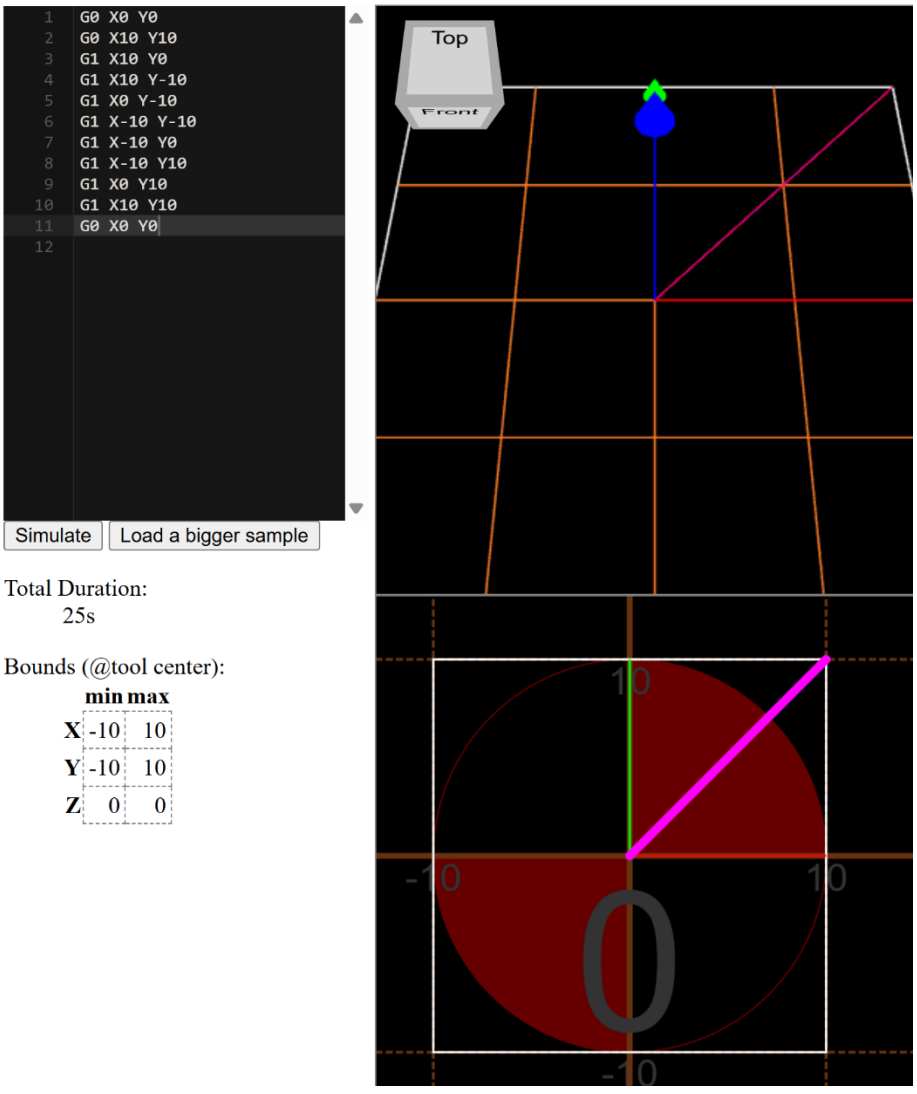
# Robot Writer 2025 – System Manual

## Program Overview

This program controls a robotic writing system, driving actuators via serial communication to draw characters based on text input.

- Initializes encoder state machines
- Reads sensor inputs based on input text and font files
- Sends commands to actuators via serial communication with RS232 port
- Returns pen to origin position upon writing completion

With the surrounding elements (e.g. extra printed text) removed to only contain the G code, an emulator provides the following image when attempting to draw a basic square shape. Had to be edited from main code to produce basic suitable shape – create new text file, comment out font and scaleFactor related functions to trial.




*Ben Openshaw, 20376646*

## Key Files Contained in Program

- main.c
    - Contains main program loop and initializes the hardware used, setting up serial communication with the RS232 port.
    - Config change – baud rate or loop timing, expand textHold array if required for larger file sizes
- serial.c
    - Implements all key functions, including provided functions for sending/receiving data over UART, developed functions for sending G code commands from text/font data and user input scaling.
- serial.h
    - Header for serial communication functions. Contains all function prototypes and constants
    - Config change – update buffer size, define correct com port depending on USB input
- rs232.c
    - Provides low-level implementation of RS232 serial communication across different platform (in this case Windows, Linux and Mac also available)
- rs232.h
    - Header file declaring the RS232 API functions, including platform-specific headers
    - Update RS232_PORTNR if more ports required (should only be 16 in most cases)

## Key Developed Functions and Descriptions

**int openText(const char* filePath, char* textHold, int maxLength)**

- **Description**: Opens a text file and loads its contents into a buffer for processing.
- **Parameters**:
    - **filePath** (const char*, input): Path to the text file.
    - **textHold** (char*, output): Buffer to hold the file contents.
    - **maxLength** (int, input): Maximum number of characters to read.
- **Return**: int — number of characters successfully read, or error code.

**int fetchFont(int asciiValue, FontChar* fontSet)**

- **Description**: Retrieves font data for a given ASCII character.
- **Parameters**:
    - **asciiValue** (int, input): ASCII code of the character.
    - **fontSet** (FontChar*, output): Structure holding font vector data.
- **Return**: int — success/failure status.

**int applyScaleFactor(FontChar* fontSet, float scaleFactor)**

- **Description**: Scales font vector data to adjust character size.
- **Parameters**:
    - **fontSet** (FontChar*, input/output): Font data to be scaled.
    - **scaleFactor** (float, input): Scaling multiplier.
- **Return**: int — success/failure status.

**int newPosition(float* currentX, float* currentY, float *x, float *y, int numberMovements)**

- **Description**: Updates current drawing position based on movement vectors.
- **Parameters**:
    - **currentX** (float*, input/output): Current X coordinate.
    - **currentY** (float*, input/output): Current Y coordinate.
    - **x** (float*, input): Array of X movements.
    - **y** (float*, input): Array of Y movements.
    - **numberMovements** (int, input): Number of movements to apply.
- **Return**: int — success/failure status.

**int checkWidth(float currentX, int maxWidth)**

- **Description**: Checks if the current X position exceeds the maximum allowed width.
- **Parameters**:

    - **currentX** (float, input): Current X coordinate.
    - **maxWidth** (int, input): Maximum permitted width.
- **Return**: int —0 = fits, 1 = exceeds

**int applyLineBreak(float* currentX, float* currentY)**

- **Description**: Moves drawing position to the start of a new line.
- **Parameters**:
    - **currentX** (float*, output): Reset X coordinate.
    - **currentY** (float*, input/output): Updated Y coordinate for new line.
- **Return**: int — success/failure status.

**int generateGcode(int asciiValue, float *x, float *y, int *penState, int numberMovements, int lastPenState)**

- **Description**: Generates G-code instructions for a character.
- **Parameters**:
    - **asciiValue** (int, input): ASCII code of the character.
    - **x** (float*, output): X movement array.
    - **y** (float*, output): Y movement array.
    - **penState** (int*, output): Pen up/down states.
    - **numberMovements** (int, input): Number of movements to generate.
    - **lastPenState** (int, input): Previous pen state for continuity.
- **Return**: int — success/failure status

**int applyCharacterSpacing(float* currentX, float charSpacing, float wordSpacing, int asciiValue)**

- **Description**: Adjusts X position for spacing between characters and words.
- **Parameters**:
    - **currentX** (float*, input/output): Current X coordinate.
    - **charSpacing** (float, input): Space between characters.

    - **wordSpacing** (float, input): Space between words.
    - **asciiValue** (int, input): ASCII code of current character.
- **Return**: int — success/failure status.

**int penStateOrigin(float* currentX, float* currentY, int* penState)**

- **Description**: Resets pen state and drawing position to origin.
- **Parameters**:
    - **currentX** (float*, output): Reset X coordinate.
    - **currentY** (float*, output): Reset Y coordinate.
    - **penState** (int*, output): Reset pen state (e.g., pen up).
- **Return**: int — success/failure status.

**int checkWordFit(const char *textHold, int startIndex, float currentX, int maxWidth)**

- **Description**: Checks if the next word fits within the remaining line width.
- **Parameters**:
    - **textHold** (const char*, input): Text buffer.
    - **startIndex** (int, input): Index of word start in buffer.
    - **currentX** (float, input): Current X coordinate.
    - **maxWidth** (int, input): Maximum permitted width.
- **Return**: int — 0 = fits, 1 = exceeds

# Key Data Items

| Name | Data type | Rationale |
|---|---|---|
| asciiValue | int | Required for storing characters' ASCII values |
| numberMovements | int | Number of steps taken for the drawing process |

| | | |
|---|---|---|
| x | float* | Dynamic allocation of array of x position |
| y | float* | Dynamic allocation of array of y position |
| penState | int* | Sets pen position – 0 for up, 1 for down (as pen should be up when starting). Int* with dynamically allocated array to allow for variable length and code flexibility. |
| currentX | float | Current x position as decimal |
| currentY | float | Current y position as decimal |
| FontChar | struct | Encapsulates font vector data, e.g. coords and pen state. struct chosen to logically group related fields. |
| textHold[] | char* | Stores the current character being processed from the input text file. Dynamic allocation allowing flexibility in buffer size without wasting memory. |
| textHeightmm | int | Sets text height in mm according to the user input |
| scaleFactor | float | Once height input is set, font units are scaled to a height between 4-10mm using scaleFactor |
| maxWidth | int | Allows maximum width of writing space of 100mm to be set |
| charSpacing | float | Sets horizontal spacing between characters at 2.0mm |
| wordSpacing | float | Sets greater spacing between complete words when program identifies ASCII 32 (space) |

# Testing Information

The following testing methods can be used to validate the code throughout and ensure all functions are working correctly.

| Function | Test Case | Test Data | Expected Output |
|---|---|---|---|
| openText | Checking file is opened correctly, no errors in opening | File containing “HELLO WORLD” | textHold[] set to hold characters “H”, “E”, “L”, “L”, “O”, “ “, “W”, “O”, “R”, “L”, “D”.<br><br>If error present, returns 0 and textHold[] is empty<br><br>If TestData.txt is empty, program completes without drawing. |

| | | | |
|---|---|---|---|
| fetchFont | Check font data for character | asciiValue = 65 (A) | numberMovements, x[], y[], penState[] correctly populated |
| applyScaleFactor | Check scaling factor application | scaleFactor = 2 | x[] and y[] coordinates all doubled |
| newPosition | Check pen position updates correctly | currentX = 0, currentY = 0 | currentX and currentY update to final position |
| checkWidth | Check line break is triggered only when needed | currentX >= maxwidth<br>maxwidth = 100mm | Return 1 if line break required<br>Else return 0 |
| applyLineBreak | Check line break application | currentX = 110<br>currentY = 10 | currentX becomes 0, currentY increases by 5 |
| generateGcode | Check G-code generation | asciiValue = 65 (A) with date filled | Produced G-code commands set pen up and down positions, correct X and Y movements for A. |
| applyCharacterSpacing | Check correct spacing between characters and words | Text “HELLO WORLD” | Correct spacing of 2mm between characters, and 5mm between “HELLO” and “WORLD”. |
| penStateOrigin | Check that the pen returns to the origin point at completion | currentX = 10, currentY=10, penState=1. Test with empty .txt file. | Pen should return to currentX = 0, currentY = 0, penState = 0. |
| checkWordFit | Check if word fits on the current line | “HELLO” at currentX = 90, maxWidth = 100 | Return 0, line break must be applied |
| checkWordFit | “ | “HELLO” at currentX = 10, maxWidth = 100 | Return 1, no line break needed |
| main() | Check SingleStrokeFont.txt opens correctly | No SingleStrokeFont.txt present | Program error, terminates. |
| main() | Check correct beginning-to-end function | File “HELLO WORLD”, set textHeightmm = 6 | End product shows HELLO WORLD with correct spacing between letters, fitting within 100mm width, line breaks where required, returns to 0,0 with pen in “up” position. |