# Name & Student ID

Lee Yue Shing 20517270

# Title

Developing a software which reads from a file and control the robot to draw out the text as read.

# Outline of the Problem to be Solved

(Maximum 1 page)

This project involves creating a software that allows a robot to “draw” text by reading character definitions from a specified font file. Each character can be scaled to a height and will be defined by the user, which will be between 4 and 10mm. The program should also ensure that the total width of the drawn text will not exceed 100mm. The commands will be sent to an Arduino-controlled writing robot via the virtual Rs232 serial port using G-Code, which controls the robot’s arm to “draw” out the text.

Additionally, the code should be written so that the entire font data file is read and stored in memory in an appropriate date format. Each word in the file should be processed and output to the robot before the next word is read from the file. The code can command the Arduino to raise and lower the pen, as well as move to specified X,Y locations to write the word out. The lines of text written needs to be spaced 5mm apart. Each word should not be broken up so if a word is too long to be fitted in one line it should remind the user to scale down the word. When writing the text, the code should determine if the remaining space is enough to fit the word, if not, open a new line.

# Key Data Items

|  |  |  |
| --- | --- | --- |
| Name | Data type | Rationale |
| DataRow | struct | Defines a single row of data which is 1 movement/stroke of each character |
| xPos | int | Represent the x-coordinate position of each movement of stroke. |
| yPos | int | Represent the y-coordinate position of each movement of stroke. |
| penStatus | int | Indicates the pen state, either up or down |
| dataArray | struct DataRow\* | An array to store the entire font data in after reading the file. |
| height | float | Store the user-defined height for drawing text, so the font size can be scaled. |
| scaleFactor | Float | Since base font is 18 units, it is used to scale each character’s coordinates, which is 1/18 |
| textFileName | char\* | Stores the name of the text file to be read and written |
| fontDataFile | char\* | Stores the path for the font data file |
| currentXPos | Float | Stores current x-coordinate position in the writing area, to check the next word is within bounds |
| currentYPos | Float | Stores current y-position in writing area, for making a new line after reaching the end of each row |
| MAX\_WIDTH | const int | Constant value of 100mm, which is the max allowable width of writing area. Also to determine if the next word to be written is still within boundaries or not. |
| NUM\_ROWS | const int | Represents number of rows in the font data file, for correct allocation into the dataArray. |
| wordWidth | float | Stores the calculated width of the current word to check if it can fit within the remaining line width or if a new line is needed |
| lineSpacing | float | Stores the fixed line spacing (5mm), ensuring consistent vertical distance between lines. |
| yPos | float | |  | | --- | |  |  |  | | --- | | Stores the current y-coordinate; decreases by 5mm + height when moving to a new line. | |
| wordTooLong | bool | |  | | --- | |  |  |  | | --- | | Tracks whether a word is too wide to fit on a single line; if true, prompts the user to adjust scale. | |

Extend table as required

# Function Declarations

1. **void readFile(const char \*filename, struct DataRow \*dataArray, int numRows);**
   * **Parameters:**
     + filename: The path to the font data file to be read.
     + dataArray: Pointer to an array of DataRow structs where the font data will be stored.
     + numRows: The total number of rows in the font data file.
   * **Return value**: None (void function).

**-**Reads the font data from the specified file and stores it in the dataArray structure.

1. **void displayFirstNRows(struct DataRow \*dataArray, int numRows, int N);**
   * **Parameters:**
     + dataArray: Pointer to an array of DataRow structs.
     + numRows: Total number of rows in dataArray.
     + N: The number of rows to display.
   * **Return value**: None (void function).

**-** Displays the first N rows of data from dataArray in the same format as the original font file.

1. **void scaleFont(float height, float scaleFactor);**
   * **Parameters:**
     + height: The user-defined height (in mm) for scaling the font.
     + scaleFactor: The factor used to scale each coordinate based on the user-defined height.
   * **Return value**: None (void function).

**-**Scales the X and Y coordinates in the font data based on the user-defined height.

1. **int checkValidCharacter(char character, struct DataRow \*dataArray, int numRows);**
   * **Parameters:**
     + character: The character to be checked for validity.
     + dataArray: Pointer to the array of DataRow structs containing the font data.
     + numRows: The number of rows in the font data array.
   * **Return value**: Returns 1 if the character is valid (exists in the font data), and 0 if it is invalid.

**-**Checks if the specified character exists in the font data file by comparing the ASCII value against the font data.

1. **int checkLineWidth(float xPos, float widthLimit);**
   * **Parameters:**
     + xPos: The current x-coordinate position in the drawing area.
     + widthLimit: The maximum allowed width of the drawing area.
   * **Return value**: Returns 1 if the current position is within bounds, or 0 if it exceeds the width limit.

**-**Checks if the current x-coordinate position will exceed the width limit (100 mm) and determines whether a new line should be started.

1. **void processTextFile(char \*textFileName, struct DataRow \*dataArray, int numRows, float scaleFactor, float widthLimit);**

**Parameters:**

* + - textFileName: The name of the text file containing the text to be drawn.
    - dataArray: The array containing the font data.
    - numRows: The total number of rows in dataArray.
    - scaleFactor: The scaling factor to adjust the font size.
    - widthLimit: The maximum width limit of the writing area.

**Return value**: None (void function).

**-**Reads the text file, processes each word, and sends the corresponding commands to the robot to draw the text while managing line breaks.

1. **int checkWordFit(int xPos, int wordWidth);**

**Parameters**:

* + xPos: (current x-coordinate)
  + wordWidth: (calculated width of the word).

**Return Value**: Returns 1 if the word fits in the current line, otherwise 0.

**-**Checks if xPos + wordWidth exceeds MAX\_WIDTH to determine if a new line is necessary.

1. **processWord(char \*word, int \*xPos, int \*yPos, float scaleFactor);**

**Parameters:**

* + word (character array storing the word to draw)
  + xPos: (current x-coordinate
  + yPos: (current y-coordinate
  + scaleFactor: (for scaling coordinates

**Return Value**: None.

-Take a word and sequentially process each character within it, drawing each and updating xPos.

**9. updatePosition(float\* yPos, float lineSpacing, float height)**

* + yPos: current y-coordinate
  + lineSpacing: space of each line
  + height: The user-defined height (in mm) for scaling the font.

**Return Value**: None.

- Updates yPos by subtracting the sum of lineSpacing (5mm) and height to move to a new line.

1. **checkWordWidth(const char\* word)**
   * word: width of word after scaling

**Return Value**: returns true if the word can fit within MAX\_WIDTH; false otherwise

- Calculates the total width of the word at the current scaleFactor. If the word is too wide, sets wordTooLong to true.

# Testing Information

|  |  |  |  |
| --- | --- | --- | --- |
| Function | Test Case | Test Data | Expected Output |
| main | Program initialization and setup | Valid font data file and file path | Program initializes successfully; |
|  | Error opening files | Invalid font data file | Error message displayed: "Error opening font data file." |
| readFile | Successful file read | Valid font data file | Font data loaded into dataArray correctly, NUM\_ROWS rows read. |
|  | Error handling for missing font file | Invalid file path | |  | | --- | |  |  |  | | --- | | Error message: "Error opening file." | |
| displayCharacterData | Character data retrieval | ASCII value of 'A' | Outputs xPos, yPos, penStatus for character strokes. |
|  | |  | | --- | |  |  |  | | --- | | Invalid character input | | Character with ASCII value out of bounds | Error message: "Character not available in font data." |
| scaleFont() | Scaling factor calculation | height = 5 mm | scaleFactor set to 5 / 18 = 0.27778. |
| processWord | Word within line bounds | "Hi" in 100mm width area | Draws word; xPos updated correctly without moving to next line. |
|  | Word exceeds line width, needs new line | "HelloWorld" exceeding remaining space in line | yPos updated; xPos reset to 0 for new line. Check the logic to start a new line is correct. |
|  | Non-existent character in word | "Hello£" (where £ is not in font data) | Error message for unsupported character. |
| calculateWordWidth | Word width calculation | "Hello" | Width of "Hello" calculated as the sum of individual letter widths.  Ensures the calculation logic is accurate. |
|  | Fit check for word within current line | "Hello" with xPos = 80 | Returns false, indicating word does not fit; moves to new line. |
| updatePosition | Coordinate update for new line |  | yPos decreases by scaled line height; xPos set to 0. |
| isCharacterInFontData | |  | | --- | |  |   Check if character is valid | Character = 'A' | Returns true for 'A' (exists in font data). |
|  | |  | | --- | |  |  |  | | --- | | Invalid character check | | Character = '!' | |  | | --- | |  |  |  | | --- | | Returns false for '!' (if not in font data); triggers error message. | |
| updatePosition | New line start with line spacing | yPos = 0, lineSpacing = 5, height = 5 | yPos updates to -10, reflecting line height + spacing. |
| processWord | Word is too wide for one line | "LongWordTest", scaleFactor = 1 | Error message prompts user to scale down |
|  | Word fits within line width | "Short", scaleFactor = 1 | |  | | --- | |  |  |  | | --- | | Word processes normally; xPos updated for next word | |
| checkWordWidth | Word width exceeds MAX\_WIDTH | "ExceedinglyLongWord", scaleFactor = 1 | Returns false, sets wordTooLong to true |
|  | Word width is within MAX\_WIDTH | "Fit", scaleFactor = 1 | Returns true, no prompt needed |

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

# Flowchart(s)

May be included as separate pdf