# Archie Clayton, 20454063

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

# Software Description

The project is designed to control a robotic pen, using G-code commands to draw text specified in a text file. The software reads font data from a predefined file (SingleStrokeFont.txt), processes the text from the input file, and then generates the corresponding G-code that then moves the robot’s pen to form the characters. The main goal is to allow the robot to draw text on a physical surface based on the defined character shapes, and the program includes features such as adjusting character height, handling text across multiple lines, and ensuring proper spacing between words.

At the start of the program, the font data is read from a file called ‘SingleStrokeFont.txt.’ Each character is represented by a series of strokes defined by their X and Y coordinates. The pen's state (up or down) is also recorded for each stroke, which determines whether the robot should move without drawing or make a pen stroke. This stroke data is stored in an array of ‘FontChar’ structures, which hold the character's ASCII code, the number of strokes, and the stroke data for the X and Y coordinates as well as the pen state. This storage format is essential for the program’s ability to generate the movement instructions for the robot.

Once the font data is read, the user is prompted to input the desired height for the characters, which must be between 4 and 10 mm. This value is used to calculate a scaling factor, which is then applied to the X and Y coordinates of each stroke. The scaling ensures that the drawn text corresponds to the size specified by the user. If the input height falls outside this range, the program displays an error message and terminates. The scaling factor is computed by dividing the input height by 18, as the font data is originally defined with a height of 18 units. This calculation ensures that the scaled strokes fit the desired size on the drawing surface.

After determining the scaling factor, the program prompts the user for the name of the text file to be drawn. The program reads the file, processing it word by word. For each word, the program calculates its total width, including the scaling factor and the spacing between characters. If a word exceeds the available width of the line, the program moves the drawing to the next line and resets the X-offset to the beginning of the new line. This ensures that the text is correctly formatted and does not overflow beyond the drawing area. If a word is too large to fit on the current line, it is wrapped automatically.

For each word, the program generates G-code commands to control the robot’s movements. The G-code includes instructions to move the pen to each stroke’s X, Y coordinates while adjusting the pen state as needed. The program ensures that the pen is lifted (S0) when moving to a new position and lowered (S1000) when drawing. These commands are sent sequentially to the robot via a serial interface (RS232). The program waits for the robot to acknowledge each command before sending the next, ensuring that the robot performs the drawing operations accurately.

The program also includes error handling mechanisms. It checks for valid input from the user, verifies that files can be opened successfully, and handles cases where a word exceeds the available line width. If any of these errors occur, appropriate messages are displayed to the user, and the program terminates to prevent incorrect behaviour. After processing all words in the text file, the program sends a final G-code command to lift the pen and return the robot to the starting position at coordinates (X=0, Y=0), ensuring that the robot is ready for the next task.

A black background with white text

Description automatically generatedA graph with a purple line

Description automatically generatedSample Output

# Project Files

1. Main.c

This is the core of the program, where the user can interact with the system. It initialises the communication with the robot, processes font and text data, scales the text and generate the G-code.

Key function of this code includes aspects such as opens the serial port using the functions provided in serial.h and rs232.h. It reads font data from SingleStrokeFont.txt and text from the user-specified text file, then processes the font and text data to generate G-code, which is sent to the robot. It handles scaling of characters and ensures proper line breaks and word spacing during drawing, then finally closes the serial port and finalises the process.

The COM port can be set in the serial.h file, which is used by main.c. Conditional compilation is not directly used in this file, but it relies on configuration flags in serial.c and rs232.c.

1. Serial.h

This header file declares the functions for managing serial communication. It ensures consistency between the declarations and the implementations in serial.c.

Provides function prototypes for serial communication management: PrintBuffer: Sends a string to the serial port. WaitForReply: Waits for an acknowledgment from the robot. WaitForDollar: Waits for the robot's startup signal. CanRS232PortBeOpened: Checks if the serial port can be opened. mCloseRS232Port: Closes the serial port.

The #define cport\_nr and #define bdrate constants specify the COM port and baud rate, which may need adjustment depending on the robot and system configuration.

1. Serial.c

Implements the functions declared in serial.h. The Key Features includes the Initialisation and Cleanup where CanRS232PortBeOpened initialises the serial port and CloseRS232Port closes the serial connection. The Communication Functions then use PrintBuffer to send text to the robot or emulator. Then WaitForReply and WaitForDollar handle specific acknowledgment responses from the robot. A Conditional Compilation is the #ifdef Serial\_Mode directive allows switching between real and mock serial communication. If Serial\_Mode is defined, actual RS232 communication is enabled; otherwise, a mock implementation is used for testing or debugging.

1. Rs232.h

This file declares the low-level functions and constants used for serial communication. It ensures compatibility with different operating systems by including platform-specific headers. This features aspects such as an comprehensive API for serial communication, which will open, close, reading from, and writing to serial ports. Configuration Changes include the #if defined(\_\_linux\_\_) || defined(\_\_FreeBSD\_\_) which ensures the correct headers and dependencies are included based on the operating system.

1. Rs232.c

This file implements the functions declared in rs232.h. It directly interacts with the hardware or operating system to perform serial communication. The key Operations include RS232\_OpenComport which opens a serial port with specified parameters and RS232\_CloseComport which closes the port. RS232\_PollComport then reads data from the serial port. RS232\_SendBuf and RS232\_SendByte will send data to the port, while RS232\_cputs sends a null-terminated string to the port.

# Key Data Items

|  |  |  |
| --- | --- | --- |
| Name | Data type | Rationale |
| fontDataArray | FontChar [] | This array stores font data for all ASCII characters. The FontChar struct contains stroke details, including coordinates and pen states, this then allows it to be reused throughout. |
| FontChar | struct | A structured data type for each character's strokes, ensuring organised storage of multiple aspects such as the ASCII code, stroke count, and pen state. |
| x and y | Float [] | Arrays within FontChar to store the X and Y coordinates of strokes. Float is chosen to handle the positioning for the robot’s movements. |
| draw | int[] | Indicates whether the pen is up (0) or down (1) during each stroke. As is it only 1 or 0, integer can be used. |
| scalingFactor | float | Used to scale the font size based on user input. Float allows for scaling at the appropriate size. |
| x\_offset | float | Tracks the current X position of the text being drawn. Float provides precision for accurate spacing. |
| y\_offset | float | Tracks the current Y position of the text being drawn. Float ensures accurate positioning for different lines. |
| textFileName | char[100] | Stores the name of the input text file. The fixed-size character array accommodates typical file names without dynamic memory being needed. |
| |  | | --- | | line[256] |  |  | | --- | |  | | char[] | Temporarily holds lines read from the font data file. The size is suitable for typical line lengths with sufficient buffer space. |
| buffer | char[100] | Used to store G-code commands temporarily before sending them to the robot. The size is adequate for the G-code. |
| |  | | --- | | currentChar |  |  | | --- | |  | | int | Stores the ASCII code of the character being processed. Using int aligns with the ASCII range. |
| numStrokes | int | Tracks the number of remaining strokes for the current character. An integer is appropriate for counting purposes. |
| word[100] | Char [] | Stores individual words read from the input text file for processing. The size is chosen to handle typical word lengths. |
| wordWidth | Float | Stores the calculated width of the current word for line-wrapping decisions. Float ensures precision in width. |

# Functions

**To read font data from a file and populates the fontDataArray with stroke data for each character.**

int readFontData(const char \*filename, FontChar fontDataArray[])

* **Parameters**:
  + filename: A pointer to the name of the font file to be read.
  + fontDataArray[]: An array where the font data will be stored.
* **Return Value**: Returns 1 if the data is successfully read, otherwise returns 0 if an error occurs (e.g., file cannot be opened or invalid data format).
* **Operation Details**: This function reads through the font file line by line, extracting ASCII character data and the strokes for each character. If the format is correct, the data is stored in fontDataArray for later use.

**To generate G-code commands for drawing text based on the font data and the input text file.**

void generateGCodeForText(FontChar fontDataArray[], const char \*textFileName, float scalingFactor)

* **Parameters**:
  + fontDataArray[]: An array of FontChar structures containing the font data.
  + textFileName: A string containing the name of the text file that specifies the text to be drawn.
  + scalingFactor: A float used to scale the coordinates of the font data to match the desired size.
* **Return Value**: This function does not return a value.
* **Operation Details**: The function reads the specified text file word by word, calculates the width of each word, checks if it fits within the specified line width, and generates G-code for each character. It handles the movement of the pen across the canvas and sends G-code to the robot.

**Sends a single G-code command to the robot.**

void sendGCodeCommand(const char \*command)

* **Parameters**:
  + command: A string containing the G-code command to be sent.
* **Return Value**: This function does not return a value.
* **Operation Details**: It formats the command into a buffer and calls the SendCommands function to send the G-code to the robot.

**Prompts the user to input a valid character height, and then calculates the scaling factor.**

int validateHeight(float \*scalingFactor)

* **Parameters**:
  + scalingFactor: A pointer to a float that will store the calculated scaling factor based on the user input.
* **Return Value**: Returns 1 if the height is valid, otherwise returns 0.
* **Operation Details**: This function prompts the user for a height between 4 and 10 mm. If the input is valid, it calculates the scaling factor by dividing the height by 18, which is the standard height used for scaling.

**Generates G-code for a single word by processing each character individually.**

void generateWordGCode(FontChar fontDataArray[], const char \*word, float \*x\_offset, float \*y\_offset, float scalingFactor)

* **Purpose**: Generates G-code for a single word by processing each character individually.
* **Parameters**:
  + fontDataArray[]: An array of FontChar structures containing the font data.
  + word: A string representing the word to be processed.
  + x\_offset: A pointer to the current X offset for positioning the word.
  + y\_offset: A pointer to the current Y offset for positioning the word.
  + scalingFactor: A float used to scale the coordinates of the font data.
* **Return Value**: This function does not return a value.
* **Operation Details**: This function loops over each character in the word, processes its coordinates (by calling processCharacterCoordinates), and updates the x\_offset and y\_offset accordingly. It generates G-code for each character in the word.

**Processes the coordinates for a given character and generates G-code for each stroke.**

void processCharacterCoordinates(FontChar currentChar, float \*x\_offset, float \*y\_offset, float scalingFactor)

* **Parameters**:
  + currentChar: A FontChar structure representing the character being processed.
  + x\_offset: A pointer to the current X offset for positioning the character.
  + y\_offset: A pointer to the current Y offset for positioning the character.
  + scalingFactor: A float used to scale the coordinates of the font data.
* **Return Value**: This function does not return a value.
* **Operation Details**: For each stroke in the current character, this function generates the appropriate G-code command (pen down for drawing, pen up for moving). It updates the x\_offset and y\_offset to move to the next position as needed.

# Testing Information

|  |  |  |  |
| --- | --- | --- | --- |
| Function | Test Case | Test Data | Expected Output |
| readFontData | Valid Font Data | A valid font file named "SingleStrokeFont.txt" with correct stroke data for characters. | Returns 1 (success).  And the fontDataArray is populated with the stroke data for the characters. |
| readFontData | Invalid Font File (File Not Found) | A non-existent font file, e.g., "NoFont.txt". | Returns 0 (failure) and displays error: "Error: Unable to open file Nofont.txt." |
| readFontData | Invalid Font File (Font File with incorrect formatting) | A font file with incorrect formatting. | Returns 0 (failure) and displays error: "Error: Invalid stroke data format." |
| generateGCodeForText | Valid Text File | A valid text file named "text.txt" containing the word "Hello". | The function processes the text file, and for each character, it generates G-code to draw the text. It sends the G-code commands for each letter to the robot. |
| generateGCodeForText | Empty Text File | An empty text file "empty.txt". | No G-code is generated, and the function simply returns without generating any output. |
| sendGCodeCommand | Send Valid G-code Command | A valid G-code command such as "G1 X100 Y100". | The command is successfully formatted and passed to the SendCommands function. The G-code is printed or sent via the serial connection. |
| sendGCodeCommand | Send Invalid G-code Command | An invalid G-code command such as "G1 X Y" | The function attempts to send the command but will likely fail or not be executed correctly. |
| validateHeight | Valid Height Input | User inputs 6.0 for character height. | The scaling factor is calculated as 6.0 / 18.0 = 0. 333.Then returns 1 (success). |
| validateHeight | Invalid Height Input | User inputs 12.0 for character height. | Displays error: "Error: Height must be between 4 and 10 mm." Also, returns 0 (failure). |
| generateWordGCode | Valid Word | The word "Hello" | The function generates G-code for each character of the word "Hello". The G-code for each letter in the word is generated and passed to SendCommands. |
| generateWordGCode | Empty Word | An empty string "" | No G-code is generated and the function returns without processing any characters. |
| processCharacterCoordinates | Valid Character | Character 'A' | The function processes the character 'A' and generates G-code for its strokes. |
| Main( ) | Complete Program Flow | Valid font file, text file, height = 6.0 | Program runs without errors. Generates G-code and sends to the robot. The robot draws the text, and the COM port is closed at the end. |

*Extend table as required. Note that ‘Function’ includes main()*

# Flowchart(s)

Included on separate PDF