

Software Description

This software is aimed to generate a G-code to the writing robot for writing the text which is read from a file. The G-code will be executed by the writing robot and draw the text on the paper.

The text which needs to be 'draw out' is input to the software by saving as a file. The output G-code will be sent to the control unit of the writing robot (Arduino in this case) using a virtual RS232 serial port.

In this software, some font data and open-source libraries will be used, for example, an additional font file contains the pre-defined G code for each character will be required. The RS232 serial port communication will be handled by a RS232 library.

For the serial port communication, before sending more G-code, the software will wait until receiving the acknowledge signal from the Arduino.

The software interface with user will be a console window. The communication code and possible errors during the process will be displayed on this console window.

Project Files

File Structure

- src	source code and build path, code files are directly in this folder
- include	the head files which are included
- build	store the object files after compile
- asset	
- font	store the static font g-code files
- input	the relative path for input.txt
- bin	the final output file for executable program
- docs	store the necessary documents

Head Files

utility.h	some macros to control the debug/release mode
main.h	head file for main, defined the basic functions
rs232.h	head file provided by rs232 library
serial.h	head file provided as a packed interface for rs232

Code Files

main.c	main file code, execute the necessary functions
serial.c	some functions which packed for rs232 serial connection
rs232.c	code file provided by rs232 library

Possible changes needed during further development: Some functions for load static files can be encapsulated into some functions.

Pack for release: pack the bin, asset and input folders after the compile.

Function Declarations

`int CanRS232PortBeOpened()`

Function: initial the rs232 library

Return value: always 0.

`int WaitForWakeUp(void);`

Function: pause and wait for the first reply

Return value: always 0.

`int WaitForReply(void);`

Function: pause the g-code output to wait for the receiving command from serial port

Return value: always 0.

`int convertCharArrayToInt(char numarray[], int *startPosition, int charLength, int *returnValue)`

Function: convert the char in a char array to integer

Parameters:

`char numarray[]`: file pointer of the font g code file;

`int *startPosition`: from which point of the string to convert the int number

`int charLength`: the max length of this conversion, must larger than length+1

`int *returnValue`: return the integer after this conversion

Return value: 0 for failed and 1 for success.

`returnValue` will handle the result as a pointer

`int generateFontIndex(FILE *filePointer, struct FontIndex fontGcodeLineIndex[]);`

Function: generate an index to make g-code generation much easier

Parameters:

`filePointer`: file pointer of the font g-code file.

`fontGcodeLineIndex[]`: an array stores the start line and end line number of each character G-code;

Return value: 1 for success, 0 for failed.

 Index will be handle and returned by the Pointer `fontGcodeLineIndex`.

`int initializeWritingMachine();`

Function: send the initialize g-code for writing robots.

Return value: 1 for success, 0 for failed

```
int createFontDataCache(FILE *filePointer, int fontGcodeData[]);
```

Function: convert the plain text g code to a more executable data saved in memory

Parameters:

filePointer: file pointer of the font g-code file.

fontGcodeData[]: the pointer point to the pre allocated memory for the storage;

Return value: 1 for success, 0 for failed.

Data after convert will be saved to the memory block allocated before and pointed by the int pointer fontGcodeData.

```
int generateCharGcodeCommand(int charAsciiNum, double *tempOffsetX, double *tempOffsetY, char commandBuffer[], int fontDataCache[], struct FontIndex fontIndexArray[], double Scaler);
```

Function: use the offset and scaler to adjust the font g-code for writing robot to execute. This function will be called for each character.

Parameters:

charAsciiNum: the ASCII number of current character need to be printed;

*tempOffsetX: the offset position X for this character;

*tempOffsetY: the offset position Y for this character;

commandBuffer[]: the commandBuffer of serial port, which will be send;

fontDataCache[]: the pointer for the memory block which stores the font g-code cache;

fontIndexArray[]: an array stores the start line and end line number of each character G-code;

Scaler: the global scaler for this writing task;

Return Value: 1 for success and 0 for failed.

G-code command will be saved to command buffer and send through serial port.

```
int updateCharactorOffsetPosition(double *tempOffsetX, double *tempOffsetY, double commandWidthChange, double commandHeightChange, double globalScaler);
```

Function: change the offset for next character.

Parameters:

*tempOffsetX: the offset position X needs to update;

*tempOffsetY: the offset position Y needs to update;

commandWidthChange: required move on x axis for next character's offset;

commandHeightChange: required move on y axis for next character's offset;

globalScaler: the scale set for the offset change.

Return value: 1 for success, 0 for failed.

The offset after updated will be returned through the change on the variables pointer point to.

Key Data Items

Name	Data type	Rationale
fpFont	FILE *	File pointer for font data
fontIndexArray[128]	FontIndex Self-defined struct	An array with size of 128, start line for each character is .start_line, the length of g-code is .line_num. both are int.
memForFontData	int *	Point to a block of memory assigned after the index function sort out the size required to store the g-code data for each character
fpText	FILE *	File pointer for text file
outputOffsetX	double	The x offset of current character
outputOffsetY	double	The y offset of current character
generalScaler	double	A variable to determine size scale of the output text
charReadyToWrite	char	The next char read out from the input text file, which will be ready to print by sending into a function

Test Information

Function	Test Case	Test Data	Expected Output
generateFontIndex	Load font data	Font data file path, empty array.	Return 1, with the font index loaded
updateGcodeTargetPosition	Initilaize the start of whole code	gcodeLineNum = -2, offset x and y both = 0, lastTimeReturn Value = 1;	Return -1, with the char array become 'F 1000'.
updateGcodeTargetPosition	Test the output of Char H's third line	gcodeLineNum = fontDataIndex [2*(int) 'H' + 2], offset x and y both = 0, lastTimeReturn Value = 1;	Return -1, with the char array become 'S 1000'.
updateGcodeTargetPosition	Test the output of Char H's third line after pen move down	gcodeLineNum = fontDataIndex [2*(int) 'H' + 2], offset x and y both = 0, lastTimeReturn Value = -1;	Return 1, with the char array become 'G1 X0 Y18'.
updateCharactorOffsetPosition	Move one width	tempOffsetX = 0, tempOffsetY = 0;	Return 1, tempOffsetX = _CHAR_WIDTH, tempOffsetY = 0;
updateCharactorOffsetPosition	Shift to next line	tempOffsetX = _MAX_WIDTH, tempOffsetY = 0;	Return 1, tempOffsetX = 0, tempOffsetY = _LINE_HEIGHT;

Flowcharts

