Manav Pandya ID-20414930

Robot writer software project

Software Description

The software is designed to control a robotic writing arm that can transform digital text into physical writing. It does this by converting input text into a series of G-code commands that guide a robotic arm's movements. The software's primary objective is to read text from an input file and recreate it with a single-stroke font, allowing for precise and consistent text reproduction.

The script starts with the user is prompt to specify the desired text height, which can range between 4 and 10 millimeters. This input is scaled ensuring the text maintains consistent proportions regardless of the chosen size. Once the height is set, the program loads a predefined font file containing stroke information for each ASCII character, creating a mapping of how each letter should be drawn.

Then the text processing It reads the input text file character by character, managing line breaks and word wrapping within a maximum width of 100 millimeters. For each character, the software retrieves the corresponding stroke data, applies the user-specified scaling, and generates G-code commands that instruct the robotic arm how to move its pen. This process involves calculating exact X and Y coordinates, managing pen up and down states, and ensuring smooth transitions between characters and words.

Communication with the robotic arm is handled through a serial communication protocol, via the RS-232 library. The software can operate in two modes: a simulator mode for testing and debugging, and a direct robot control mode. The communication in the main section of the script includes sending initialisation commands to wake up the robot, waiting for acknowledgments, and managing the robot's drawing sequence.

The software's has six functions handling font loading, user interaction, text processing, and robot communication. A key data structure, the FontCharacter struct, stores important information about each character's drawing requirements, including ASCII code, number of strokes, and detailed stroke data.

Project Files

serial.h

The serial.h file declares functions and constants for RS-232 serial communication. It defines macros for configuring the COM port (cport_nr) and baud rate (bdrate). Key functions include PrintBuffer for sending data, WaitForReply and WaitForDollar for handling acknowledgments and synchronisation, CanRS232PortBeOpened to verify port availability, and CloseRS232Port to safely close the connection. It serves as the interface for serial communication functionality.

serial.c

The serial.c file implements the functions in serial.h to manage RS-232 communication. It includes logic for opening and closing the port, sending and receiving data, and handling specific conditions like synchronisation ($) and acknowledgments. Conditional compilation (#ifdef Serial_Mode) allows switching between real hardware and debugging modes. This file ensures robust communication between the robot and the controlling system.

SingleStrokeFont.txt

The SingleStrokeFont.txt file provides the stroke data for single-stroke fonts, used to generate G-code for robotic writing. Each character is defined by a 999 marker, an ASCII code, the number of strokes, and the stroke data, which includes X and Y coordinates and pen state (up or down). It is essential for converting text into G-code instructions.

rs232.h

The rs232.h file declares low-level serial communication functions for RS-232. It provides the foundation for opening, closing, and interacting with the COM port, which serial.h and serial.c build upon.

rs232.c

The rs232.c file implements the functions from rs232.h. It manages low-level interactions with the RS-232 port, including sending and receiving data. It enables the higher-level communication logic used in the robot writer project.

Key Data Items

Name	Data type	Rationale
textHeight	double	Stores the height of the text input by the user in millimeters. A `double` is used because text height is measured in decimal values (e.g., 4.5 mm), requiring floating-point precision.
scaleFactor	double	Holds the calculated scale factor to adjust font size based on `textHeight`. A `double` is used as the scaling requires high precision for proportional calculations.
buffer	char[100]	Stores commands to be sent to the robot or for debugging purposes. A `char[100]` array is used to accommodate text and G-code strings up to 100 characters.
fontArray	FontCharacter*	Pointer to an array of `FontCharacter` structs containing font data. A pointer is used to handle dynamic allocation for potentially large data sets.
characterCount	int	Tracks the number of characters loaded from the font file. An `int` is used because it represents whole numbers, and the count will not require decimal precision.
word	char[100]	Temporary buffer for storing words read from the input text file. A `char[100]` array is suitable for storing individual words, which typically do not exceed 100 characters.
xPos	double	Tracks the X-coordinate for G-code positioning. A `double` is used to allow fractional movements, ensuring high precision in robotic arm positioning.
yPos	double	Tracks the Y-coordinate for G-code positioning. Like `xPos`, a `double` is necessary for precise decimal-based movement values.
state	enum ProcessingState	Keeps track of the current state in the text processing state machine. The `enum ProcessingState` type is ideal for defining specific named states (e.g., `SEEKING_WORD`).

charData	FontCharacter*	Pointer to the data of the current character being processed. A `FontCharacter*` pointer is used to access the character data dynamically without duplicating it.
strokeData	int (*strokeData)[3]	Holds the stroke data (coordinates and pen state) for each character. The `int (*strokeData)[3]` allows dynamic memory allocation for 2D arrays of integers (x, y, pen state).
wordWidth	double	Calculates the total width of a word for positioning and wrapping logic. A `double` is used to ensure precise scaling word dimensions.
ch	int	Holds the ASCII value of the current character being read from the file. An `int` is appropriate because it can represent the full range of ASCII values (0–127).

Functions

Function 1

Short description: Prompts the user to input the desired text height in millimeters and validates it.

*double **promptTextHeight()***

Parameters:

None

Return value:

Returns a double representing the validated text height (between 4 and 10 mm).

Function 2

Short description: Computes the scaling factor for the text height.

*double **computeScaleFactor(double textHeight)***

Parameters:

textHeight – The user-defined text height (input in millimeters).

Return value:

Returns a double representing the scale factor.

Function 3

Short description: Sends G-code commands to the robot through the serial interface.

*void **SendCommands(char *buffer)***

Parameters:

buffer – A string containing the G-code commands to be sent.

Return value:

None

Function 4

Short description: Loads font data from a file into memory and stores it in a dynamic array of FontCharacter structs.

FontCharacter* loadFont(const char *filename, int *characterCount)

Parameters:

filename – The name of the font file to load.

characterCount – Pointer to an int to store the number of characters loaded.

Return value:

Returns a pointer to the dynamically allocated FontCharacter array, or NULL if loading fails.

Function 5

Short description: Converts input text into G-code instructions for the robot to draw the text.

void convertTextToGCode(const char *filename, FontCharacter *fontArray, int characterCount, double scaleFactor)

Parameters:

filename – The name of the text file to process.

fontArray – The array of FontCharacter structs containing font data.

characterCount – The number of characters in the font array.

scaleFactor – The scaling factor for the text height.

Return value:

None

Function 6

Short description: Prints G-code commands to the terminal for debugging purposes.

void printGCodeLine(char *buffer)

Parameters:

buffer – A string containing the G-code command to print.

Return value:

None

Testing Information

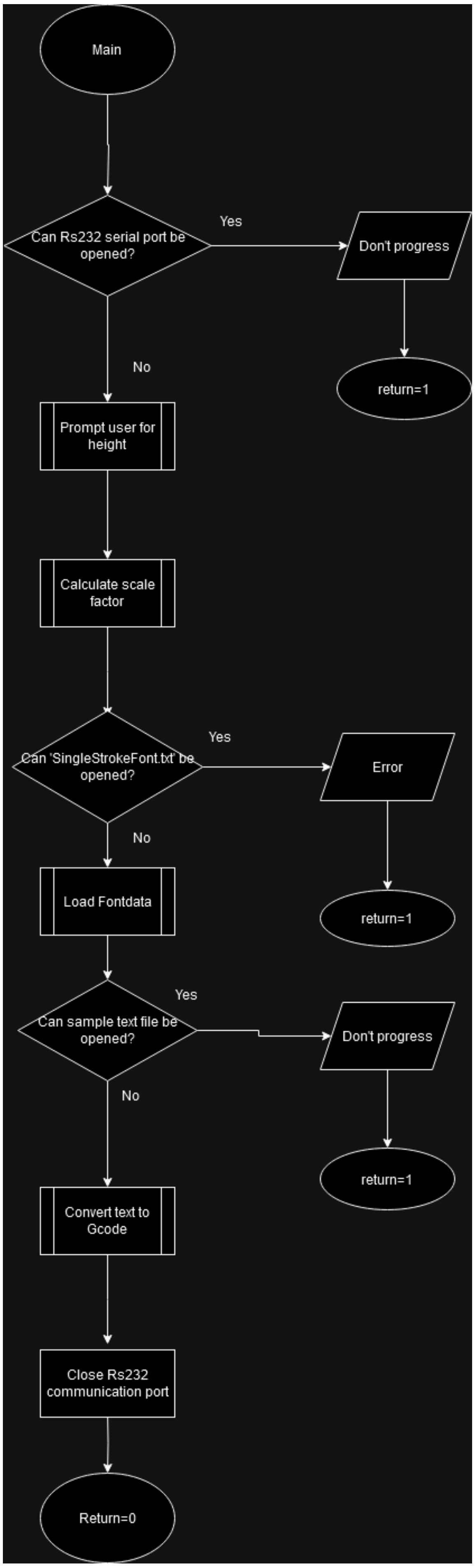
Function	Test Case	Test Data	Expected Output
promptTextHeight()	Validate that the user input is within the valid range (4–10 mm)	6.5	Function returns 6.5.
	Test input below the valid range	3.0	Error message: "Text height must be between 4 and 10 mm."
	Test invalid (non-numeric) input	"abc"	Error message: "Please enter a numeric value."
computeScaleFactor	Verify scale factor calculation based on valid text height	6.5	Function returns 0.3611 (calculated as 6.5 / 18.0).
	Test boundary values for text height	4.0	Function returns 0.2222 (calculated as 4.0 / 18.0).
SendCommands	Test if a valid G-code command is sent to the robot	"G0 X10 Y10\n"	Command "G0 X10 Y10\n" is sent successfully.
	Test with an invalid command	""	No data is sent, and no errors occur.
loadFont	Validate successful loading of font data from a valid file	"SingleStrokeFont.txt"	Returns a valid FontCharacter* array with characterCount > 0.
	Test with an invalid or non-existent font file	"invalidFont.txt"	Function returns NULL and prints error message.
convertTextToGCode	Check if valid text is correctly converted to G-code commands	"RobotTesting.txt" with valid font data	G-code commands are generated and sent to the robot.
	Test with unsupported characters in the text	"InvalidCharTest.txt"	Prints an error: "Character '<char>' is not supported by the loaded font."

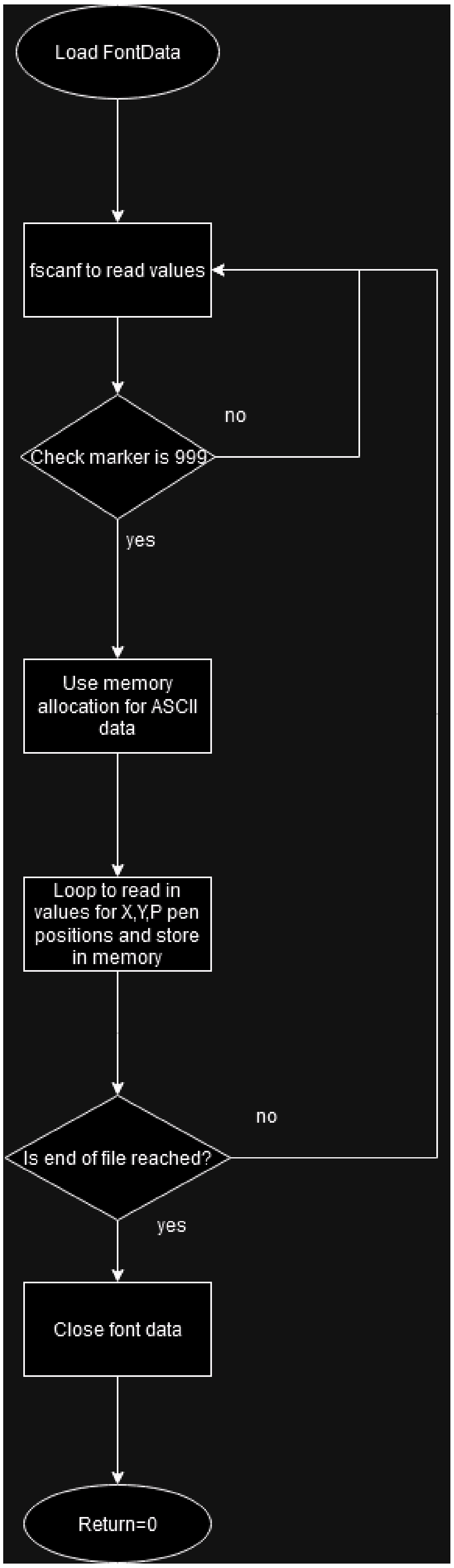
printGCodeLine	Verify that a G-code line is correctly printed to the terminal	"G1 X10 Y10\n"	Line "G1 X10 Y10\n" is printed to the terminal.

Flowchart(s)

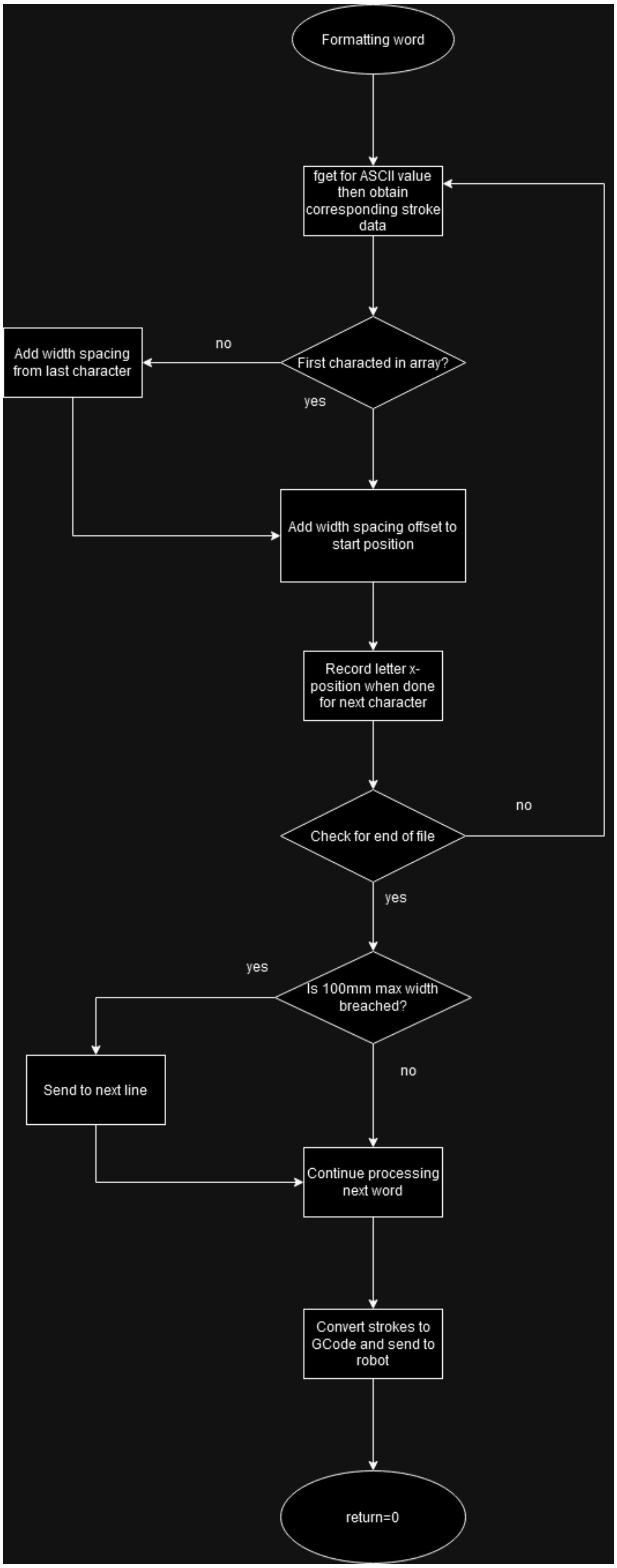
intMain():

LoadFont:





convertTextToGCode:


Formatting word
fget for ASCII value then obtain corresponding stroke data
First characted in array?
no
Add width spacing from last character
yes
Add width spacing offset to start position
Record letter x-position when done for next character
Check for end of file
no
yes
Is 100mm max width breached?
yes
Send to next line
no
Continue processing next word
Convert strokes to GCode and send to robot
return=0