

Servomotor Control with Arduino Integrated Development Environment

Application Notes

Bingyang Wu

Mar 27, 2015



Introduction

Arduino is a tool for making computers that can sense and control more of the physical world than your desktop computer. It's an open-source physical computing platform based on a simple microcontroller board, and a development environment for writing software for the board.

Arduino can be used to develop interactive objects, taking inputs from a variety of switches or sensors, and controlling a variety of lights, motors, and other physical outputs. Arduino projects can be stand-alone, or they can communicate with software running on your computer (e.g. Flash, Processing, MaxMSP.) The boards can be assembled by hand or purchased preassembled.

The open-source Arduino Software (IDE) makes it easy to write code and upload it to the board. It runs on Windows, Mac OS X, and Linux. The environment is written in Java and based on Processing and other open-source software. This software can be used with any Arduino board.

Objectives

The Application Notes is a user guide introducing the Arduino Integrated Development Environment (IDE) and its application in controlling the servomotor. The servomotor is to be used for lifting and lowering the arm in the gate security system designed by the team. The following parts will take you through the setup and functionality of different components of the Arduino IDE. The programming code regarding controlling of the servomotor will also be explained in detail.

Arduino IDE Setup

When the Arduino IDE is opened up, the following window will pop up. The window may look slightly different for different Operating Systems (OS); they are generally identical in terms of layout and functionality. For general purposes, this tutorial uses Mac OS X as example.

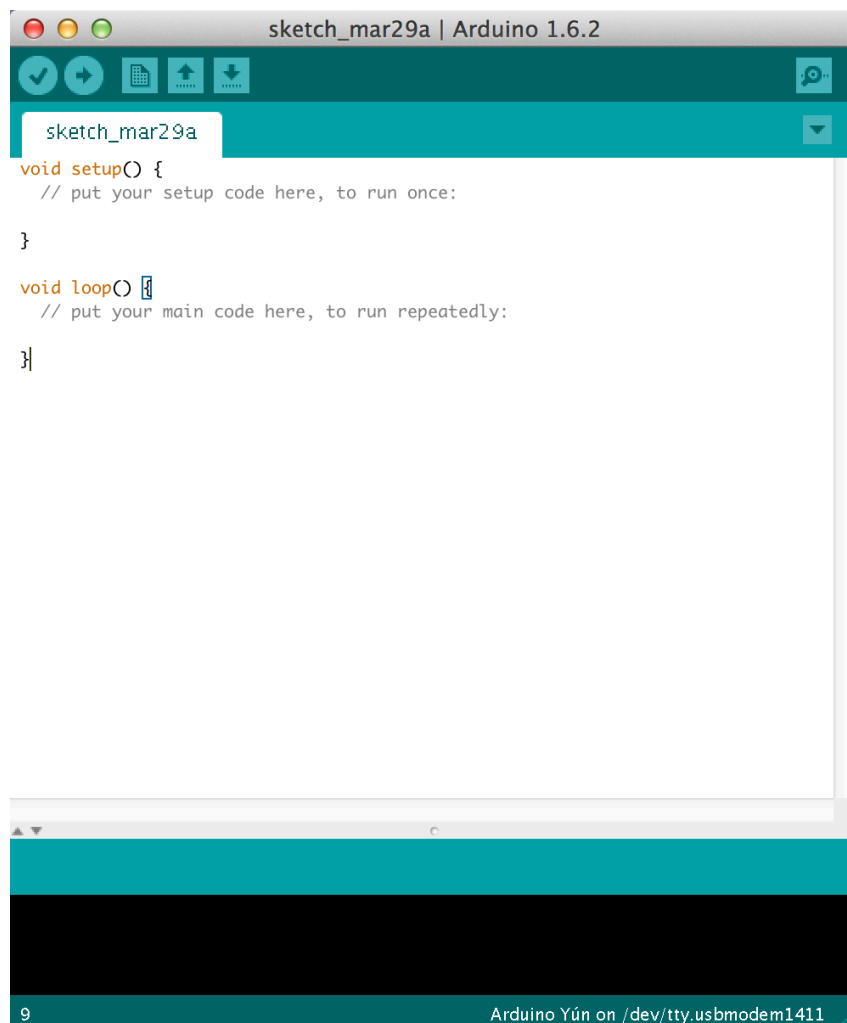


Figure 1. Arduino IDE Interface

The interface is fairly clear and straightforward, which is divided into three sections, with a toolbar at the top; the code editing area in the middle; and the notification/status area at the bottom. For the latest version of Arduino IDE (Arduino IDE 1.6.2), the toolbar consists of six buttons as shown in Table 1 below. The tab with a file name is located between the toolbar and the code editing area.

Button Icons	Functionalities
	Verify - Check the errors from the code
	Upload - Write the current code into the microcontroller
	New - Open a new blank window for coding
	Open - Open an existing sketch
	Save - Save current program
	Serial Monitor - Send and receive data to and from Arduino

Table 1. Toolbar Buttons and Their Corresponding Functionalities

Among the introduced buttons and their corresponding functionalities above, serial monitor is one of the most useful components, which is of a great importance as testing the program. Serial monitor displays the serial data received from the output of the microcontroller. One can also input data to the microcontroller via serial monitor. In other words, serial monitor provides a way for the users to interact with the microcontroller by immediate data exchange.

Clicking on the serial monitor icon opens up a window as shown below. One can select the baud rate at which the data is sent to or received from the Arduino. Baud rate stands for the rate of change per second at which the status (or bit data) is sent to or received from the Arduino microcontroller. The default baud rate is 9600, meaning the user is sending 1200 characters per second via the serial connector (USB cable in the case of the team) for sending a character record (9600 bit/8 bit per character = 1200 characters).

A textbox is located at the top of the serial monitor window; one can input data by clicking the “send” button. The blank area below the textbox is where the serial data output is displayed.

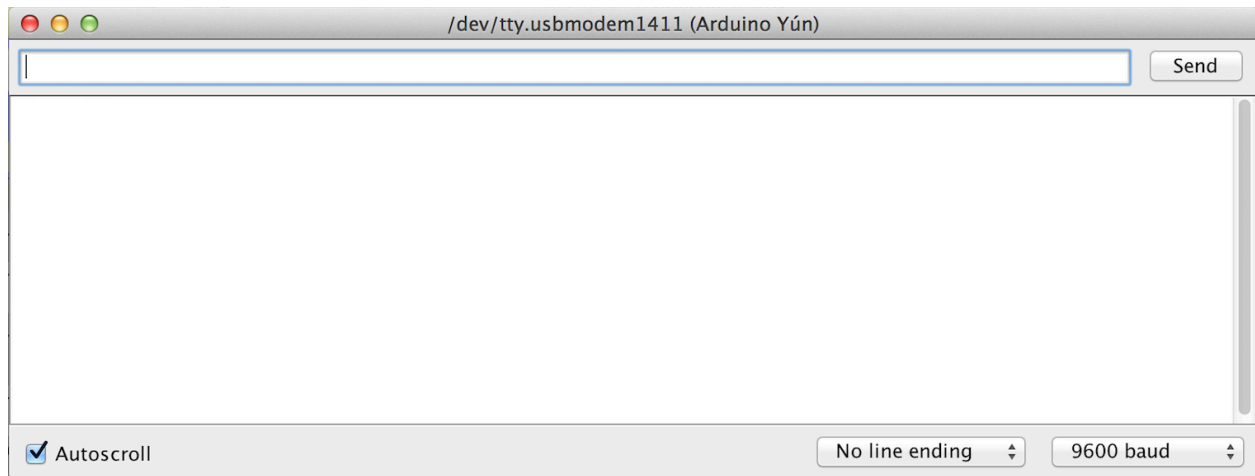


Figure 2. Serial Monitor Window

Servomotor Setup

Once you are confident with manipulating the Arduino IDE, you are well prepared to move on to the next stage and work on the projects in a variety of scopes. For the gate security system project for ArcelorMittal, an RFID reader and a servomotor are connected to the microcontroller (Arduino Yun for this project). Once an RFID tag/card is put near to the RFID reader, a character string corresponding to the specific tag/card is displayed in the serial monitor window. If the character string matches one of the preset strings in the database, the motor is triggered and it lifts the arm allowing vehicles to pass through. Then the motor runs reversely to put the arm to its original position.

The Arduino IDE implements C/C++ programming language. Following are the code segments that program the servomotor. The logic and functionality of the essential parts of the code will be explained in paragraphs following each of the code segments.

```

#include <Servo.h>
Servo servo;
int pinServoStart = 9;
int pinServoStop = 7;
int servoLockedRotation = 150;
int servoUnlockedRotation = 30;
int unlockTime = 3500;
int pinLed = 13;

```

The top line of the code is `#include` directive followed by a header file. The effect of “`#include <Servo.h>`” is the same as if one has typed the entire contents of the “`Servo.h`” file into the file at the point where `#include` line appears. The “`Servo.h`” file is supplied as part of all C compiler packages. It contains information about input and output functions that are needed for programming the servomotor.

The third and fourth lines assign the corresponding microcontroller pins on which the servomotor will attach. Please be noticed that only one servomotor is used in the design, instead two pins are assigned. The reason for this will be discussed in later paragraphs in a more detailed manner. For now just keep in mind that the servomotor is only attached on pin 9. The next two lines determine the position of the servomotor for lifting and lowering the arm, respectively. “`unlockTime`” determines the duration of the lifting and lowering process for the arm. The last line assigns pin 13 to the LED, which works as an indication when the access is granted.

```
servo.attach(pinServoStop);
```

This line initializes the servomotor at rest position because as mentioned above that the servomotor is attached only on pin 9, not pin 7.

```
pinMode(pinLed, OUTPUT);
digitalWrite(pinLed, LOW);
```

These two lines define the pin for the LED as an output pin, and initialize the LED as off.

```

val = Serial.read();
    if(val == 'G')
    {
        digitalWrite(pinLed, HIGH);           // Access Granted
        servo.attach(pinServoStart);
        servo.write(servoUnlockedRotation);
    }

```

```

        delay(unlockTime);
        servo.attach(pinServoStop);
        delay(3500);
        servo.attach(pinServoStart);
        servo.write(servoLockedRotation);
        delay(unlockTime);
        servo.attach(pinServoStop);
        digitalWrite(pinLed, LOW);
    }
    else if(val == 'D') // Access Denied
    {
        delay(1000); // Wait a moment to scan the next card
    }

```

The code segment is for the control of the servomotor. “`val = Serial.read();`” at the first line reads whatever inputted by the user via serial monitor. A reading of “G” represents access granted, similarly a “D” for access denied. This is only for the purpose of testing the proper working of the servomotor.

If the access is granted, “`digitalWrite(pinLed, HIGH);`” makes the indication LED on. The next two lines attach the servomotor on pin 9 and run the motor in the direction of lifting the arm. The duration of the lifting process is determined by the following delay code. Then the “`servo.attach(pinServoStop);`” stops the servomotor by switching the effective pin from 9 to 7, which detaches the servomotor that is originally attached on pin 9. The following delay holds the servomotor at its lifted position for a specified time period. After that, the servomotor is reattached to pin 9 and “`servo.write(servoLockedRotation);`” indicates that it is to be run in the direction of lowering the arm. The following delay assures same duration for the lowering process as it is for the lifting process. Once the servomotor goes back to its original position, it stops running again for being detached from pin 9.

For the access denied situation, there will be only a delay code in place that gives a short break to the reader before the next card can be scanned.

If you find that attaching the servomotor to an unused pin is a waste, an alternative way to achieving the same result is to replace the lines of “`servo.attach(pinServoStop);`” with “`servo.detach();`”. It detaches the servomotor from the pin it was attached and stops the motor immediately.

The character strings corresponding to the cards read are as shown below in Fig.3.

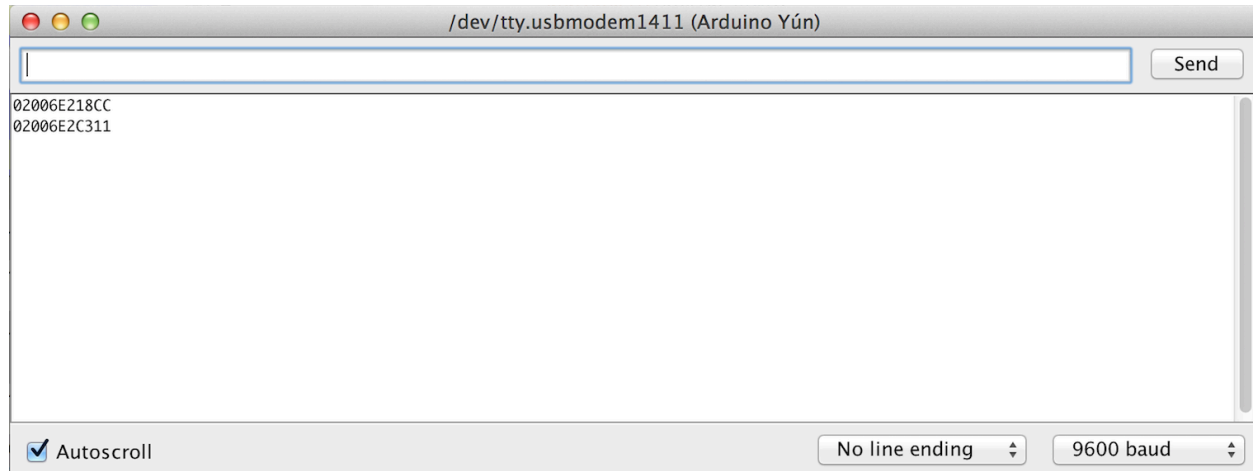


Figure 3. Character Strings of Cards

Conclusion

The Arduino IDE is a powerful development tool that allows the users to take advantage of its user-friendly interface and achieve a complex design goal in an easy manner at both the prototype and manufacturing stage. For the motor part of the project, the Arduino IDE did an exceptionally good job in terms of conciseness and efficiency. There are, of course, many ways in which the project can be improved in some aspects if time is not that much of a constraint. An obvious example could be adding a sensor that replaces, or possibly works with, the delay implemented that holds the arm in its lifting position to allow passing of the vehicles. In this case, the arm will not go down unless the clear path under it is confirmed by the system. Thus the design would be much more applicable in real life regarding its higher safety standard and more efficient implementation, especially for the always busy traffic flows in front of the gate. Also, another sensor could be added to distinguish passengers from vehicles to prevent people from taking advantage of the ease of getting a new ticket if the parking lot is charging by hours. Therefore, the point is that the Arduino IDE along with the Arduino microcontroller provides the users with unlimited ways to bringing to reality whatever is in their mind, and there are always various methods available for a single task. It is always a good idea to try different blueprints to find the best one for the targeted product.

Reference

<http://arduino.cc/>

Stephen Prata, C primer Plus Sixth Edition