# SOFTWARE DESIGN DESCRIPTION

for

# Automated Taping Process La Machina GUI

Release 1.0

Version 1.0 approved

Team 22:
Jon Sobanski, Gary Basile, Tashi McSweeny
Faculty Advisor: Vito Moreno
Industry Advisor: Donald Scott

Prepared by Adam Fowles

April 12, 2018

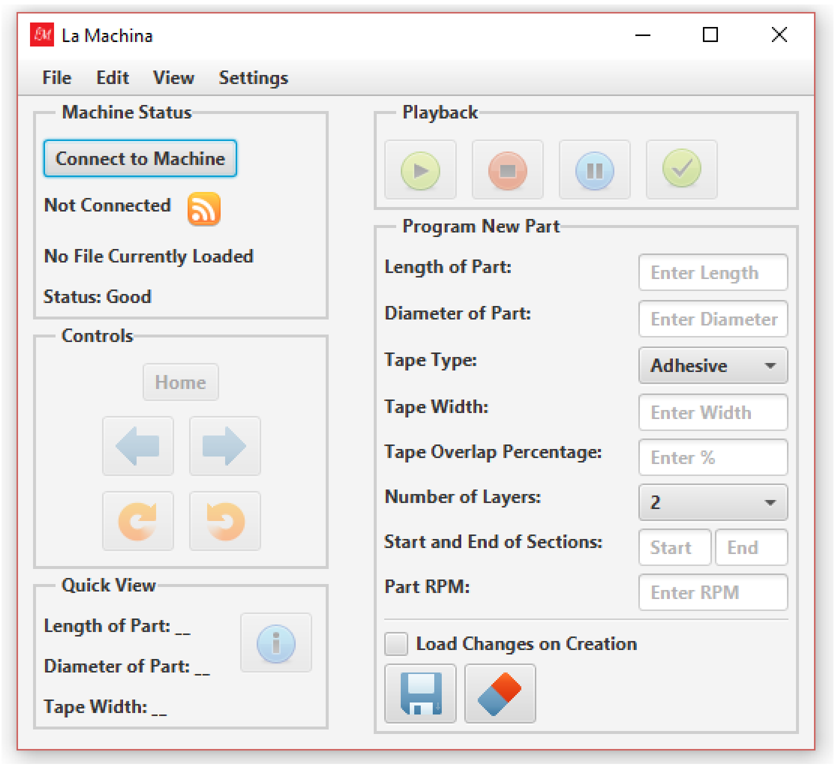
# Contents

## 1 Start Up and Usage

From command line run:

```
java -jar senior-design-team-22-gui-alpha.jar
```

java version 1.8 is required to run the application, javafx is being used. There are no specific command line arguments passed to this application.



Not Connected

No File Currently Loaded

Status: Good

Length of Part: Enter Length

Diameter of Part: Enter Diameter

Tape Type: Adhesive

Tape Width: Enter Width

Tape Overlap Percentage: Enter %

Number of Layers: 2

Start and End of Sections: Start End

Part RPM: Enter RPM

Length of Part: __

Diameter of Part: __

Tape Width: __

Figure 1: Main Window

### 1.1 Controls and Use Case

A basic walk through of the normal usage case is as follows: once the application has been run, using the above command, a quick splash screen will appear (see Figure 5.) and then the main window will appear as shown in Figure 1. The first thing that should be done is the user should attempt to connection to the machine using the "Connect to Machine" button inside of the Machine Status area. This will connect the graphical application to the arduino through a USB serial connection.

Once the connection has been established, the connected field will read "Connected" (instead of "Not Connected") and the icon will change color to blue. The status label at the button indicates the status of the machine. A "Good" status means everything is operational. Additional status messages will be displayed for other scenarios. If a connection could not be made the label will indicate that the application could not connect and a warning icon will appear.

Given that a connection has been made the GUI is now fully operational. The "Connect to Machine" Button will be disabled and the homing button will appear. The machine needs to be homed before it can be used. The controls to manually move the machine are contained within the Controls area. It is possible to rotate and translate the machine using the left and right arrows and clockwise, counterclockwise arrows.

The Playback area is used to tape a part from start to finish. Once a file has been loaded the user can hit play, indicated by the green play button, this will move the machine to the appropriate place. From there the user hits the green check mark to indicate the tape has been placed and then is required to hit play again to continue the taping process. At any point in time during playback the user can hit the stop or pause buttons to stop the machine completely or pause it.

To load a file for playback, navigate, using the start menu, to File -¿ Open Program File. A file manager will appear and prompt for the file to use, the file manager is specific to the operating system. For a Windows operating system the regular Windows file manager will appear. If a valid file is clicked, the program will load the values inside of the file into the application. For more information see section 2.2 for file specifications.

The GUI provides the functionality to program a new part and save that into a file. This can be done from the Program New Part area. The specific information required is listed, for more details see the Final Report document. Input validation is done for each of these fields, to make sure the user programs a part properly. The file can be loaded into the program immediately, as opposed to saving it then opening it, by using the "Load Changes on Creation" checkbox. The save button will prompt the user to save the information into a file. All fields can be cleared used the Eraser button which is next to the Save button.

The last area on the main window is the Quick View area. This shows some quick information about the program details of the current file loaded, as a sanity check to make sure the proper file was in fact opened. The current filename will be displayed under the machine status area.

The Menu bar at the top of the application contains some addition commands. Under View a user can access the Serial Monitor, which is explained in further detail in the next section. Under the Settings tab is Parameters which will modify the specific values used by the application for jogging the machine.

## 2 Code Structure

The code is structured in the following way under, under src are a number of different directories (java modules).

```
src/
    Components/
        Menu/
    FileIO/
    GCodeUtil/
    Main/
    Resources/
    Serial/
    Test Programs/
```

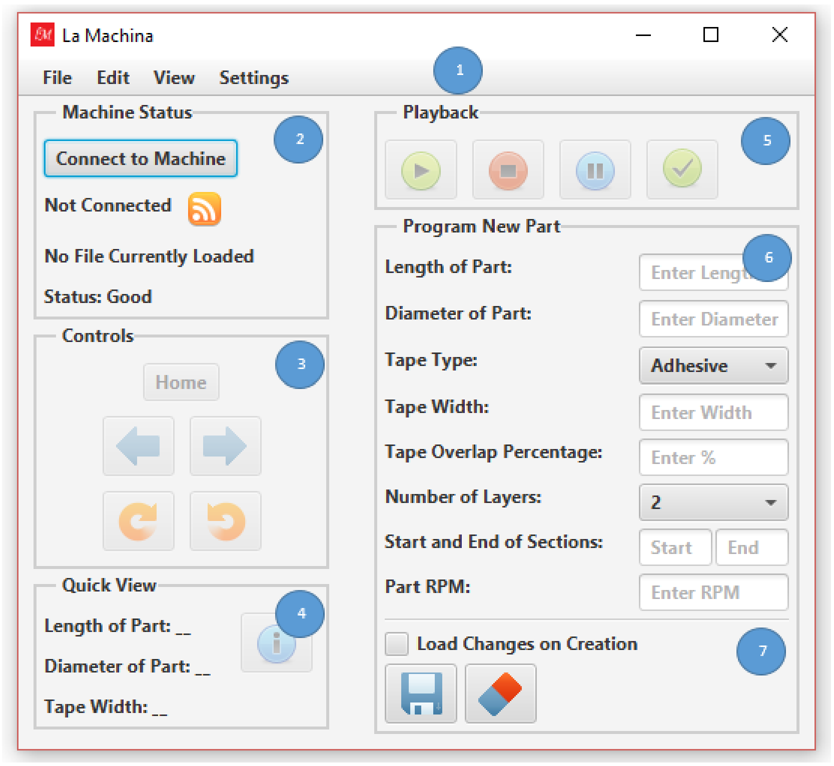
Beginning from top to bottom. Inside of `Components are all the java files related to the graphical look and feel, with the only exception being the main application source file located in Main. Components also contains the Menu/ package which is used to further split up graphical components related to the menu bar. The application layout is structured by using a number of different horizontal or vertical components. These are the VBox and HBox javafx classes. The classes within this directory have been subclassed to fit the specific application needs.`

## 2.1 Components

Looking more specifically at the classes inside of the Components module:

```
Components/
    Menu/ (1)
        FileMenu.java
        LaMachinaMenuBar.java
    ComponentInterface.java
    FileControlHBox.java (7)
    FullViewVBox.java
    MachineStatusVBox.java (2)
    MovementControlsVBox.java (3)
    ParameterWindowHBox.java
    PartCreationVBox.java (6)
    PlaybackVBox.java (5)
    QuickViewVBox.java (4)
    SerialMonitorVBox.java
```

Again, most of these classes either subclass VBox or HBox, both can be derived from the file names themselves. Additionally all components implement the ComponentInterface which further organizes functionality and allows each java class to be recognized as a type of the graphical application.



Not Connected

No File Currently Loaded

Status: Good

Length of Part: __

Diameter of Part: __

Tape Width: __

Length of Part: Enter Length

Diameter of Part: Enter Diameter

Tape Type: Adhesive

Tape Width: Enter Width

Tape Overlap Percentage: Enter %

Number of Layers: 2

Start and End of Sections: Start End

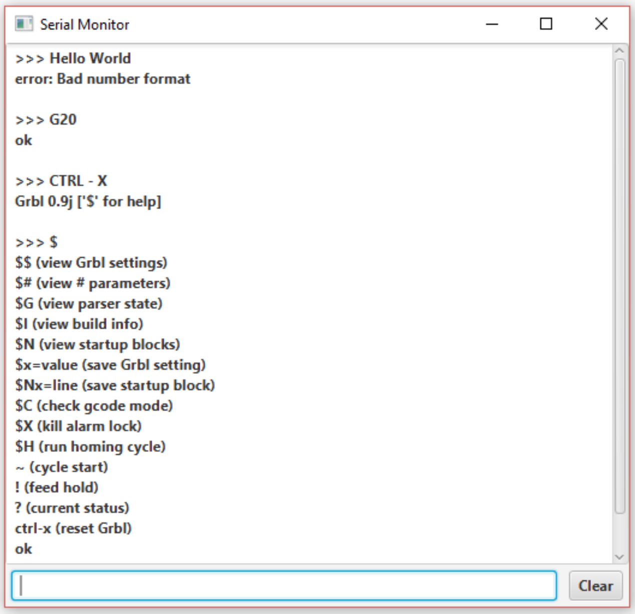
Part RPM: Enter RPM

Figure 2: Labeled Component Areas

For convenience Figure 2. provides numbered labels to identify the sections which correspond to java class files. For example all of the code for the Controls area labeled "3" falls within the MovementControlsVBox.java file.

Everything associated with the menu, labeled 1, is contained within the Menu module sub-directory. It should be noted that FileControlHBox (7) is actually contained within the PartCreationVBox (6). This was done to keep things more modular and because the functionality to save a new part should be in the same area as the part creation itself.

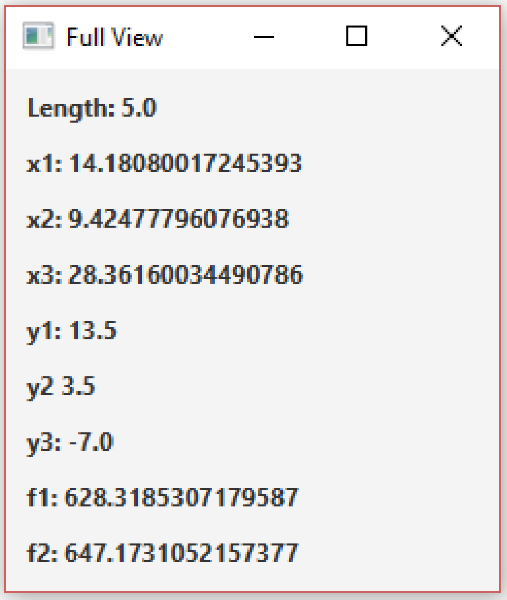
The remaining files within the package relate to items not seen on the main window of the application. These include the SerialMonitorVBox, ParameterWindowHBox and FullViewVBox.



```
>>> Hello World
error: Bad number format

>>> G20
ok

>>> CTRL - X
Grbl 0.9j ['$' for help]

>>> $
$$ (view Grbl settings)
$# (view # parameters)
$G (view parser state)
$I (view build info)
$N (view startup blocks)
$x=value (save Grbl setting)
$Nx=line (save startup block)
$C (check gcode mode)
$X (kill alarm lock)
$H (run homing cycle)
~ (cycle start)
! (feed hold)
? (current status)
ctrl-x (reset Grbl)
ok
```

Figure 3: Serial Monitor

The Serial Monitor allows direct communication with the arduino, specifically the grbl firmware, which in turn communicates with the hardware of the machine. This interface is accessed through the Menu by going to View -> Serial Monitor.



Length: 5.0

x1: 14.18080017245393

x2: 9.42477796076938

x3: 28.36160034490786

y1: 13.5

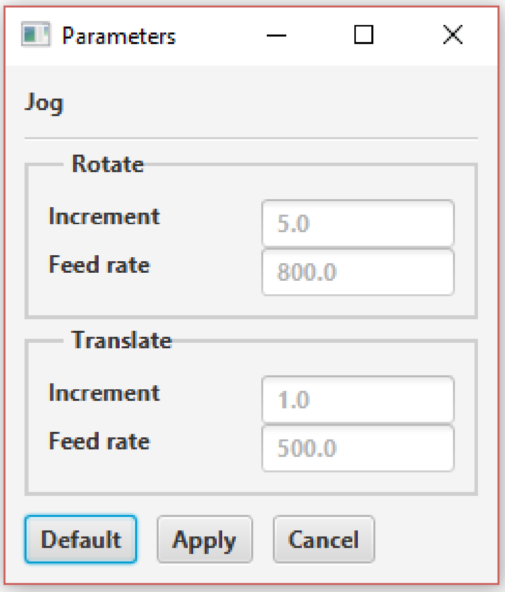
y2 3.5

y3: -7.0

f1: 628.3185307179587

f2: 647.1731052157377

(a) Full View Window



Rotate

Increment 5.0

Feed rate 800.0

Translate

Increment 1.0

Feed rate 500.0

(b) Parameters Window

The above figures round out the components inside of the Components sub-directory. The Full View Window corresponds to the FullViewVBox, Parameters Window to ParameterWindowHBox.

The Serial Monitor is useful for when debugging needs to be done, more specific grbl commands need to be entered or an unexpected error occurs with the firmware and certain instructions are needed.

The Full View window can be accessed once a part file has been loaded. Clicking on the information icon (I) inside of Quick View will open up the Full View window. The information contained within the Full View window are all of the parameters needed to program a part. These variables are used to create the grbl instructions for taping a part.

The Parameters window is used to update the existing parameters file containing specific values used by the machine to control jog.

## 2.2 File Input and Output

Underneath the FileIO package are all of the classes used for file input and output.

```
FileIO/
   ProgramFileReader.java
   ProgramFileWriter.java
```

There are only two files within this package, one for reading in files and one for writing files. Each class contains different methods for reading and writing the different types of files used by the application.

There are two main types of files read in and written out. There are the program files and parameter files. A program, in the scope of this document, refers to a sequence of instructions used to tape a part. The program file is in the form of:

```
c Program file
c <length> <diameter> <Tape Width> <Tape Overlap %> <Start> <End> <Part RPM>
p1 1.0 2.0 3.0 4.0 10.0 20.0 100.0
c <length> <x1> <x2> <x3> <y1> <y2> <y3> <f1> <f2>
p2 5.0 14.18080017245 9.42477796076 28.36160034490 13.5 3.5 -7.0 628.3185307179 647.1731052157
```

Comments are represented by a 'c' character. 'p1' indicates the input parameters typed in by the user. The 'p2' character indicates the mathematical parameters generated by the application for the gcode instructions. Both sets of parameters are written so that they do not need to be generated again, once they are loaded into the application. The comment sections above each line indicate what the numbers represent.

The second type of file to be read in or written out is the parameter file. Again these are used to control the machine manually. This corresponds to the parameter window figure (b). The parameter file is in the form of:

```
c This file is never modified, and serves as the default parameter settings file
c Parameter file
c <x increment> <x speed> <y increment> <y speed> <max part RPM>
p 1.0 100.0 1.0 100.0 120.0
```

The first characters follows the same style as the program file. There is only one Parameter Settings File used by the application. The default one is contained within Main this one is never changed. If the user wishes to change the parameter settings a secondary file will be written and placed under src. The application will first try and load the user specified file for parameter settings and if it cannot find that file it will default to using the Default Parameter Settings file.

## 2.3 GCodeUtil

The next module underneath src is the GCodeUtil module.

```
GCodeUtil/
    GCodeGenerator.java
```

The GCodeGenerator.java file contains all of the functions for manipulating values and outputting valid GCode messages. The main mathematics behind this comes from the function

```
    public static double[] modifyParameters(double[] params)
    {
        double tHalfLength = (params[5] - params[4])/2;
        double y1 = params[4] + tHalfLength - params[2]/2;
        double y2 = tHalfLength - params[2]/2;
        double y3 = 2*y2;
        double x1 = ((tHalfLength + params[2]/2)*params[1] * Math.PI)
                /(params[2]*((100-params[3])/100));
        double x2 = Math.PI*params[1]*1.5;
        double x3 = 2*x1;
        double f1 = Math.PI*params[6]*params[1];
        double f2 = (Math.sqrt(x1*x1 + y2*y2)/x1)*f1;
        if (f2 > 200*1.5*Math.PI)
        {
            f2 = 200*1.5*Math.PI;
            f1 = f2*(x1/Math.sqrt(x1*x1 + y2*y2));
        }
        return new double[]{tHalfLength, x1, x2, x3, y1, y2, y3, f1, f2};
    }
```

This takes in the parameters given by the program file and returns the modified instructions. These correspond to lines p1 and p2 within the program file. The purpose of this file is to provide the rest of the application with everything it needs to utilize GCode messages. For more details on GCode see grbl documentation.

## 2.4 Main

Main contains the main java file, LaMachinaGUI.java which is the starting point for the application.

```
Main/
   Default Parameter Settings
   GraphicsPreloader.java
   LaMachinaGUI.java
```

The LaMachinaGUI java class extends Application which is how a java class uses javafx. It creates all the different components and the displays them to the window.

The GraphicsPreloader class is used to display the splash screen seen on the following page.

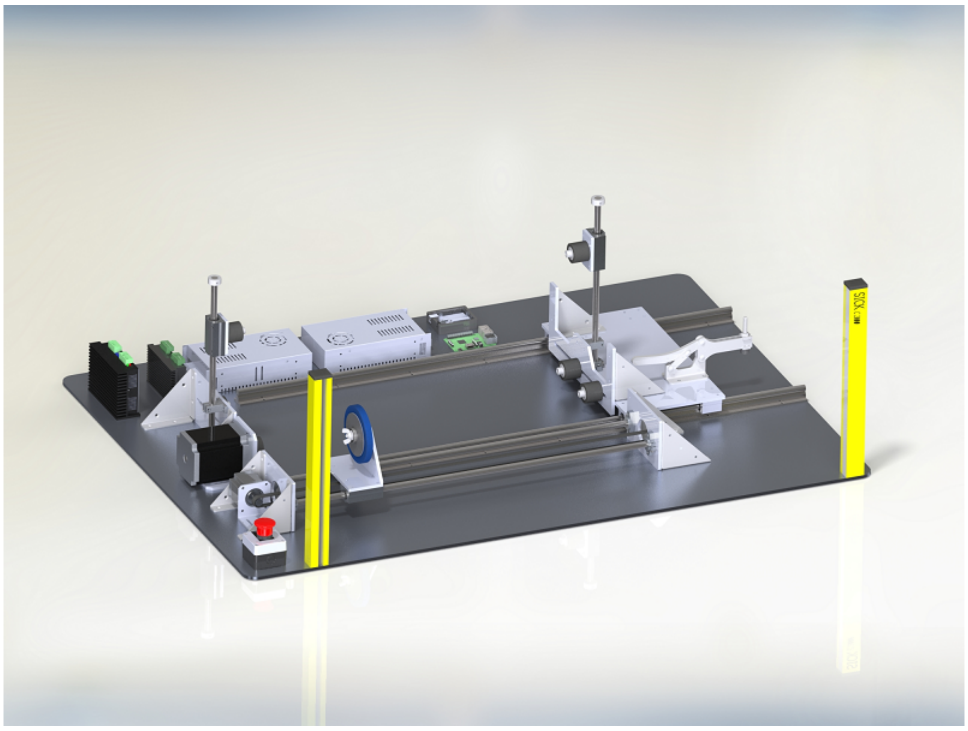


Figure 5: Splash Screen

## 2.5 Resources and Test Programs

The next two subsections are combined since they have little to do with actual code implementation and are simply regular directories for programs and resources.

```
Resources/
    Check.png
    ...
    Warning.png

Test Programs/
    Test1
    ...
    Test8
```

To keep this document shorter ... were used in place of listing all files contained with the directories. As a rule only resources, in this case image (.png) files, are kept within the Resources directory. The application uses the images mainly for buttons. The test programs section is simply a place to store test programs. The application does not care where test program files are stored however. They can be in any location the user wishes them to be.

## 2.6 Serial

The final module within src is the Serial module. This directory contains all of the code used to do serial communication with the arduino.

```
Serial/
   AbstractSerialConnection.java
   ArduinoSerialConnection.java
   SerialCommunication.java
   UpdateMessageEnum.java
```

Again the purpose of separating out the functionality into its own package was to restrict the areas of code being impacted by serial IO. The object given to the main application is an instance of SerialCommunication. Whenever the application needs to communicate GCode to the arduino it simply calls a method within the SerialCommunication object, specifically sendMessage

```
/**
 * Send messages writes out a string
 * on the serial connection
 * @param s - the string to write out.
 */
public void sendMessage(String s)
{
    try
    {
        connection.writeString(s + "\n");
    }
    catch (SerialPortException e)
    {

        e.printStackTrace();
    }
}
```

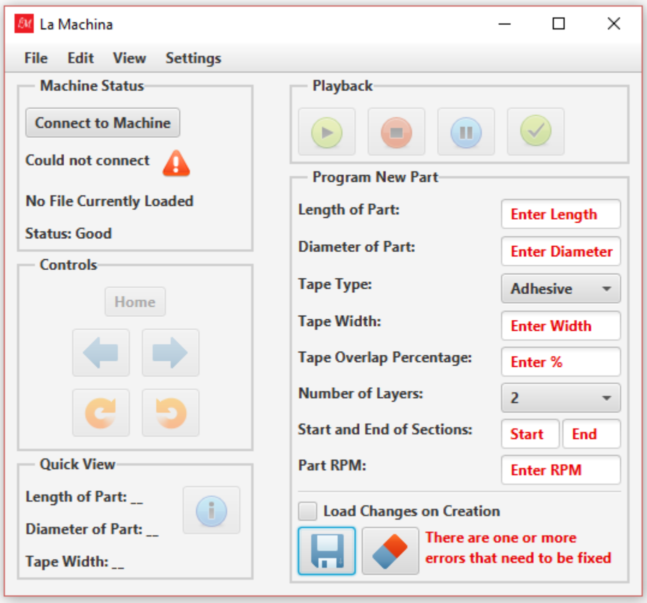
The SerialCommunication object contains within it the ArduinoSerialConnection object which is the underlying connection to the arduino. When the application calls sendMessage, the SerialCommunication object makes use of ArduinoSerialConnection to serially write that message out to the USB. These classes also contain functionality to send individual bytes and can open and close different serial connections. The purpose of the AbstractSerialConnection is to implement an interface for future serial communications that are not specific for an arduino. This class contains basic functionality that would apply to any such connection.

## 3 Error Cases and Hints

### 3.1 Cannot Connect and Bad Input

As mentioned there are instances where an error could occur. This section lays out a few handled error cases. One error which was touched upon briefly is what happens when the application cannot connect to the arduino. A second error is if a user tries to enter bad values when programming a part.

Other issues that may come up are preempted by the application itself. For instance certain buttons are disabled until such a time as they are safe to use. The jog rotations and translations are disabled until the machine has homed itself. The playback buttons are disabled until a program has actually been loaded into the machine.



Could not connect

No File Currently Loaded

Status: Good

Length of Part: __

Diameter of Part: __

Tape Width: __

Length of Part: Enter Length

Diameter of Part: Enter Diameter

Tape Type: Adhesive

Tape Width: Enter Width

Tape Overlap Percentage: Enter %

Number of Layers: 2

Start and End of Sections: Start End

Part RPM: Enter RPM

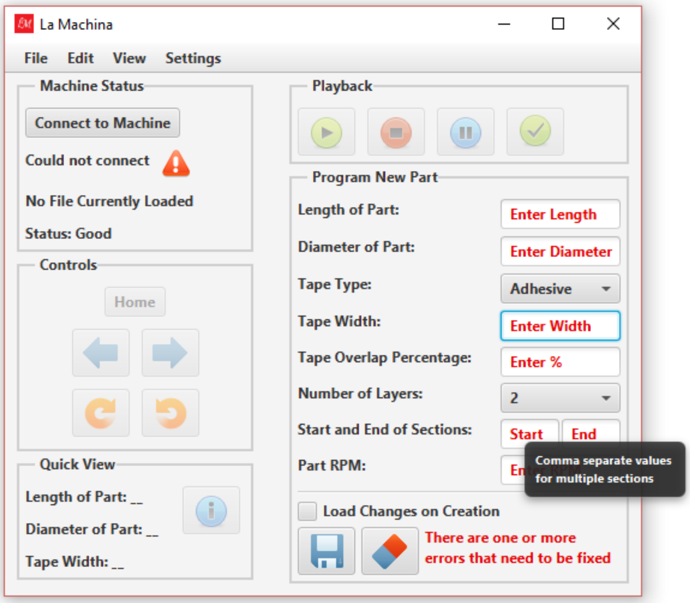
There are one or more errors that need to be fixed

Figure 6: Errors and Bad Input

The above figure shows what will happen when both of the errors previously described happen. The application warns the user that it could not connect inside the Machine Status area and all of the fields that contained an error are changed to red text. Since it is difficult to pinpoint exactly which field was wrong, in the case of many incorrect entries, a simple message telling the user that incorrect input was given is used.

## 3.2 Hints

The program also provides the user with hints as to what certain areas mean and what specific buttons do. These hints come in the form of "tool tips" that when a user mouses over them give further directions.



Could not connect

No File Currently Loaded

Status: Good

Length of Part: __

Diameter of Part: __

Tape Width: __

Length of Part: Enter Length

Diameter of Part: Enter Diameter

Tape Type: Adhesive

Tape Width: Enter Width

Tape Overlap Percentage: Enter %

Number of Layers: 2

Start and End of Sections: Start End

Part RPM: En

Comma separate values for multiple sections

There are one or more errors that need to be fixed

Figure 7: Hints

## List of Figures