Stefano Geymonat

Student ID: 20465966

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

# Outline of the Problem to be Solved

The program is designed to control a robot that “writes” text by reading and converting each character from a user-specified text file into precise movement commands. First, the program reads a file called SingleStrokeFont.txt, which contains the necessary coordinate and pen-up/pen-down data for each ASCII character. This font data is loaded into an array, which allows the program to quickly find and access the details for each letter’s drawing sequence. The user specifies the desired text height as a floating-point number between 4 and 10, and the program calculates a scaling factor based on this height. This scaling factor will be applied to each character’s coordinates so that the text fits within the user-defined range.

When the program begins processing the text file, it reads each character in sequence, retrieving its ASCII value and then locating the corresponding entry in the font array. The third value in each character’s data structure specifies the number of movements required to form the letter, and the next set of lines provide the precise movement commands needed for the robot to draw it. This data chunk is then transferred to a temporary array known as the “word array.” The program continues reading characters and adding each one’s movement data to the word array until it encounters a space, which signals the end of a word.

When a word ends, the program multiplies all coordinates in the word array by the previously calculated scaling factor, which adjusts each movement to the specified text height. After scaling, it checks if the word will fit within the robot’s 100mm wide writing area. To do this, it calculates the remaining space on the line by subtracting the total width of the word and the final X coordinate of the previous character from 95mm. If the result is positive, the word fits within the line; otherwise, the program moves the robot’s arm down 5mm and back to x = 0, creating a new line before proceeding with the next word. Each time the word array is scaled and positioned correctly, it is converted into G-Code commands, which tell the robot’s arm how to position itself and when to raise or lower the pen.

The program transmits each G-Code command to the robot via a serial connection. After sending each command, it waits for an “ok” response from the robot before proceeding with the next one, ensuring synchronization. This process of reading characters, scaling and positioning each word, and generating G-Code continues until the end of the user’s text file. Once all words have been processed, the program ensures that the pen is raised, and the robot’s arm is returned to the origin (0,0) position.

# Key Data Items

|  |  |  |
| --- | --- | --- |
| Name | Data type | Rationale |
| Single stroke array | array | Using an array to store data from a text file in C is efficient and simple. Arrays let you quickly access any element by index, making it easy to process each part of the data. Since arrays are fixed in memory, they’re predictable and work well with C’s standard functions for strings and data handling. |
| Font size | float | Can be any value between 4 – 10 including decimal values. |
| Space remaining | float | As the font size is a float this can lead to the space remaining being having a decimal value. |
| Word width | float | As the font size is a float, this means that the word width must be allowed to be a float. |
| Scale factor | float | As the equation is user font size/18, this is a fraction which requires the use of a float. |
| SingleSrokeFont.txt | File | this will be declared as a pointer of type FILE so we can work with it. |
| ASCII | int | ASCII values vary from 0-255 as whole numbers, so they are always integers. |
| Arm movements | structs | In each row we have structs that represent the X,Y,Z position of arm. |
| G - code | Array of strings | Each command line is becomes a string when we convert to G-code. |
| character | char | For Letters in C they can be stored as char. |
| Number of arm movements | int | The number next to the ascii value is a whole number that says how many individual arm movements there needs to be for the letter/symbol to be written. |
| Word array | Array of structs | Use of the array here is so we can represent all the collection of arm movements for the entire word in one array. In each row we have structs that represent the X,Y,Z position of arm. |
| Stroke coordinates | structs | The structs have 3 members: X,Y,Z(pen up or pen down), this is the simplest way to collect these values to them use/move them around. |

# Function Declarations

**check\_file\_open:**

int check\_file\_open(const char\* filename);

**Parameters**:  
filename – The name of the file to open given by user.

**Return value**:  
Returns 1 if the file opened successfully, 0 if it failed.

**end\_of\_file**

int end\_of\_file(FILE\* file);

**Parameters**:  
file – Pointer to the file being read.

**Return value**:  
Returns 1 if the end of file (EOF) is reached, 0 otherwise.

**get\_ascii\_value**

int get\_ascii\_value(char character);

**Parameters**:  
character – The character to convert, read from file.

**Return value**:  
Returns the ASCII value of the character.

**word\_width**

float word\_width(const float\* word\_array);

**Parameters**:  
word\_array – Array of character movements for a word.

**Return value**:  
Returns the width of the word in units, calculated from x movements and offsets.

**space\_remaining**

float space\_remaining(float previous\_word\_end);

**Parameters**:  
previous\_word\_end – The largest x position of the previous word.

**Return value**:  
Returns the available space remaining on the line.

**check\_word\_fits**

int check\_word\_fits(float space\_remaining, float word\_width);

**Parameters**:  
space\_remaining – Remaining space on the line.  
word\_width – Width of the word to fit.

**Return value**:  
Returns 1 if the word fits within the remaining space, 0 otherwise.

**is\_char\_space**

int is\_char\_space(char character);

**Parameters**:  
character – The character to check.

**Return value**:  
Returns 1 if the character is a space (ASCII 32), 0 otherwise.

**get\_scale\_factor**

float get\_scale\_factor();

**Parameters**:  
None.

**Return value**:  
Returns a scale factor between 4 and 10, as specified by the user.

**robot\_arm\_letter\_inputs**

float robot\_arm\_letter\_inputs(int ascii\_value, float\* word\_array);

**Parameters**:  
ascii\_value – ASCII value of the character to map.  
word\_array – Array to store corresponding robot arm movements.

**Return value**:  
x number of structs representing arm movements

# Testing Information

|  |  |  |  |
| --- | --- | --- | --- |
| Function | Test Case | Test Data | Expected Output |
| **get\_ascii\_value** | Check to see if correct ascii value is provided | A | 65 |
| **check\_file\_open** | Give a text file to open, see if it is done successfully | Viable filename | 1 |
| **end\_of\_file** | Once at end of file check if code can detect this | Run a loop until end of file | 0  0  0  …  1 |
| **get\_scale\_factor** | Check if user gives invalid | 11 | That is not between 4-10 |
| **Robot\_arm\_inputs** | Check if the correct inputs are given for the corresponding character | A | 0 0 0  6 18 1  12 0 1  3 9 0  9 9 1  18 0 0 |
| **Is\_char\_space** | Check if code can correctly check if ascii value respresnts space | 32 | 1 |
| **Check\_word\_fits** | Check if calculation is correct | Space remaining(30)-word width(35) | 0 |
| **Word\_width** | Check word width of the word HI | HI | 9 |
| **Space\_remaining** | Check for space remaining if only word HI is present on line | HI | 86 |

# Flowchart(s)

Have been added as pdfs to folder