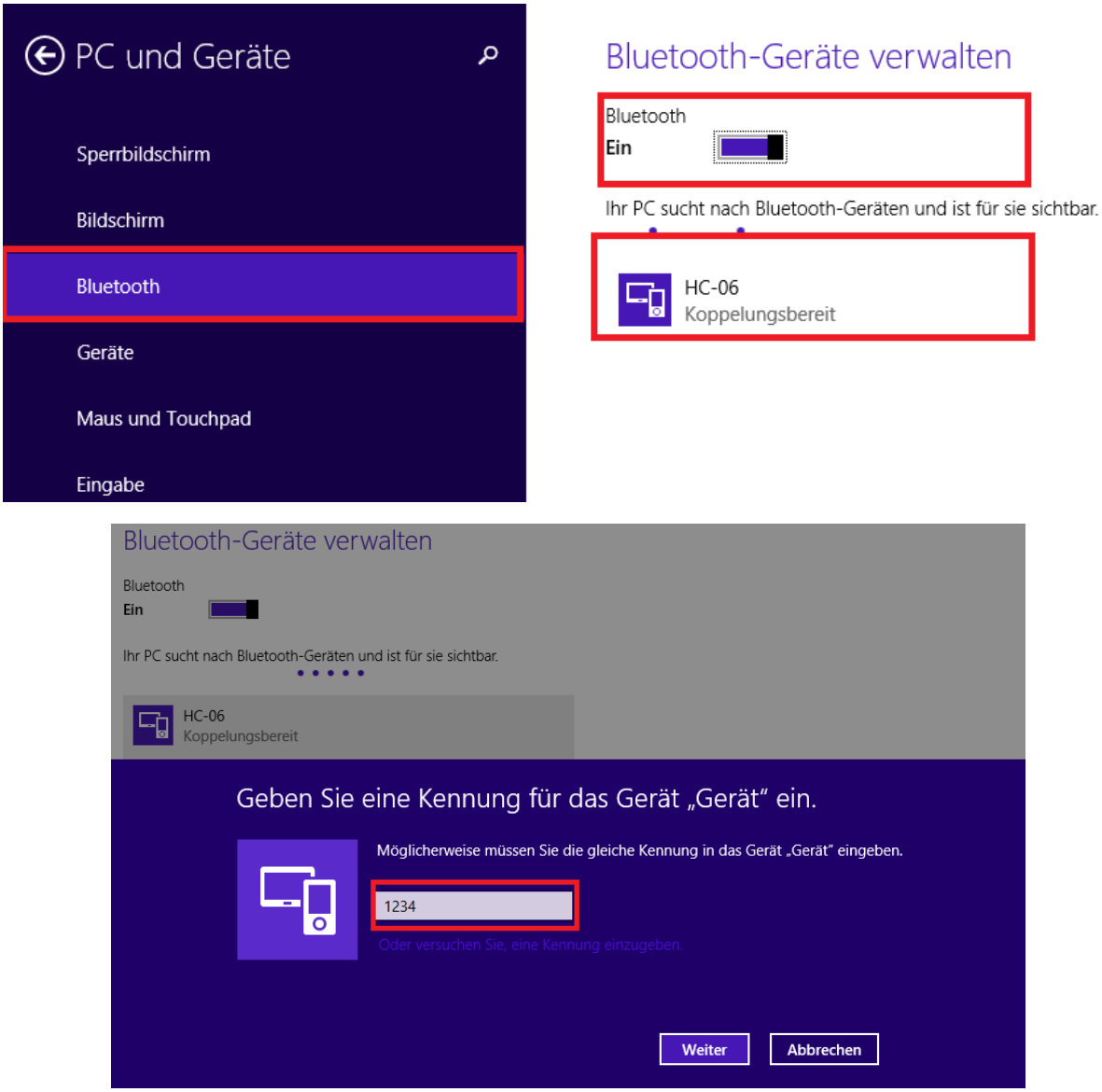
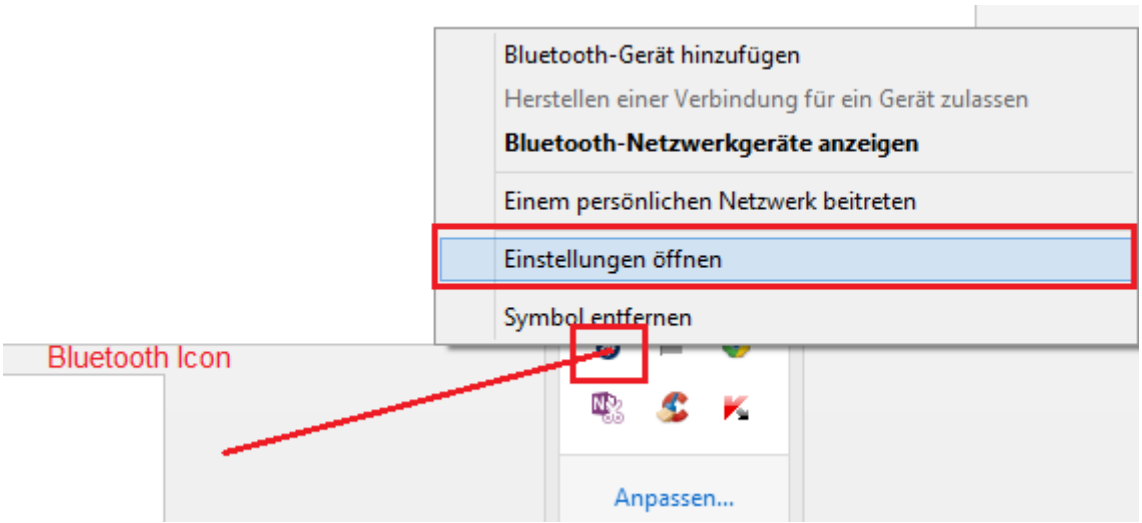


Figure 3.2 Paring HC-06 with Computer

Please note: the default passcode for pairing HC-06 is **1234**.

Step 2: Getting COM Ports for paired HC-06 Module



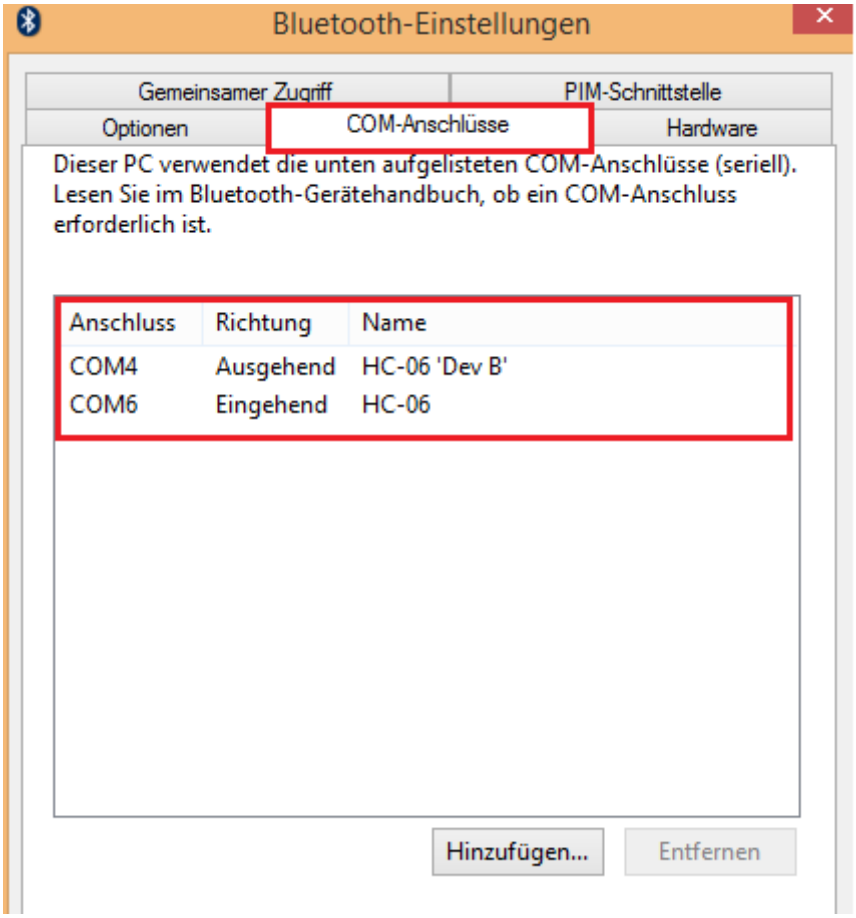


Figure 3.3 COM Ports

If you pair HC-06 with your computer successfully, under tab (COM Ports) the detailed information about it will be displayed. In our case, we need a COM Port, whose direction is outgoing.

Step 3: Using Tera Term

Open Tera Term, then enable Serial option. After that choose your COM Port in the drop down list and finally click OK. It will build connection between your computer and your HC-06 Bluetooth Module.

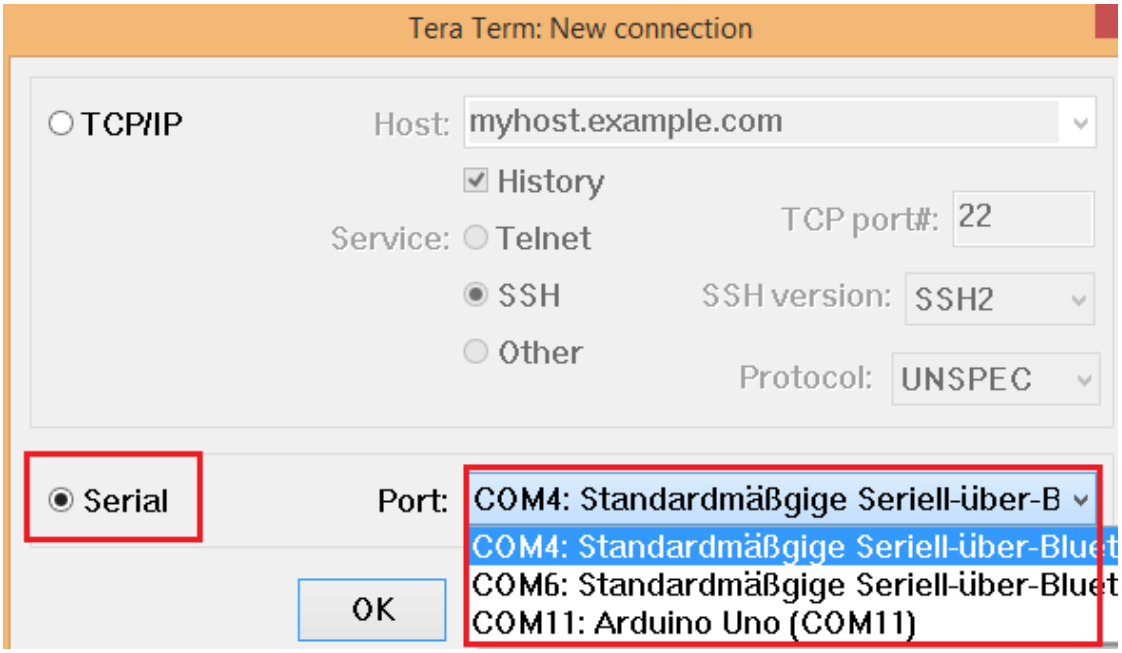


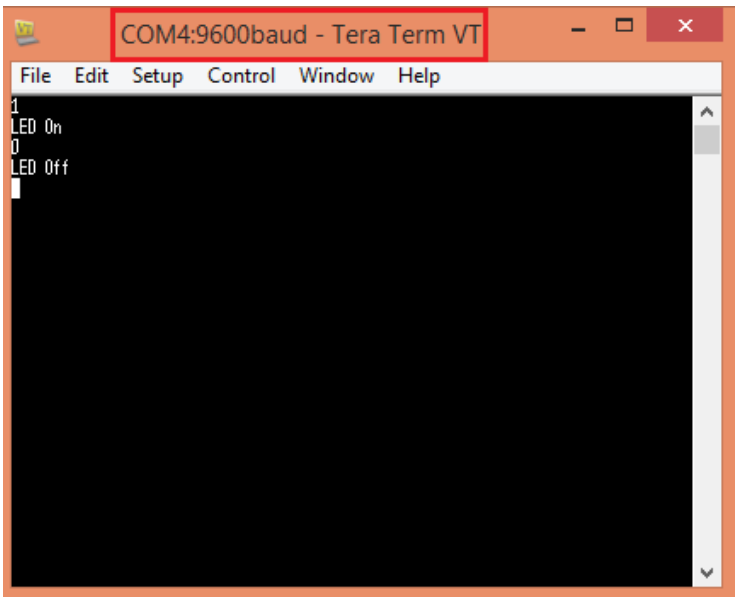
Figure 3.4 Using Tera Term

The following demo code lets you send ‘1’ to switch a LED on and ‘0’ to switch a LED off.

```
/**
* LED Code
* Send 1: Switch LED On
* Send 0: Switch LED Off
*/
// Pin 12: LED
const int LED = 12;
char mIncoming;

void setup()
{
  pinMode(LED, OUTPUT);
  // Set baud rate 9600
  Serial.begin(9600);
}

void loop()
{
  if (Serial.available() > 0)
  {
    mIncoming = Serial.read();
    Serial.println(mIncoming);
    if (mIncoming == '1')
    {
      digitalWrite(LED, HIGH);
      Serial.println("LED On");
    }
    else if (mIncoming == '0')
    {
      digitalWrite(LED, LOW);
      Serial.println("LED Off");
    }
  }
  delay(100);
}
```



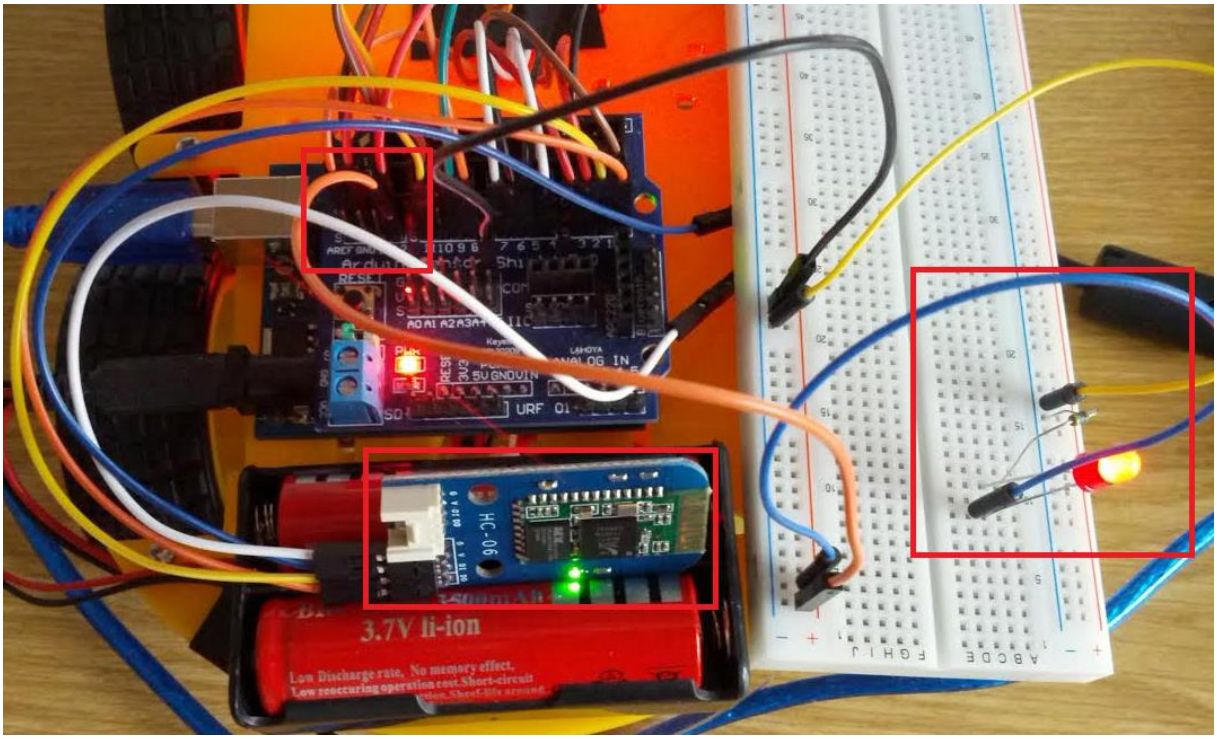


Figure 3.5 HC-06 Testing Demo

4 Using Android to Control Arduino

In the previous post, I have already talked about how to use Tera Term to communication with Bluetooth Module HC-06. Nowadays almost each smart handy has embedded Bluetooth feature. In this post, I will discuss about how to write your own Android APP to control your Arduino project using HC-06 Bluetooth module.

The Android platform includes support for the Bluetooth network stack, which allows a device running Android to wirelessly exchange data with other Bluetooth devices. This application framework allows developers to access to the Bluetooth functionality through the Android Bluetooth APIs. Using these APIs, an Android application is capable of managing the local Bluetooth adapter, scanning for nearby Bluetooth devices, querying the local Bluetooth adapter for paired Bluetooth devices, transferring data between Bluetooth devices and more.

In this post I will show you my own summary about how to start with programming Android Bluetooth API. Actually the documentation from Android is absolutely a good start point.

Step 1: Setting Bluetooth Permissions

In order to work with Bluetooth in your application, you must declare the BLUETOOTH permission in the AndroidManifest.xml file.

<uses-permission android:name="android.permission.BLUETOOTH" />

This permission enables you to perform any Bluetooth communication, such as requesting a connection, accepting a connection and transferring data.

If you want your app to initiate device discovery or manipulate Bluetooth settings, the BLUETOOTH_ADMIN permission is needed. Many applications needs this permission only for the ability to discover nearby Bluetooth devices.

<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />

Step 2: Setting up Bluetooth

BluetoothAdapter is the entry point for all Bluetooth interaction. It is used as a singleton to access the Bluetooth radio on the Android device. To get the BluetoothAdapter, call the static *getDefaultAdapter()* method. If *getDefaultAdapter()* returns null, it means that the device does not support Bluetooth and your application should gracefully end here.

```
BluetoothAdapter bluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
if (bluetoothAdapter == null) {
        // Device does not support Bluetooth;
}
```

If the target device supports Bluetooth, next we need to check whether Bluetooth is enabled or not. If *isEnabled()* method returns false, then the Bluetooth is disabled. In order to enable it, call *startActivityForResult()* with the **ACTION_REQUEST_ENABLE** action intent.

```
if (!bluetoothAdapter.isEnabled()) {
        startActivityForResult(new Intent(
                        BluetoothAdapter.ACTION_REQUEST_ENABLE), BLUETOOTH_ENABLE);
}
```

Step 3: Finding remote devices

Through the BluetoothAdapter, you can find remote devices either through device discovery or by query the list of paired devices. The following section describes how to find devices that have been already bonded, or discover new devices using device discovery.

Querying paired devices

To get detailed information of paired devices, we need to call *getBondedDevices*(). It will return a set of BluetoothDevice, which are already bonded to your device. The following code snippet was used in my program.

```
// Show already paired devices in the paired list
Set<BluetoothDevice> pairedDevices = bluetoothAdapter
                .getBondedDevices();
if (pairedDevices.size() > 0) {
        findViewById(R.id.tvPairedDevices).setVisibility(View.VISIBLE);
        for (BluetoothDevice device : pairedDevices) {
                // Signal strength isn't available for paired devices
                devPairedList.add(new Device(device.getName(), device
                                .getAddress(), (short) 0));
        }
}
```

Discovering new devices

A BroadcastReceiver for the ACTION_FOUND Intent must be registered in order to receiver information about each device discovered. This Intent carries the extra fields EXTRA_DEVICE containing a BluetoothDevice object. For example, I use the following code to handle the broadcast when devices are discovered.

```
// Add found device to the devices list
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
        @Override
        public void onReceive(Context context, Intent intent) {
                String action = intent.getAction();
                if (BluetoothDevice.ACTION_FOUND.equals(action)) {
                        // Found device in range
                        BluetoothDevice device = intent
                                        .getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
                        Device foundDevice = new Device(device.getName(),
                                        device.getAddress(), intent.getShortExtra(
                                                        BluetoothDevice.EXTRA_RSSI,
Short.MIN_VALUE));
                        // If it is not a paired device then add it to the list
                        if (device.getBondState() != BluetoothDevice.BOND_BONDED) {
                                devAvailableList.add(foundDevice);
                                // Signal list content change
                                devAvailableListAdapter.notifyDataSetChanged();
                                // Make the available devices title visible
                                findViewById(R.id.tvAvailableDevices).setVisibility(
                                                View.VISIBLE);
                        }
                } else if (BluetoothAdapter.ACTION_DISCOVERY_FINISHED
                                .equals(action)) {
                        // When finished (timeout) remove the progress indicator and
                        // enable search button
                        setProgressBarIndeterminateVisibility(false);
                        btnFindDevices.setText(R.string.searchDevices);
                        btnFindDevices.setEnabled(true);
                }
        }
};
```

Step 4: Building and managing a connection with remote device

Until now we have already been able to find a nearby Bluetooth device. Next we can use the **BluetoothDevice** to acquire a **BluetoothSocket** and initiate a connection with the discovered remote device. When you have successfully connected these two devices, each one will have a connected BluetoothSocket. Then they can communicate with each other. In this post, I just send some commands from Android to HC-06 module.

Here is the basic procedure

1) Get a **BluetoothSocket** by calling bluetoothDevice.**createRfcommSocketToServiceRecord**(UUID)
2) Initiate the connection by calling bluetoothSocket.**connect**()
3) Get InputStream and OutputStream that handle transmissions through the socket, via bluetoothSocket.**getInputStream()** and bluetoothSocket.**getOutputStream()**
4) Read and write data to streams with **read(byte[])** and **write(byte[])**

Because *connect()*, *read(byte[])* and *write(byte[])* are blocking calls, it is important that a dedicated thread should be used.

If you want to learn more about Android Bluetooth, the following links are recommended:

1) Official document: http://developer.android.com/guide/topics/connectivity/bluetooth.html
2) Sample Project from Android official website (Bluetooth Chat): https://android.googlesource.com/platform/development/+/eclair-passion-release/samples/BluetoothChat

From my point of view, the above mentioned project helps me understand Android Bluetooth API a lot. It is really a good starting point to learn programming Android Bluetooth.

The following figures are screenshots of my little Android APP, which is called Arduino Car Console. It contains two activities. The first activity (as shown in Figure 4.1) is in charge of finding and listing paired or new devices. When user clicks on a selected device, then a connection to this device will be built up.

If the connection is successfully set up, the second activity will be used to send commands to HC-06 module. From figure 4.2 we know that, this simple console can send four commands: Forward, Backward, Left and Right. When user clicks on Forward button, a series of 'f' will be sent out. The sending interval is 50 ms.

Button	**Command**
Forward	f
Backward	b
Left	l
Right	r

I have tested this APP on my handy, which is based on Android 4.04. It works very well. Theoretically it can be used on any Handy, which is based on Android 4.0 or above. Thanks for downloading and using this APP. It would very nice, if you could give me some feedback.

Source code of this APP can be found:
https://github.com/jiehou/ArduinoRobotCar/tree/master/ArduinoCarConsole
APK file of this APP can be found:
https://github.com/jiehou/ArduinoRobotCar

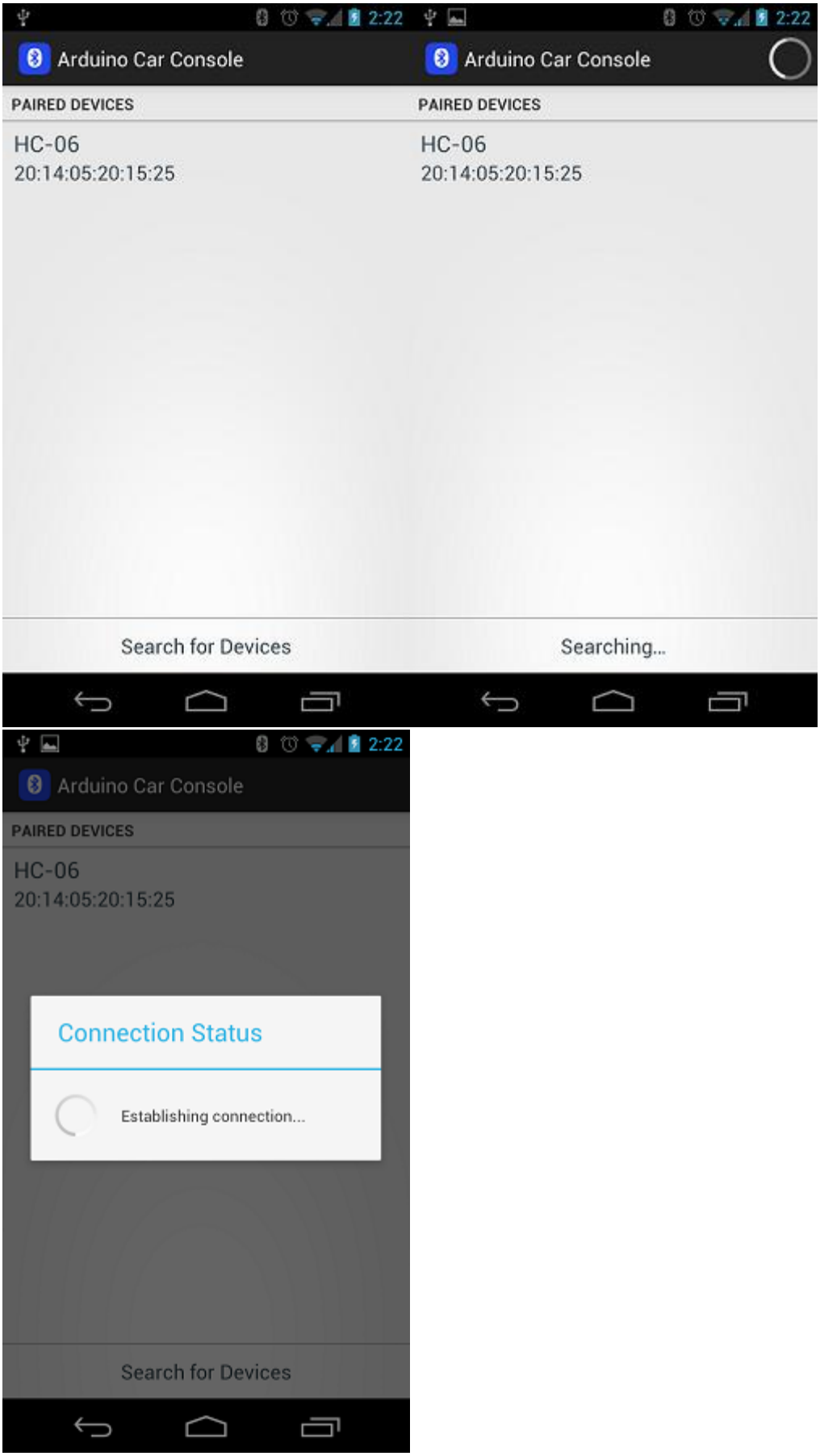


Figure 4.1 Activity for devices and connection

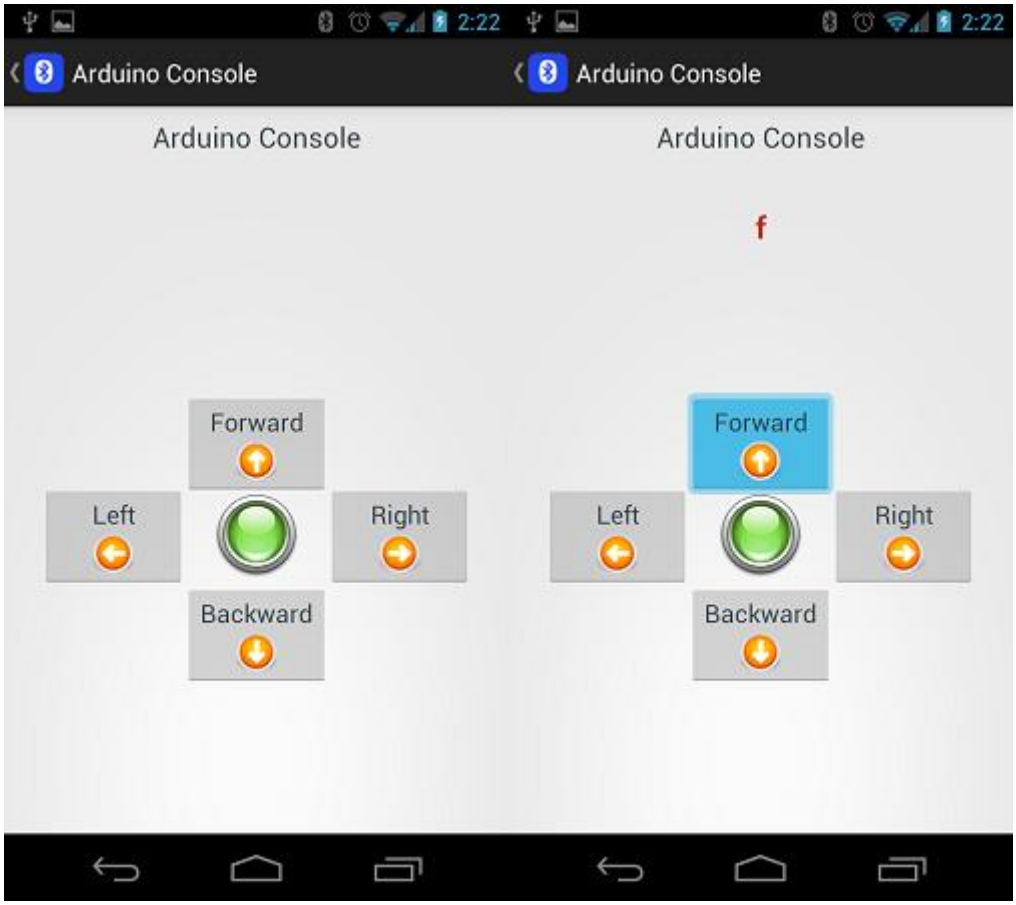


Figure 4.2 Activity for sending commands

5 Final Goal – Arduino Bluetooth Car

In previous post, I have already explained how to send commands from an Android handy to an Arduino using HC-06 Bluetooth module. In this post, the complete Arduino Car will be assembled, which will be described step by step. Because HC-06 Bluetooth module was already introduced, here it will not be discussed any more.

Step 1: L298N

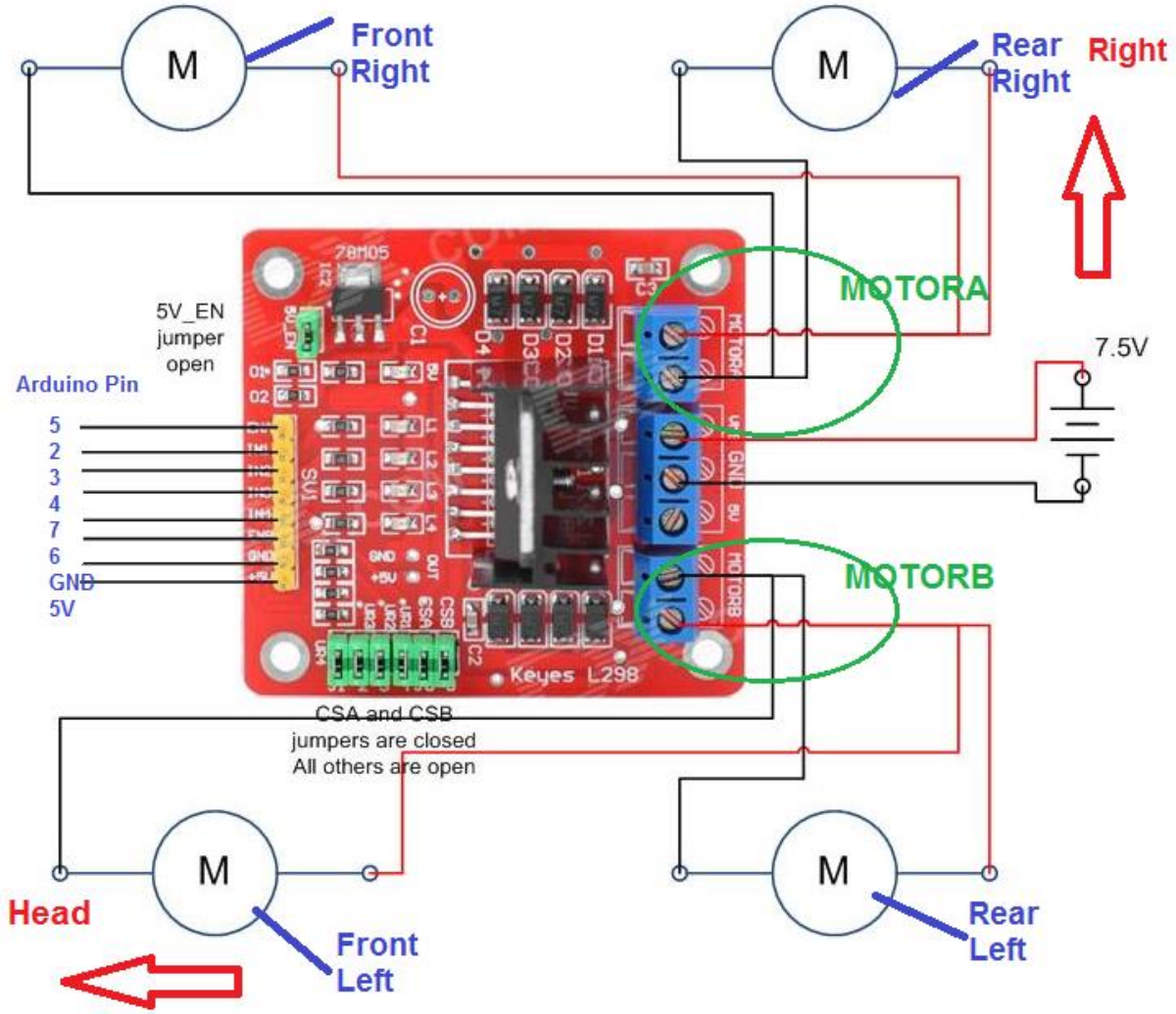


Figure 5.1 L298N

Arduino Uno Pin	L298N (from top to under)
5	ENA
2	IN1
3	IN2
4	IN3
7	IN4
6	ENB
GND	GND
5V	+5V (Input)

Furthermore, we provide L298N an external 7.5V voltage. MOTORA is in charge of two right side motors, correspondingly two left side motors are connected to MOTORB. Figure 5.2 describes how to control directions of this robot car.

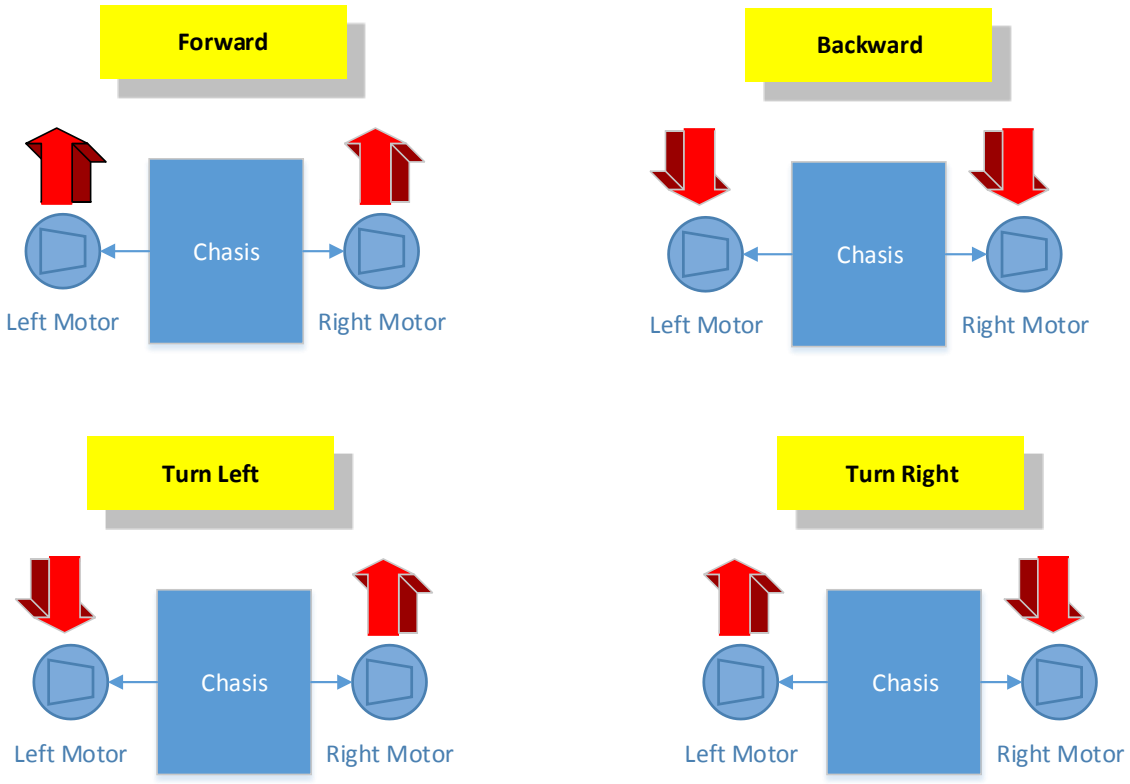


Figure 5.2 Control Directions

Step 2: A sweeping distance sensor

This robot kit contains an ultrasonic distance sensor (HC-SR04), a small servo motor and a plastic holder, which can be used as a base to hold a servo. Using these three components we can build a sweeping distance sensor, which is shown in figure 5.3.

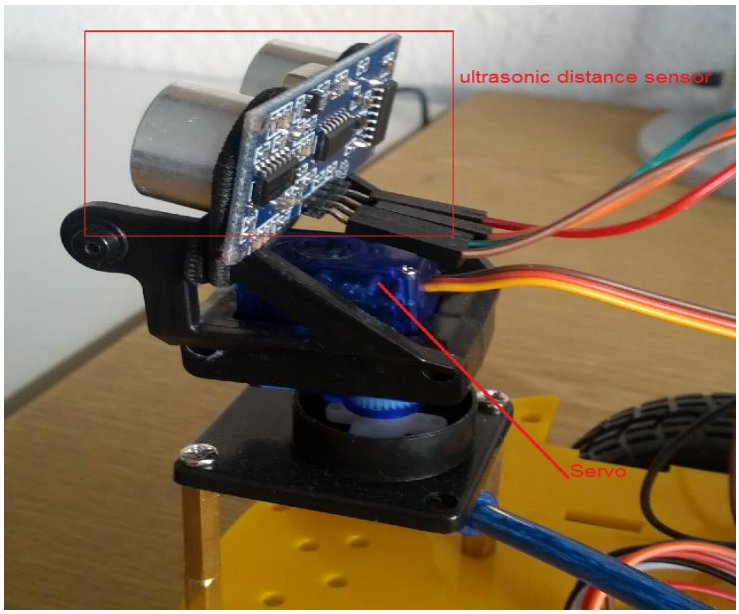


Figure 5.3 Sweeping distance sensor

Unlike DC motors, servo motors have three pins: power (usually red), ground (usually brown or black), and signal (usually orange).

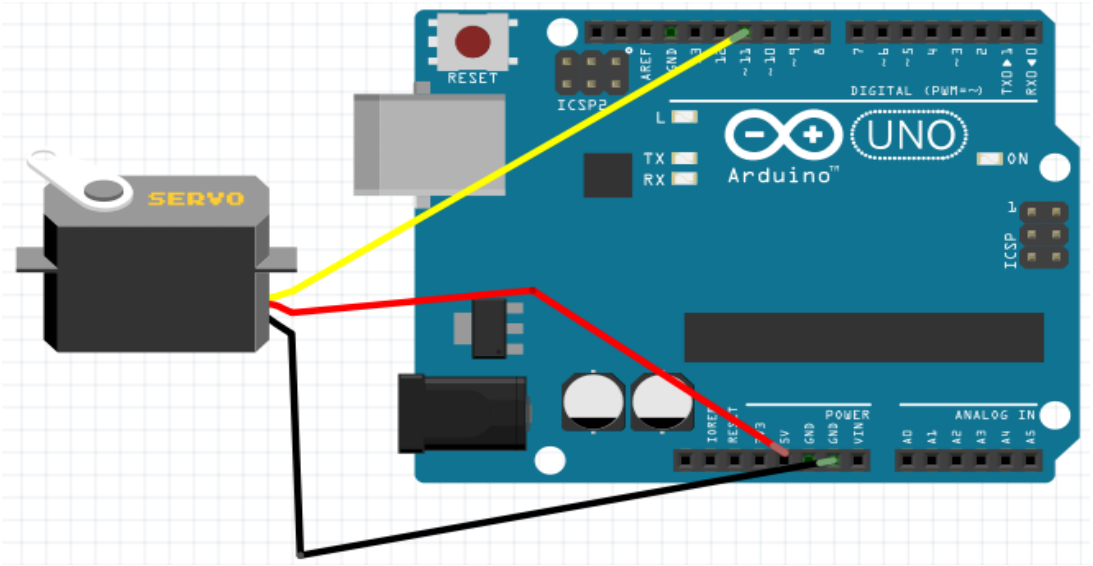


Figure 5.4 Wiring servo

As figure 5.4 shows, the signal pin of servo is connected to Arduino pin 11. The Arduino IDE contains a built-in library, which makes controlling servos a breeze. All you have to do is attaching a servo object to a particular pin and give it an angle to rotate to, which is displayed in the following code snippet.

```
#include <Servo.h>
//Servo on Pin 11
const int SERVO = 11;
Servo myServo;
void setup() {
  myServo.attach(SERVO);
}
void loop() {
  myServo.write(90);
  delay(1000);
}
```

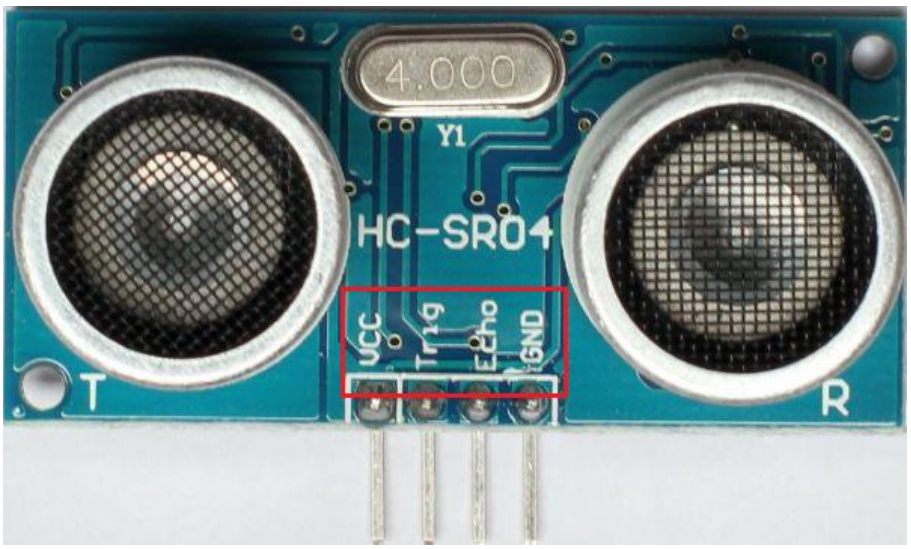


Figure 5.5 HC-SR04 ultrasonic sensor

The HC-SR04 ultrasonic module uses sonar to determine distance to an object, just like dolphins do. It includes ultrasonic transmitter, receiver and control circuit. It works as follows:

1) Using I/O trigger for at least 10us high level signal
2) This module automatically sends eight 40kHZ signal and detect whether there is a returned pulse signal
3) If the signal returns back, time of high output I/O duration is the time from sending ultrasonic to returing
4) Distance = (high level time * velocity of sound) / 2

The following table describes how this module is connected to Arduino Uno in my case:

HC-SR04 Ultrasonic Sensor	**Arduino Uno Pin**
VCC	5V
Trig	8
Echo	9
GND	GND

To sum up, the following table list the detailed pin assignments for all modules, which are used on this robot car.

Arduino Uno Pin	**Description**	**L298N**	**HC-SR04 Ultrasonic**	**HC-06 Bluetooth**	**Servo for sweeping**
VCC	VCC	+5V	VCC	VCC	Red (Power)
GND	GND	GND	GND	GND	Brown (GND)
0 (RXD)	Digital Pin 0/Rx			TXD	
1 (TXD)	Digital Pin 1/Tx			RXD	
2	Digital Pin 2	IN1			
3	Digital Pin 3/PWM	IN2			
4	Digital Pin 4	IN3			
5	Digital Pin 5/PWM	ENA			
6	Digital Pin 6/PWM	ENB			
7	Digital Pin 7	IN4			
8	Digital Pin 8		Trig		
9	Digital Pin 9/PWM		Echo		
10	Digital Pin 10/PWM				
11	Digital Pin 11/PWM				Orange (Signal)
12	Digital Pin 12				
13	Digital Pin 13				

Following figures draw the flowchart of the program that is run by this robot car. The complete source code for this Android APP and Arduino robot car, and the document about this project can be found in github. If you have interest, please visit the following link: https://github.com/jiehou/ArduinoRobotCar.

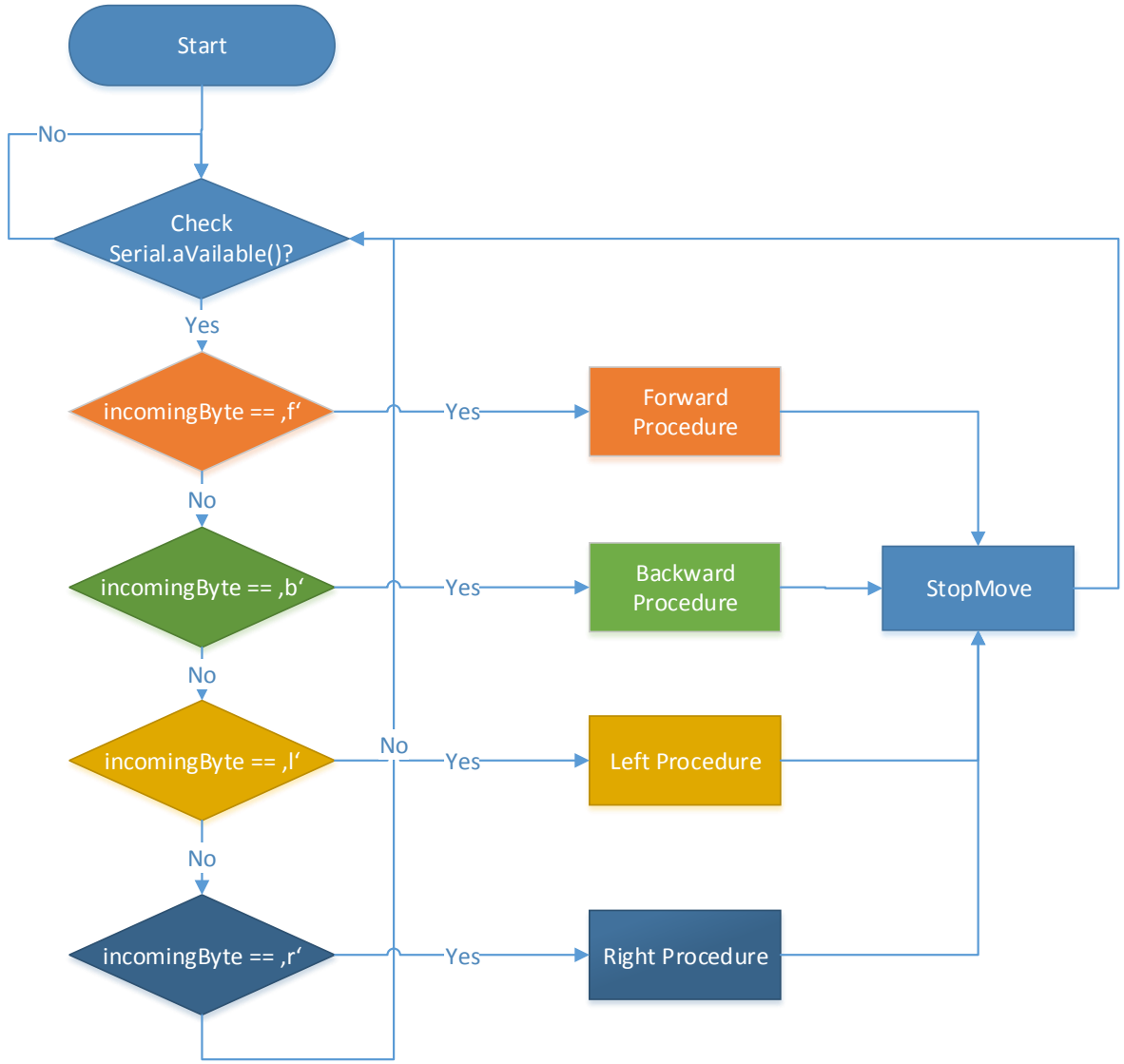
Under the above link there are several folders:

1 ArduinoCarConsole: An Android APP to send commands from Android to Arduino.

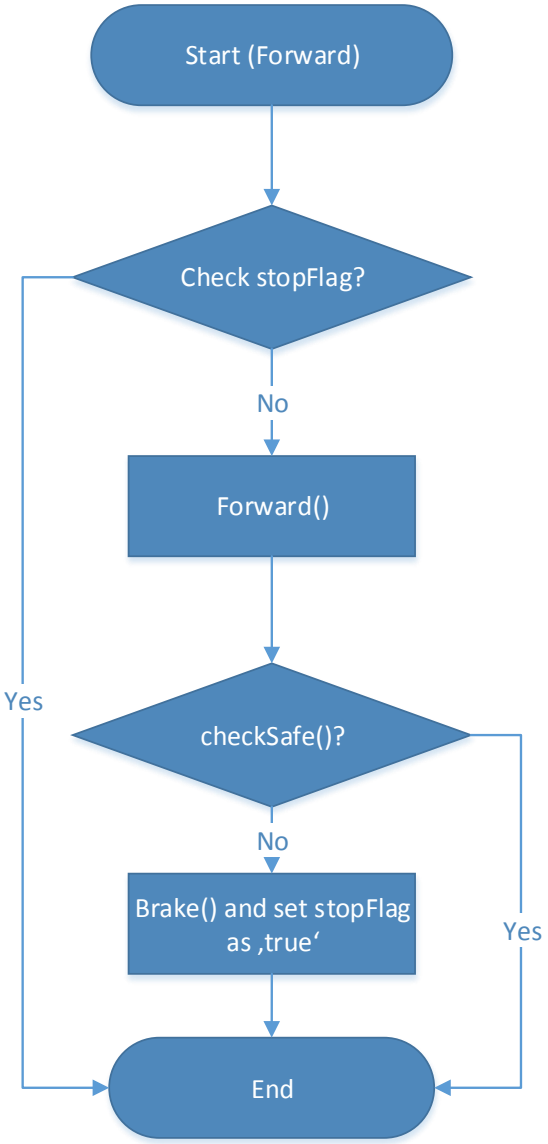
2 BluetoothCar: The complete source code of this robot car, it is written in C.

3 L298N: Source code of Demos about how to use L298N motor shield.

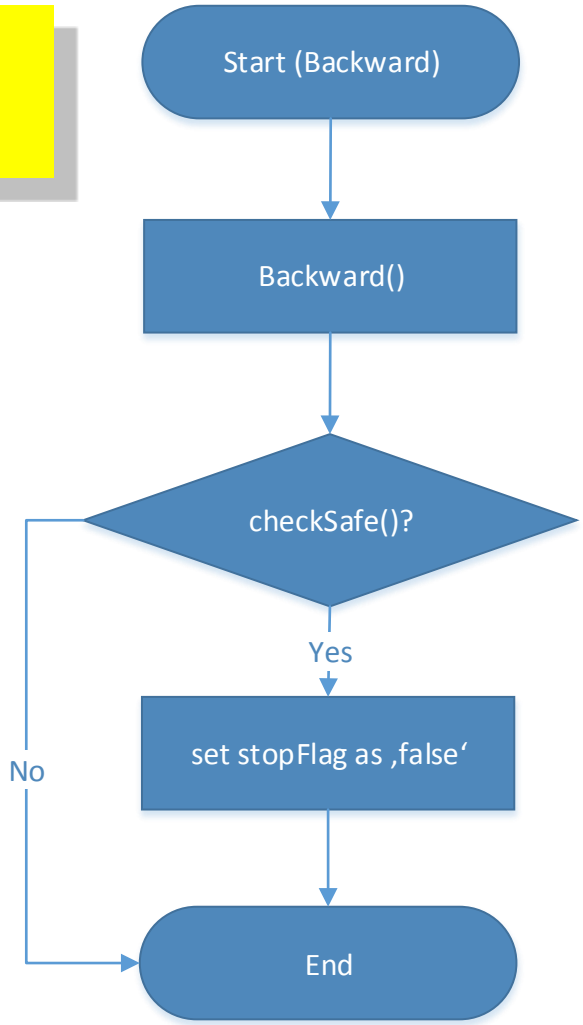
4 Documents: The complete document about this project and some program flowcharts.



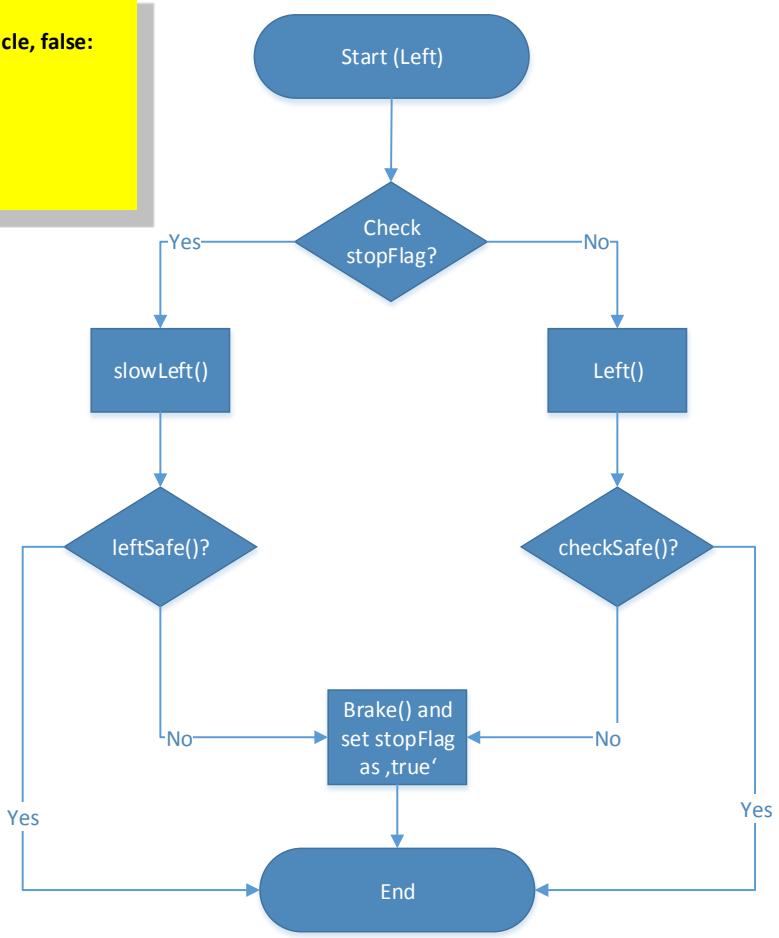
Forward Procedure
1) stopFlag: global variable, true: Obstacle,
2) false: Safe
3) Safe Distance (Foward): 20cm
4) Safe Distance (Left and Right): 10cm



Backward Procedure
1) stopFlag: global variable, true: Obstacle, false: Safe



Left Procedure
1) stopFlag: global variable, true: Obstacle, false: Safe
2) Left(): PWM 255
3) slowLeft(): PWM 128
4) Safe Distance (Foward): 20cm
5) Safe Distance (Left and Right): 10cm



Right Procedure
1) stopFlag: global variable, true: Obstacle, false: Safe
2) Right(): PWM 255
3) slowRight(): PWM 128
4) Safe Distance (Foward): 20cm
5) Safe Distance (Left and Right): 10cm