Computer Engineering and Mechatronics Project: Developing software for a writing robot

Part 1: Project Planning Assignment

Introduction

You will be developing the software required to transmit commands from a file, via a virtual RS232 serial port, to the writing robot such that it is able to 'draw out' text as read from a file as per Figure 1.

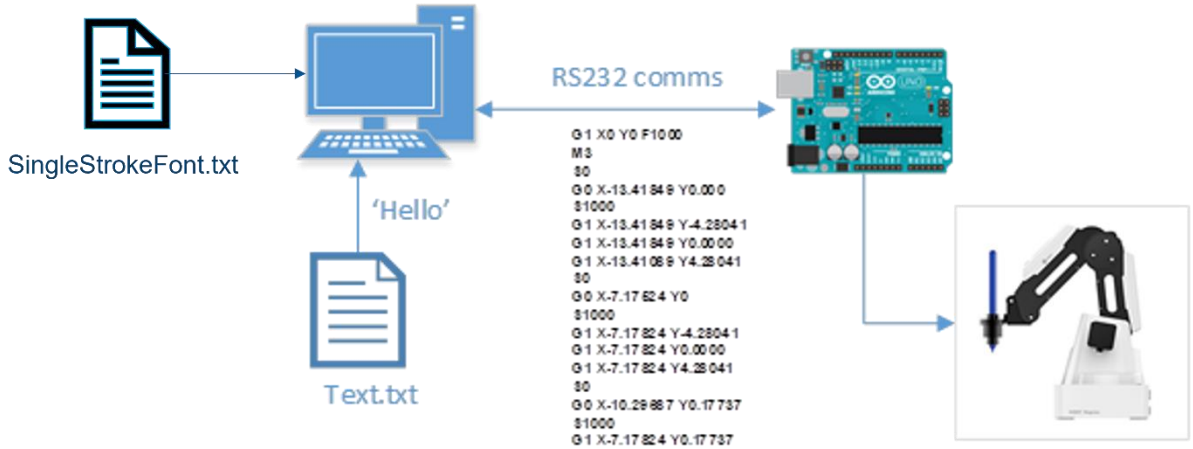

Figure 1: Overview of the application

Font data is provided on Moodle in the file 'SingleStrokeFont.txt. Details of the format are provided in Appendix 1. Note: As part of the task you will be required to scale the x and y movements such that the height of a letter (excluding ascenders and descenders) is 5mm; it is 18 units in the font file, so that the movements derived from the font data will need to be scaled within your program by the fraction 5/18 (0.278) so that they come out 5 mm high.

As explained in Appendix 1, each letter will need to be offset in the X direction so that its origin corresponds to the final point in the definition of the previous letter written. Furthermore, successive lines will need to be spaced 10 mm down from each other. In other words, you will need to consider both the X offset of each letter from the previous one on the same line, and the Y offset of each line from the previous one. The Y-offsetting of lines, and the re-setting of the X-offset for the start of the new line, will be triggered by the LF (ASCII 10) and CR (ASCII 13) codes respectively in the text file.

Your code is required to generate and send the required G-Code commands (based on the font file) to the Arduino to raise and lower the pen and to move the arm to specified X,Y locations (details on the G-Code format is provided in Appendix 2). The Arduino has been pre-programmed to accept the G-codes and to transmit them to the writing arm.

The text to be written will be read from a text file. The code should be written such that it will process a file containing text of any length.

To assist you in developing your application you will be provided with code developed for CodeBlocks to handle the serial communications and a sample project that shows how serial communication is initialised and then how data is sent and received (Appendix 3). A more complex worked example (Appendix 4), which will form the starting point for writing your program, illustrates the sending of G code to the Arduino and awaiting the "ok" acknowledgement that a new block of G code is expected.

Testing of your application

A basic simulator (for Windows) is available at https://nraynaud.github.io/webgcode/ to allow you to verify the commands intended to be sent to the writing robot.

Testing of your code with the real robot will take place during the week beginning 13th December in the laboratory sessions

Submission

The project planning assignment submission is a document which provides a specification for the software which will be developed in the final submission. Please note that you do **not** need to write any code for this submission.

Software Project Planning Submission (5%)

Learning outcomes:

- To be able to analyse a design brief to understand software requirements
- To produce a software design to fit a set of software requirements

Learning outcomes will be demonstrated in the software planning document submitted by:

- Giving an explanation of precisely what the program needs to do (explain it in plain English).

- Planning the program with the aid of flowcharts
    - Draw a flowchart with sufficient detail that it is possible to use it as a basis for writing the code for the program.
    - Include blocks to indicate function calls and flowcharts for the function where appropriate

- Defining the function declarations (prototypes) for all functions to be used in the program
    - Define all parameters and their types using meaningful names
    - Show whether parameters are input and/or output, whether they are changed and the return value, if any.
    - Includes functions which you will write yourself, not the ones which are supplied in the program template.

- Specifying the main data items required in the program.
    - Give data type and why this has been chosen.

- Creating test data which will validate the program, confirming conformance of the program/function to its specification.
    - Ensure that all routes through the program are covered.
    - Include input data with expected outputs.

A template is provided for the submission on Moodle (ProjectPlanningTemplate21-22.docx). Save your planning document as **RobotPlanningXXX.pdf** where **XXX** is your initial. The flowchart(s) may be submitted in a separate pdf saved as **RobotFlowchartXXX.pdf** (do not submit a drawio file, use the export function in draw.io to save to pdf).

The deadline for submission to Moodle of this document is **6pm Monday 22nd November**.

References

Images:	Robot arm, https://stampanti3d.club/
	Arduino board: https://www.arduino.cc/
Font:	Derived from http://phk.freebsd.dk/hacks/Wargames/index.html
RS232 library:	https://www.teuniz.net/RS-232/index.html
The real robot:	http://eleksdraw.eleksmaker.com/

Appendix 1: Font File format

NOTE: This should be read and understood before considering the 'G-Code format'

The font file *SingleStrokeFont.txt* provided contains X,Y & pen up/down data required to draw out all ASCII characters (those we can type on a standard keyboard) – there is also one 'special' case which is 'character 1' which draws out the HP logo.

The general format for the file is as per Figure 2

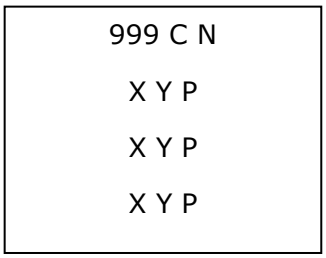

Figure 2 – Template for font information

Where

- 999: Indicates a new character is to be defined
- C: is the character number (its ASCII code, in decimal form)
- N: is the number of movements required to 'draw' the character
- X: the X position to move to (relative to 0,0)
- Y: the X position to move to (relative to 0,0)
- P: Pen up/down (0=up so no line is draw, 1=down so causing a line to be drawn)

Some important things to note

- ALL movements are **OFFSETS** relative to 0,0 (bottom left, the letter's local origin)

- The last 'X Y P' line moves the writing arm to the position to draw the next letter – this allows the new 'origin' to be determined (and to which the offsets for the next character to be drawn are added).

Example 1: Font data for a single character

If we consider the font information for the letter 'H', ASCII character 72 as shown in table 1.

Font data	Details
999 72 7	Character 72 is about the be defined 7 lines of movement information
0 0 0	Move arm to position 0,0 with pen up
0 18 1	Move arm to position 0,18 with pen down (draws line)
12 0 0	Move arm to position 12,0 with pen up
12 18 1	Move arm to position to 12,18 with pen down (draws line)
0 9 0	Move arm to position 0,9 with pen up
12 9 1	Move arm to position to 12,9 with pen down (draws line)
18 0 0	Move arm to position 18,0 with pen up (this is the starting position for the next character to be drawn)

Table 1: Line and pen drawing commands for H

Based on table 1, to 'draw' a H we would use the 7 lines (from 0,0,0 to 18,0,0)

Example 2: Font data for multiple characters

If we wished to two letters H alongside each other we repeat the data however for the second 'H' we need to add to the X&Y values an offset of 18,0 – if we did not the characters would be drawn over each other

Table 2 shows both the correct and 'wrong' (overwriting) versions of the font data we would send for 'HH'

Letter to be draw	**Correct Font data** Offset from previous 'H' added	Incorrect: no offset 2nd H will be written over the 1st.
H	0 0 0	0 0 0
	0 18 1	0 18 1
	12 0 0	12 0 0
	12 18 1	12 18 1
	0 9 0	0 9 0
	12 9 1	12 9 1
	18 0 0	18 0 0
H	18 0 0	0 0 0
	18 18 1	0 18 1
	30 0 0	12 0 0
	30 18 1	12 18 1
	18 9 0	0 9 0
	30 9 1	12 9 1
	36 0 0	18 0 0

Table 2: Line and pen drawing commands for HH

Appendix 2: G-Code

G-Code is a well-established language originally developed for programming of numerical control (NC, now known as computer numerical control or CNC) machine tools – it is now widely used for control of 3D printers.

A file in G-code format is traditionally known as a "part program" (in that it is 'programming code' that is interpreted by software to control a system).

The subset of G codes and related to the project are shown in Table 3

Command	Description
F1000	feed rate (i.e. pen speed) 1000 mm min^{-1}
G0 X Y	Move to the position X,Y
G1 X Y	Draw a straight line from the last position to X,Y
M3	Turn on Spindle (needed for arm to work!)
S0	Pen up (original meaning is 'spindle speed 0')
S1000	Pen down (original meaning is 'spindle speed 1000 rev min^{-1}')

Table 3: Required G Codes

Generating G-Code from the font drawing information.

You will need to convert the font data (as detailed in Appendix 1) to G-Code before it can be sent, via the communications channel, to the writing arm.

If we consider the case for sending a single letter H starting at the origin (Table 1, Appendix 1) we would get the commands as shown in table 4.

Font data	G-Codes to send	Details
Initialisation (send once)	F1000	Set pen speed
	M3	Turn on spindle
	S0	Pen up
0 0 0	G1 0 0	Go to 0,0
0 18 1	S1000 G1 X0 Y18	Pen down Move to 0,18 (line drawn as pen down)
12 0 0	S0 G0 X12 Y0	Pen up Move to 12,0
12 18 1	S1000 G1 X12 Y18	Pen down Move to 12,18 (line drawn as pen down)
0 9 0	S0 G0 X0 Y9	Pen up Move to 0,9
12 9 1	S1000 G1 X12 Y9	Pen down Move to 12,9 (line drawn as pen down)
18 0 0	S0 G0 X18 Y0	Pen up Move to 0,18

Table 4: comparison of font commands and G Code data

Points to note:

- Some commands are required to be sent before the 'drawing commands' to correctly initialise the arm
- The S1000/S0 commands need only be sent if the pen up/down state has changed
- The X & Y values in the above, in your application, will need to be scaled to obtain the correct font size.

Appendix 3: Communication protocol

Each G-code command is sent via a virtual serial port at 115200 baud as a string of ASCII characters terminated with a "newline" character **\n**.

When the robot is ready to receive the next command it responds with the message "ok" followed by **\n**.

In practice the existing program on the robot stores a series of G-code commands in a "buffer" (a queue of memory) until the buffer is full.

When the buffer is full, no further "ok" message is sent until there is more space in the buffer. (This enables the robot to make a rapid series of movements without having to wait for each command to be sent individually).

Communication using a virtual serial port

True serial ports (using the RS-232 protocol including ±12 V signal levels) are rare on modern PCs, but communication with Arduinos and similar devices takes place via one or more virtual serial ports which follow similar protocols but at 5 V logic levels.

Use of serial communication in a C program is not straightforward so you will be using the RS-232 library written by Teunis van Beelen, available from https://www.teuniz.net/RS-232/index.html.

For convenience, his example files for transmitting and receiving text have been combined, modified and incorporated into a Code:Blocks project which is available from the Moodle website.

The zip file, Robot_writer_v3, including the Code:Blocks project also includes the complete file distribution for that library along with the RS-232 project web page and the licence file, and a suitable test program for the Arduino. Unsurprisingly, perhaps, the combination of the two programs forms a rather unusual form of our old friend "Blink"!

You are encouraged to try out the program as follows:

1. Unzip the project to a suitable location.
2. Plug in your Arduino and make a note of the COM port number
3. Open BlinkSerialArduino.ino and compile and upload it.
4. Open the serial monitor, ensure that "newline" is selected and that the baud rate is set to 115200
5. You should be able to switch the LED on and off by typing "on" or "off" in the serial monitor.
6. Close the serial monitor or the next stage won't work!
7. Open BlinkSerialPC.cbp in Code::Blocks
8. Set the COM port number in BlinkSerial.c as explained in the comment near the start of the program. Note that the number entered is the COM port number minus 1.
9. Compile and run BlinkSerialPC.c (linked with RS232.c) . Check the COM port has been correctly set. If not, correct it and try again.
10. The Arduino should blink at 0.5 Hz, and the command window should display the commands sent to the Arduino and the acknowledgements received back.
11. To close the command window, press Ctrl-C.

Appendix 4: Example of sending G code to Arduino and awaiting acknowledgement

A rather more complex worked example, which will form the starting point for your project, is also available on Moodle as Robot Writer. The steps for getting this going are as follows:

1. Unzip the file Robot_writer_v3.zip file to a suitable location.
2. Plug in your Arduino and make a note of the COM port number
3. Open SerialEchoBlink.ino and compile and upload it.
4. Open the serial monitor, ensure that "newline" is selected and that the baud rate is set to 115200. The serial monitor should display a greeting terminated by a "$" sign e.g. "Test sketch to emulate writing robot $"
5. You should be able to switch the LED on and off by typing typing random text in the serial monitor and pressing "return" – each time the serial monitor should display "ok" and the LED should change state.
6. Close the serial monitor or the next stage won't work!
7. Open Robot_writer.cbp in Code::Blocks
8. Set the COM port number in serial.h as explained in the comment near the start of the program. Note that the number entered is the COM port number minus 1.
9. Build and run the project. Check the COM port has been correctly set. If not, correct it and try again.
10. The Code::Blocks command window should show the "conversation" between the C program (running on the computer) and the sketch running on the Arduino, with a series of (hard-coded) G-code commands being sent to the Arduino and acknowledged with "ok".
11. To close the command window, simply press return at the end of the program. If for any reason the program gets stuck, press Ctrl-C.

Your job in the project is to replace the hard-coded G-code commands with your own programming in order to read the text file and generate G-code to write the text accordingly on the robot. You should be able to use this program as the starting point for your own program, though you need to plan your work before you can do so!

Additional steps for compiling code with CodeBlocks with RS232 code

If, when compiling your code, you receive the error

undefined reference to `timeGetTime@0'

You need to add an additional library when linking, to do this follow the steps below

- From the menu select 'Settings' and from the dropdown select 'Compiler'
- On the form that appears select the 'Linker Settings' tab
- Click on the 'Add' button and in the dialog for the 'File:' type libwinmm.a
- Press OK,
- The link libraries box should now have an additional entry of 'libwinmm.a' as shown in figure 2
- Press OK

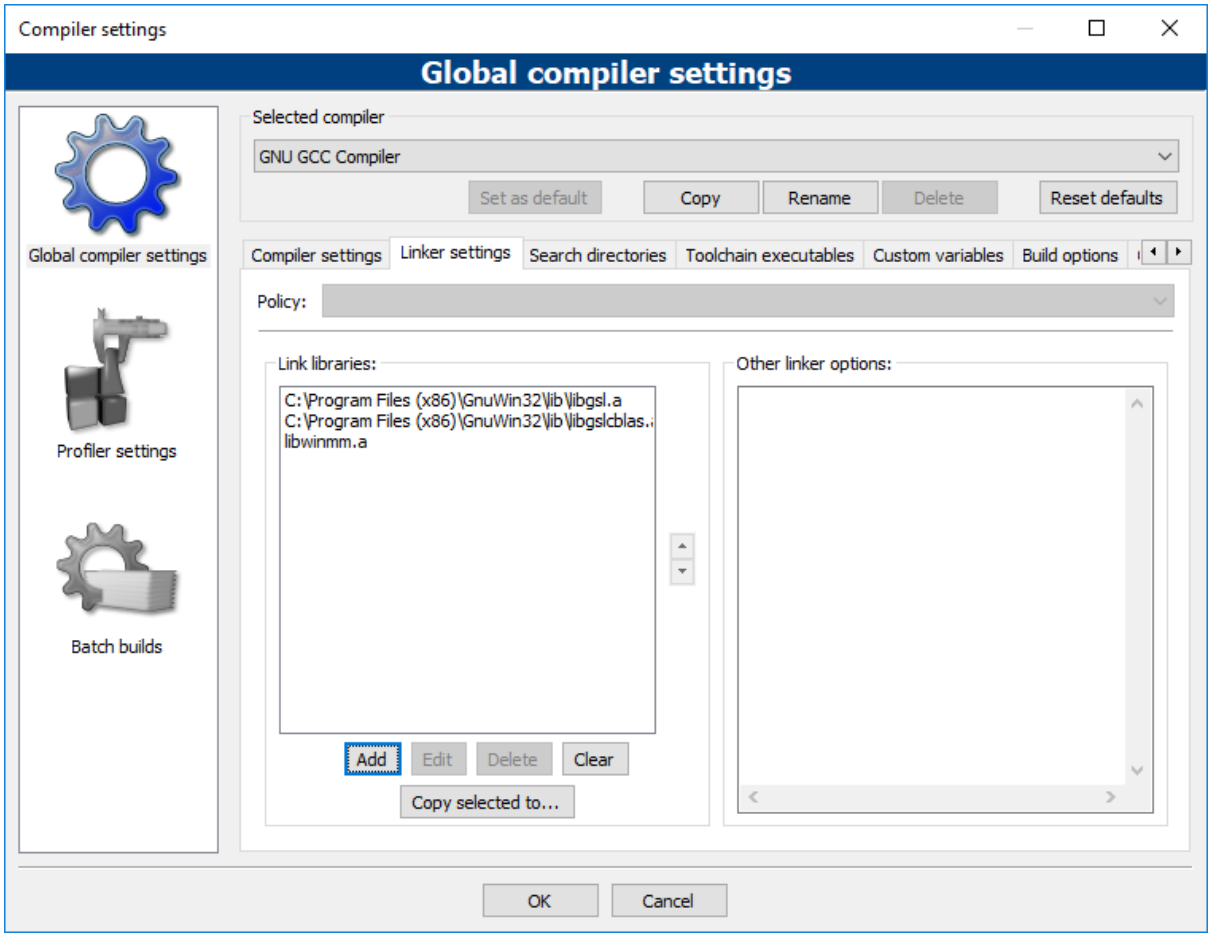

Figure 2: Additional library added to CodeBlocks project