# MOTOGURU

Motor Control Project

Done By:

1) Mahmoud Hamdy Mohamed
2) Fatima Gomaa Mohamed
3) Basma AbdelHakim Ahmed
4) Ali Samir Mattar
5) Mo'men Ahmed Bahaa

Mahmoud Hamdy
[Email address]

## Overview:

The main purpose of our project is to control a DC motor using a desktop application. This control includes controlling the following:

- Motor state (On/Off)
- Motor Rotation Direction (Clockwise/Anti-clockwise)
- Motor Speed (on a scale from 0 to full speed)

This control is achieved by combining a Java GUI with an Arduino board.

Commands are entered by user through the GUI, which then sends data to Arduino board through "Serial Communication (USB port)".

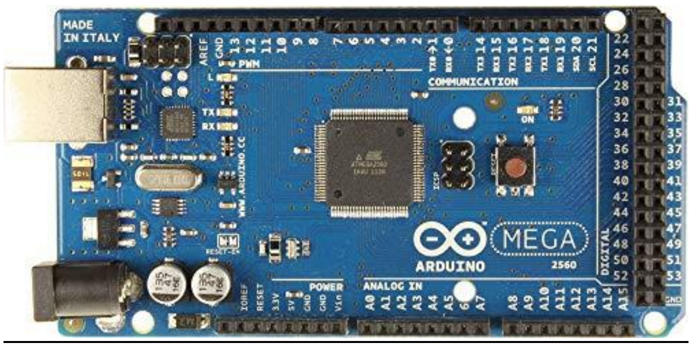
## Hardware:

The main factors that were taken into consideration during choosing hardware:

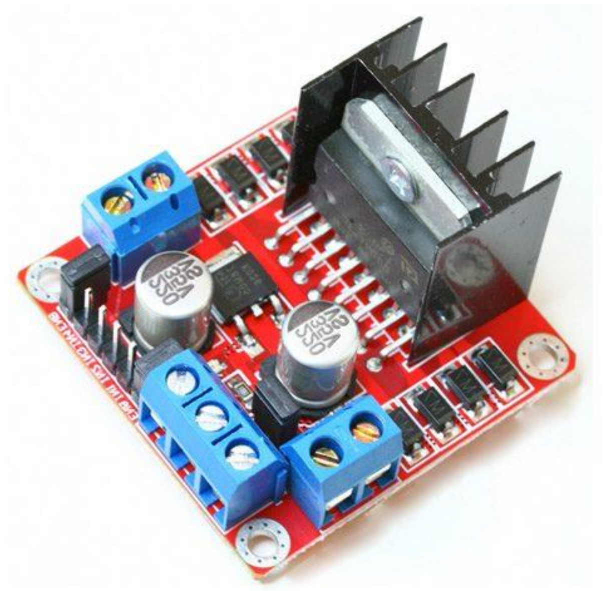
- Best performance possible
- Lowest possible cost
- Easiest form of implementation (easy setup for user)

To achieve these goals, we choose the following hardware:

- **Microcontroller:** Arduino Mega



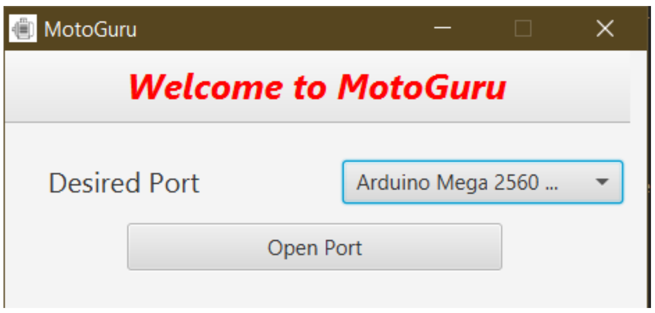
- **Motor Driver:** L298



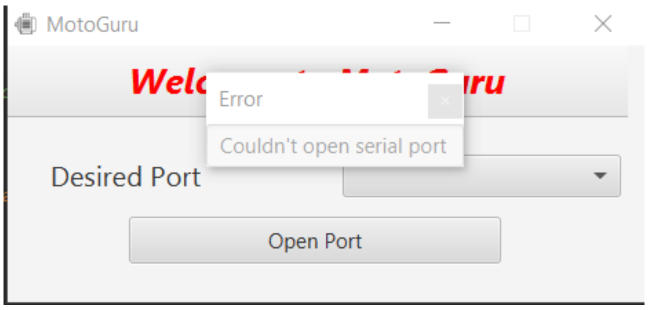
## User Manual:

### Screen #1: Port Choosing:

This is the first screen that the user will see upon launching the program, it will let him choose the desired port from the dropdown list and then click connect to open the port.



If the user doesn't choose a proper port or something wrong happens, it will display this error to the user:

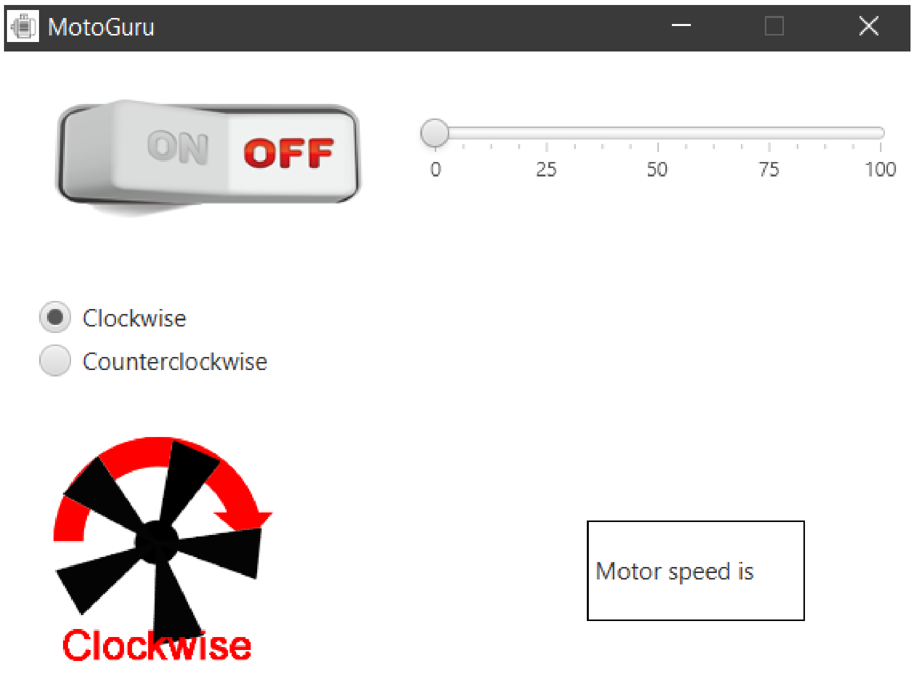


Error

Couldn't open serial port

## Screen #2: Main Program

Once a proper port has been opened, the main program screen will open which lets user control motor state, speed and rotation direction.



## GUI application:

The GUI was made using scene builder which creates the interface by generating an FXML file and tying it to a controller class.

The GUI application has two scenes

- Port chooser scene
- DC motor controller scene

### The GUI consists of four files:

- FXMLportchooserController.java

  This file has one public class that implements Initializable. This class handles GUI for choosing port. It has a dropdown list which populates the ports and confirmation button.

  In this class we have two objects, a dropdown list which populates the ports and a confirmation button.

  The class has one method which is connectPort method, Here the program will try to open port according to the value chosen from the list. If the port is successfully opened the DC motor controller scene is displayed, else an error message will be displayed for the user.

- FXMLportchooser.fxml

  This file includes the design for the first scene.

- FXMLDocumentController.java

  This file has one public class that implements Initializable. This class handles what will appear on GUI after choosing port, i.e. on/off, speed control and direction control for the motor.

  This class includes five methods:

  - handleButtonAction( )
    It fires when clicking on the on/off button. It toggles the button graphic between on and off, and starts a new thread of SerialWriter class.

  - handleClockwiseRadioButtonAction( )
    It fires when clicking on the clockwise rotation button. It changes the graphics that indicates the motor rotation and sends data over serialWriter class.

  - handleAntiClockwiseRadioButtonAction( )
    It fires when clicking on the anticlockwise rotation button. It changes the graphics that indicates the motor rotation and sends data over serialWriter class.

  - onSliderChanged( )
    It fires upon changing motor speed through slider values. It sends value to motor and displays it on the GUI.

  - initialize( )
    This method runs at the initialization of the GUI. It loads the graphics that will be used in the file like the pictures that show status of motor.

- FXMLDocument.fxml

  This file includes the design for the second scene.

## Project interface:

The interface is implemented using jSerialComm library.

### What is jSerialComm?

jSerialComm is a Java library designed to provide a platform-independent way to access standard serial ports without requiring external libraries, native code, or any other tools. It is meant as an alternative to RxTx and the (deprecated) Java Communications API, with increased ease-of-use, an enhanced support for timeouts, and the ability to open multiple ports simultaneously.

Some of the features of this library include:

- Platform-independent library deployment (automatically uses correct native library based on current architecture).
- Very lightweight and efficient implementation.
- Enumerates all available serial ports on a machine.
- Returns both a system port description and a friendly device description.
- User-specifiable port descriptors including symbolic links.
- Configurable ports according to baud rate, data bits, stop bits, and parity Configurable port timeouts (blocking and non-blocking) for both reading and writing.
- Configurable flow control parameters for the serial port (CTS, RTS/CTS, DSR, DTR/DSR, XOn/XOff).
- Ability to read and write raw data bytes directly to the serial port.
- Ability to read and write byte streams via Java's InputStream and Output Stream interface.
- Event-based reading and writing via callbacks.
- Callback notification when:
    - New data is available for reading.
    - All data has been successfully written .
    - A complete fixed-length data packet has arrived .
    - A delimited string-based message has been received Additionally.

**public class SerialWriter implements Runnable{}**

The idea behind implementing runnable is that the software is listening always in the background to the changes that may happen in the GUI and able to transfer these changes to the Hardware.

The data is sent in a form of String that holds all the changing variables needed to control the motor and the sent string is build accumulatively and sent in any time the flags rise.

The class uses the two types of output streams

- low level streams -> OutputStream.
- high level streams -> DataOutputStream.

Java uses the stream, reader, and writer classes for streamed data.

Stream classes deal with general data input and output, whereas the reader and writer classes deal specifically with Unicode and Unicode Transformation Format (UTF) string input and output.

Data received from or sent to general I/O devices consists of bytes. However, Java can support higher-level I/O by piecing together bytes to represent other types of data, such as integers, characters, or strings.

**public class SerialComm {}**

This class has only one member which is the serialPort. It is defined static so that no exception will be thrown. The member method **connect** is responsible for the actual physical connection that is initialized in the software, it connects to the specified port name and communicates in the predefined BaudRate.