X-Y Plotter Project

Pavel Dounaev, Nhan Phan

Metropolia University of Applied Sciences

October 2018

1. Introduction and goal of the project

1.1. Introduction

The project consists of designing and implementing a software for XY-Plotter. The XY-Plotter is designed so it can to draw different shapes and forms based on picture.

1.2. Hardware

Main hardware part of the project is the Makeblock XY-plotter. The full list of the hardware used in the project is listed above.

Component name	Quantity (pcs)	Description
NXP LPC 1549	1	Microcontroller, responsible for controlling the XY-plotter system.
42BYG Stepper Motor	2	Stepper motor responsible for moving linear rail system.
RC Servo Motor SG90	1	Used for controlling pen position up and down.
3B class Laser	1	Used as drawing peripheral if needed.
Me Stepper motor driver	2	Used to control the stepper motors.
Voltage regulator circuit	1	Regulates the voltage between microcontroller and stepper motor drivers.
Linear rail system	1	To move the pen/laser across X/Y coordinates

1.3. Goals

The main goal was to make a functional XY-plotter with LPC1549 microcontroller. The communication between mDraw and LPC1549 needed to be flawless. LPC needed to receive the G-code commands and respond to them accordingly.

Another requirement for LPC was to be able to drive stepper motors smoothly and with high accuracy. The drawn picture had to be smooth, even at diagonal line and without any missed line.

2. Implementation

2.1. Hardware implementations

2.1.1. Stepper Motor

Stepper motor is used to move the plotter rails. LPC 1549 microcontroller controls the stepper motor via A4988 microstepping driver module. The A4988 driver module provides simple interface for microstepping the stepper motor. Driver controls the stepper motor using two pins: DIR and STEP. DIR pin is used for direction and STEP pin is used for stepping. Driver is controlled with two pulses, HIGH and LOW. To drive the motor, driver's DIR pin receives one HIGH pulse and one LOW pulse or vice versa, so one step contains two pulses. Pulses need to be same width and within specified range in order driver to drive the stepper motor to step. Maximum pulse width is 500μs and minimum width is 45μs. The smaller the pulse width the faster motor will step.

To achieve these pulses microcontroller's RIT is used. RIT is an interrupt which interfaces with DIR and STEP pins. When RIT interrupt occurs, it's interrupt handler sends pulses to the A4988 driver.

2.1.2. RC Servo motor and Laser

RC servo motor is used to control the pen's position. Servo controls pen's movement diagonally, so pen can move in two directions: up and down. Servo motor is controlled by PWM(*Pulse With Modulation*) signal. To generate PWM signal, LPC 1549 microcontroller's SCT timer is used. Timer is configured so it would generate signals at certain frequencies. In order motor to work the period(time between two signal pulses) should be 20ms, which means that frequency should be set to 50Hz and pulse width should be between 1 - 2ms. Pen's position is changed using `void penMove(int penPos)` function, which changes frequency of the PWM signal, which is controlled by timer's registers.

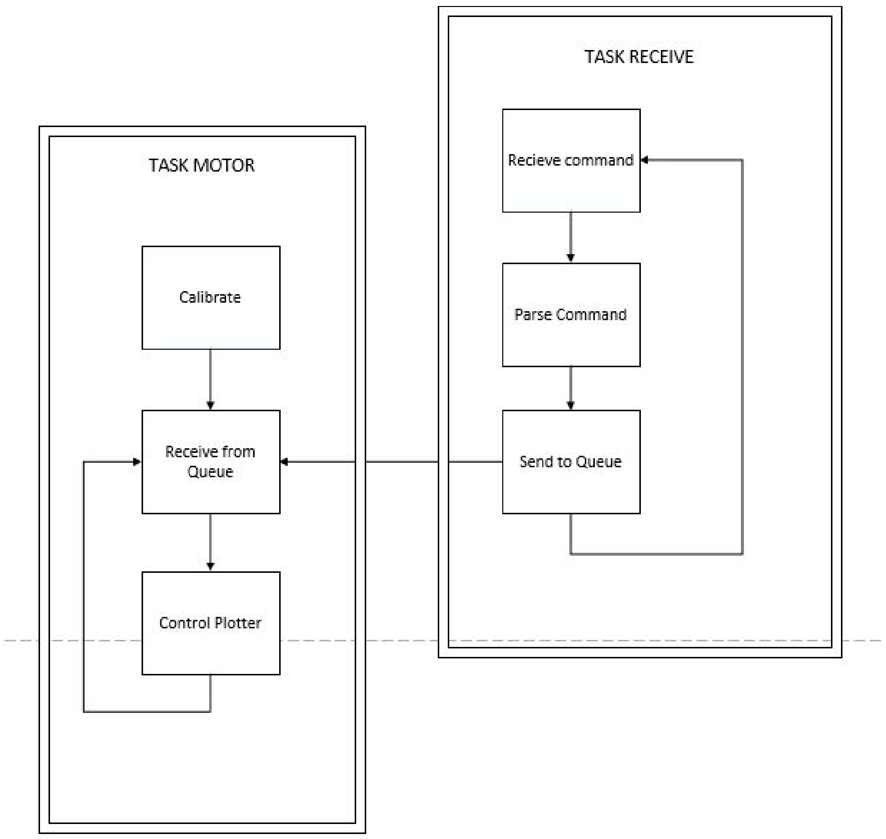
Laser works in similar manner as servo motor. It receives PWM signal to control the power of the laser. Laser also has own SCT timer in order to change the frequency of the signal and therefore control the laser power.

2.2. Software Implementation

2.2.1. Overview

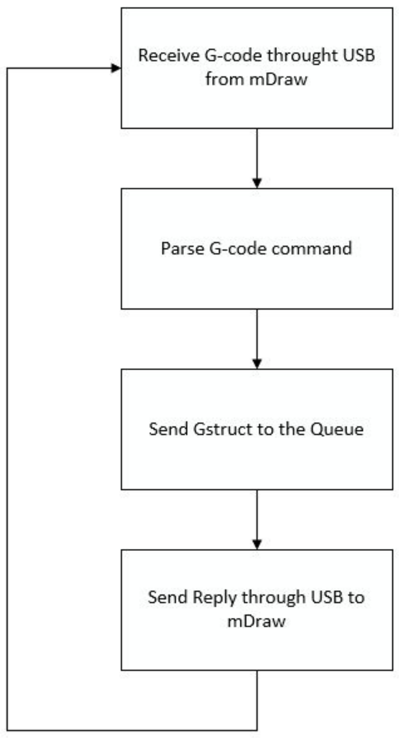
Software is implemented using real time operating system FreeRTOS. FreeRTOS provides simple interface for managing the system, such as creating and managing operating system tasks(processes), using and protecting system resources and also provided synchronization and interprocess communication between tasks.

Main program is divided into two tasks(processes): TaskReceive and TaskMotor.



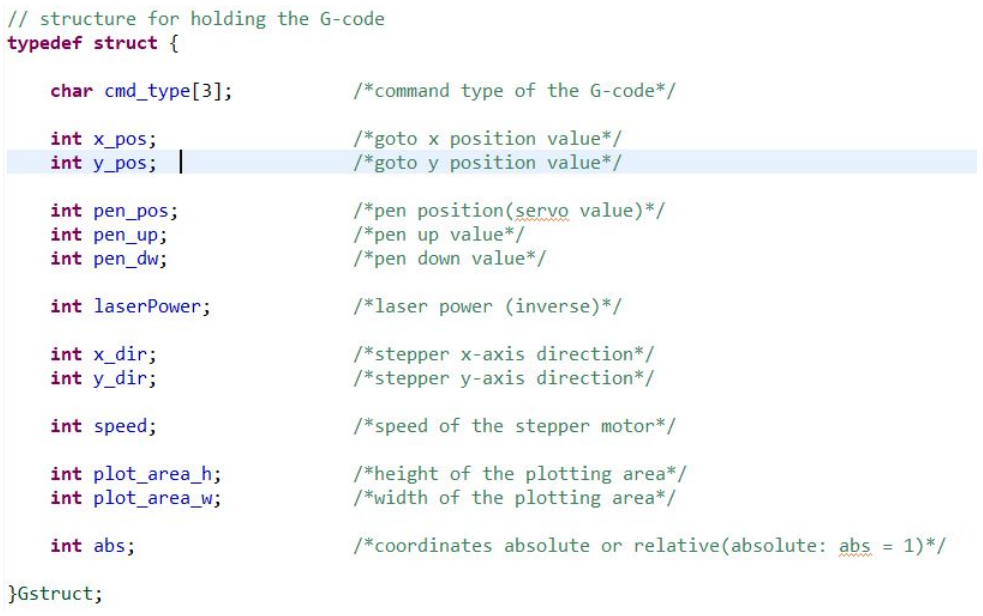
2.2.2. TaskReceive

TaskReceive's job is to communicate with mDraw program. Task receives and parsers incoming stream of G-code commands sent by mDraw. Task receives commands through USB protocol. When the command is received by task, it is parsed and corresponding command is sent back to mDraw. That acknowledges mDraw that G-code command was successfully received and task is ready to receive the next one. G-code command parsing was achieved by creating and using Parser class.


Receive G-code throught USB from mDraw
Parse G-code command
Send Gstruct to the Queue
Send Reply through USB to mDraw

2.2.2.1. Parser

Parsers job is to parse G-code commands. Parser parsers G-code by calling `Gstruct parseGcode(const char* buffer, int pen_up, int pen_dw)` function. G-code commands are parsed into `Gstruct`, a C structure that holds information about received instruction, for example in which position plotter needs to move. That information is stored inside variables. When the command is parsed into structure, that structure is then send to the queue(`cmdQueue`) for *TaskMotor* task to receive.



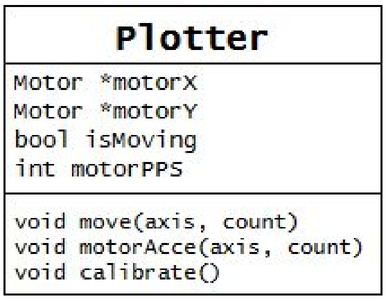
```
// structure for holding the G-code
typedef struct {

    char cmd_type[3];           /*command type of the G-code*/

    int x_pos;                  /*goto x position value*/
    int y_pos;                  /*goto y position value*/

    int pen_pos;                /*pen position(servo value)*/
    int pen_up;                 /*pen up value*/
    int pen_dw;                 /*pen down value*/

    int laserPower;             /*laser power (inverse)*/

    int x_dir;                  /*stepper x-axis direction*/
    int y_dir;                  /*stepper y-axis direction*/

    int speed;                  /*speed of the stepper motor*/

    int plot_area_h;            /*height of the plotting area*/
    int plot_area_w;            /*width of the plotting area*/

    int abs;                    /*coordinates absolute or relative(absolute: abs = 1)*/

}Gstruct;
```

2.2.3. TaskMotor

The purpose of TaskMotor is to control the stepper motors to draw a specified picture. Task receives the `Gstruct` from the queue(`cmdQueue`) that was previously sent by TaskReceive task. Then task controls the motors based on the information provided by `Gstruct`. Task also handles improvements for drawing picture, including acceleration for motors and Bresenham's line drawing algorithm. Drawing tool(pen or Laser) control is also handled inside the task.

2.2.3.1. Plotter class

Plotter class includes two motors for axis X and for Y axis. It also contains important information as the the pulses per second, is the plotter moving or drawing (motor will move faster if plotter is not drawing).



```
Plotter
Motor *motorX
Motor *motorY
bool isMoving
int motorPPS
void move(axis, count)
void motorAcce(axis, count)
void calibrate()
```

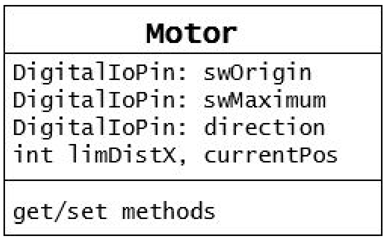
Plotter class also contains calibration function, which calibrates two motors to get the correct step between their limits.

Overall, to draw using X-Y plotter, plotter class is needed. It handles motors and controls pen and laser. Motor class is also needed, which contains information for the motor.

2.2.3.2. Motor class

Motor class contains information of the motor, such as distance of between two limit switches, measured by stepper motor steps. The Motor class also has functions for accessing that information. Motor class has only getter/setter functions, since all the movement are handled inside the Plotter class.

2.2.3.3. Calibration



```
Motor
DigitalIoPin: swOrigin
DigitalIoPin: swMaximum
DigitalIoPin: direction
int limDistX, currentPos
get/set methods
```

Plotter has calibration function to measure the actual motor steps between limit switches (original point is 0 and maximum point is where the limit switch gets hit). First it measures the total distance in steps on X axis, then total distance on Y axis. Since the actual steps of both axis are always larger than 26,000 steps, when the measured steps are less than 24,000, motor can also accelerate to move faster. When it reaches 24,000 steps, motor decelerates and stops when it hits one of the limit switches.

Acceleration and deceleration speeds up the calibration process and also keeps the accuracy the same. Without acceleration, motor can still reach the maximum speed (8,000~10,000 PPS), but the sudden movement from 0 to 10,000PPS can make the whole plotter vibrate and in the process plotter may miss some steps while moving. Without deceleration, there is possibility that when the motor hits limit switch, it will not stop fast enough and can hit the railing, resulting into possible damage for the plotter. Correct deceleration prevents this from happening.

2.2.3.4. Drawing

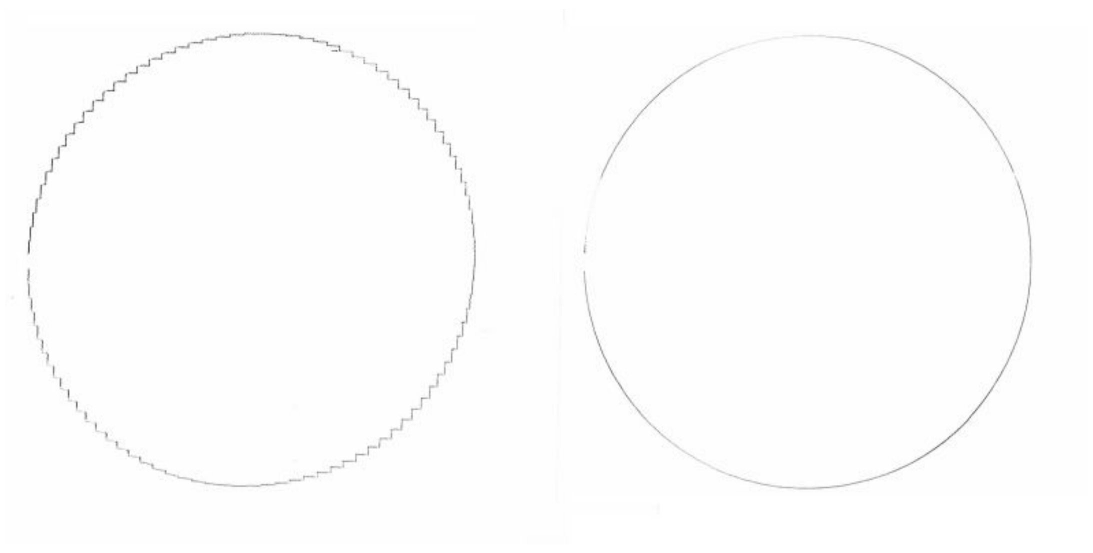
When the plotter is calibrated, it starts reading the queue(`cmdQueue`) waiting for the `Gstruct` that is send by *TaskReceive*. When task finally recieves `Gstruct` from the queue, it is ready to start plotting. When it receives the `G1` command, it first converts the X coordinate and Y coordinate from millimeter to motor steps. Then using motor steps to control the actual motors.

2.2.4. Improvements

To make the drawing smoother and faster, couple of improvements are needed.

2.2.4.1. Bresenham

Normal drawing with coordinates from mDraw is not smooth enough when drawing curves, and plotter will draw small zig-zag straight lines instead. A circle drawing using normal method would create many small straight lines, each can have up to 300 mm length. To improve the drawing quality, Bresenham algorithm is implemented. The algorithm divides the 300mm straight line further to about 260 steps, each step drawing a small straight line, about 1.15mm. It's still not a perfect curve, but with the smaller resolution, overall picture will look smooth. As long as the pen tip or laser is bigger than 1.15mm, then we will not see any straight lines.

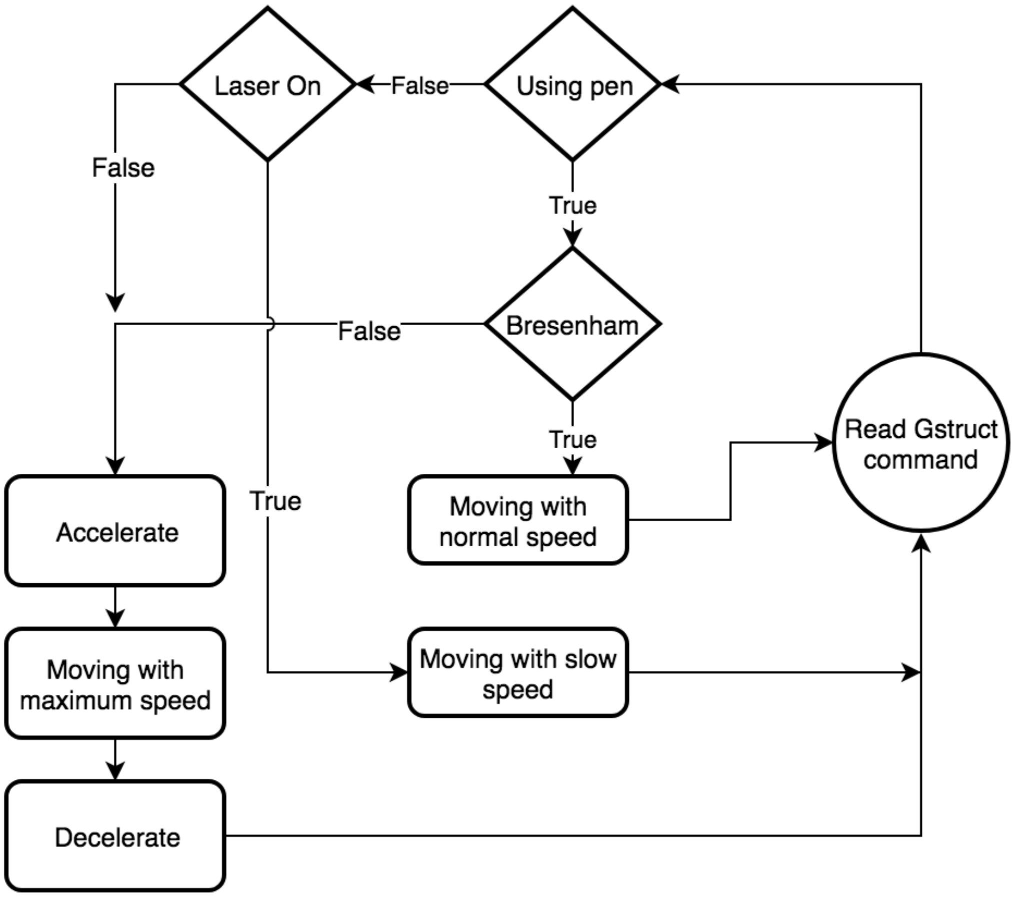


Picture drew with normal method (left) and drew with Bresenham's algorithm

Further improvement is not depend on the software, as 1 step is the minimum step the X-Y plotter can move. The only solution is using a better motor with bigger steps per mm.

2.2.4.2. Acceleration/Deceleration

The way motor accelerates and then decelerates when drawing is different compared with when it accelerates and decelerates in while calibrating. First of all, if Bresenham's algorithm is used to draw curves, motor will not accelerate. The reason behind it, is that the motor will move step by step, so it will be impossible to accelerate and decelerate. Motors will accelerate only when plotter draws straight lines, or when it is not drawing (pen is up and laser is off). Also, since the laser needs slower movement speed to engrave paper, laser motors will also not accelerate.



2.2.4.3. Laser

One benefit of using laser is that it does not wiggle like the pen does. The thickness of the engraved line is also constant. So overall picture drawn with laser is alway better quality.

To implement pen, SCTimer configured with 1 kHz clock frequency. mDraw scales the laser from 0-255. When laser power command is received from mDraw, it is rescaled to 0-999 to work with SCT timer.

One important point when programming laser is that it needs slower movement speed, especially when drawing straight lines.

3. Challenges

3.1. Laser and straight lines

Contrary to drawing with pen, laser engraves curves better. However, when engraving straight lines, motors moved faster than drawing curve, which lead to a dotted line instead of a normal straight line.

To fix this problem, we first need detect whether the line is drawing is a straight line or not, then set a slower speed so laser can engrave a better straight line.

3.2. High speed drawing

As mentioned above, high speed drawing is only applicable for pen, since laser need much slower speed. With acceleration and deceleration drawing using pen can get much faster. Nevertheless, after reach a limit, the higher the speed you increase pass that limit, the lower the quality of the picture.

The reason for this is because Bresenham's algorithm is used, which requires the motor to draw step by step. With just one step, motor can not accelerate or decelerate. Suddenly raising the speed of motor from 0 to 10,000 pulses per second would cause two problems: the pen will wiggle (problem is addressed below), and the motor will have a high change of missing steps.

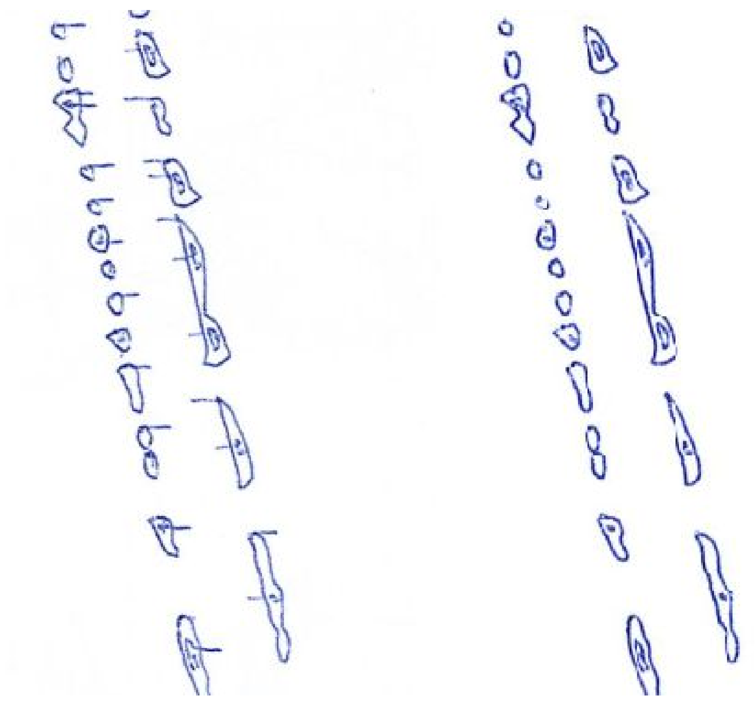
The problem was solved by trial and error. Firstly, we need to choose a image that is complex enough to guarantee that the missing steps error will happen, and must not too complex to draw since we will draw it many times. Then we set the base speed of 1,000 pulses/second, and the maximum speed after acceleration of 3,000 pulses/second. At this speed the image was drawn smoothly. Then we try to increase both the base speed and the accelerated speed, and keep measuring the quality of the image. Eventually, we get the good base speed and accelerated speed, at which the quality of the image is not noticable degraded.

We get the base speed of 2,000 pulses/second and the accelerated speed of 4,000 pulses/second after testing with many different speeds. If the moving distant is large enough (in our case about 2,750 steps - about 3.1cm), we can have more room to accelerate and decelerate, then plotter can reach to 8,000 pulses/second without lower picture quality.

3.3. Drawing dots

When the pen moves up and down, while the motor is moving too fast, it leave a small pen mark. With a normal picture, we don't see much of this problem, but if a picture have many dots, then the pen straights are very easy to spot.

To solve these problems, vTaskDelay of 200ms was used after pen was moved and before motor was moved again. Shorter delays were tested but it didn't fix the problem, and longer delays would make the overall time of the drawing much longer. So 200ms is the best number for the X-Y plotter.



Left: picture with pen marks. Right: picture without pen marks.

4. Conclusion

We learned a lot during the project. First is the benefit of using FreeRTOS. With two different tasks running, one for handling the communication with mDraw, and another for control the plotter, controlling the flow of the whole program is much clearer. FreeRTOS provided queues which are also convenient to handle the data between two parts of the program.

We also learned how to control the stepper motor, the importance of acceleration and deceleration if we want fast moving speed.

As a team, we learn how to collaborate with others. We learned to use github, enjoy the benefit of sharing codes online, always have a backup version. We also spend a lot of time to fix many problems when using Github, especially when you want to merge two different branches into one.

All goals of the project were achieved.