Expertise
and insight
for the future

Plotter Project

Metropolia University of Applied Sciences

Bachelor of Engineering

Smart Systems Engineering

28.9.2020

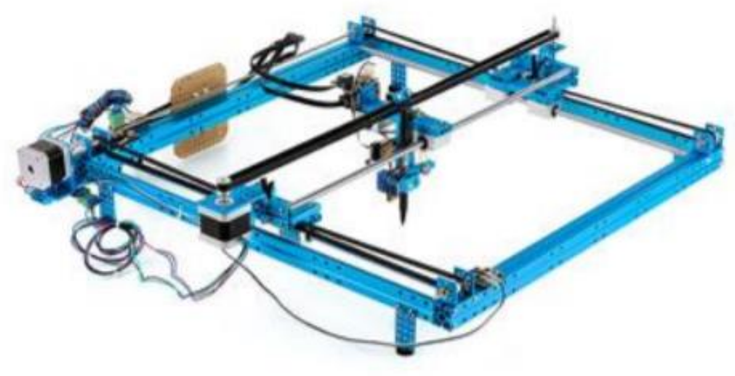
Metropolia
University of Applied Sciences

Contents

1 Introduction

This is a project for running a XY plotter with a microcontroller and mDraw application input.

1.1 Plotter

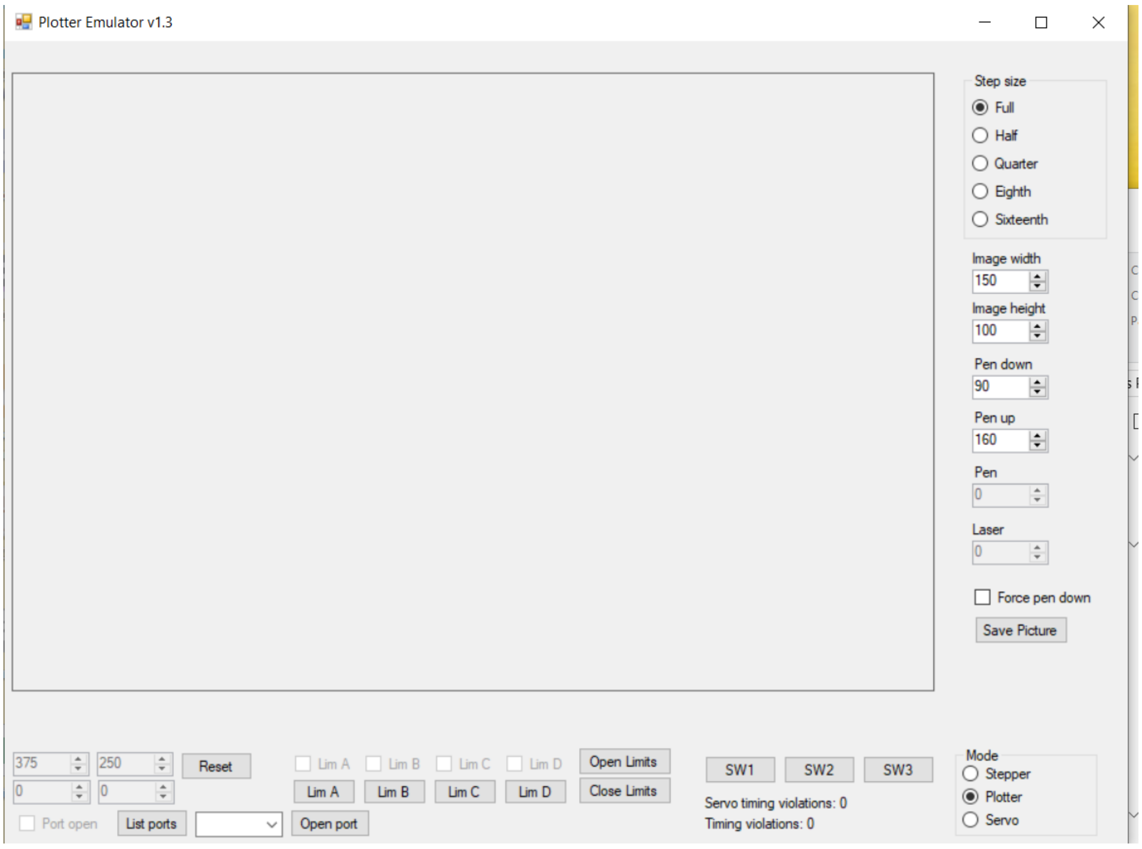


XY plotter, image 1.

XY Plotter has the following interface features:

- X-Axis stepper motor
- Y-Axis stepper motor
- A limit switches for each edge (4)
- Pen servo motor
- Laser engraver

As part of this project a XY plotter emulator was also used to ease work from home during the corona pandemic.



Step size: [✓ Full] Half Quarter Eighth Sixteenth

Image width: 150

Image height: 100

Pen down: 90

Pen up: 160

Pen: 0

Laser: 0

375 | 250

0 | 0

Servo timing violations: 0

Timing violations: 0

Mode: Stepper [✓ Plotter] Servo

Plotter Emulator, image 2.

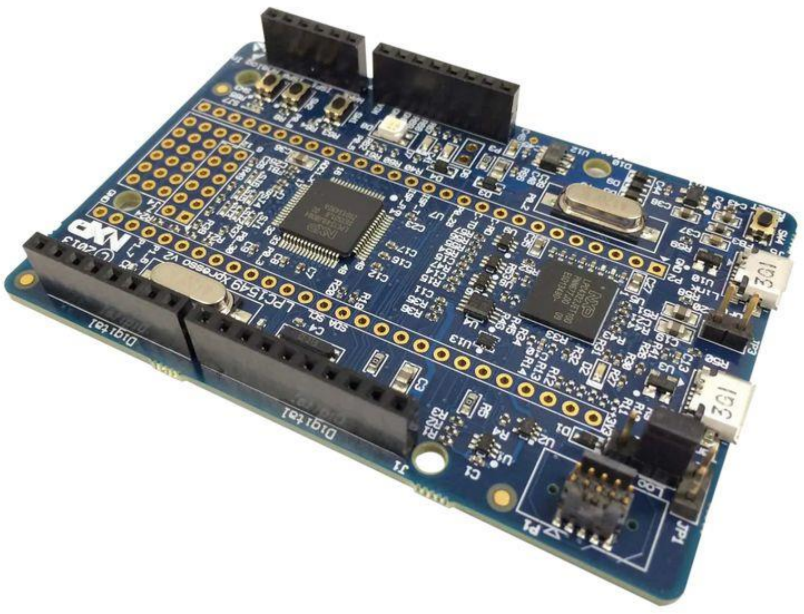
Plotter interfaces are defined as follows:

Pin	**Port**	**Description**
D6	1_3	Limit SW
D7	0_0	Limit SW
D3	0_9	Limit SW
D2	0_29	Limit SW
D12	0_12	Laser
D4	0_10	Pen
D10	0_24	XMotor
D11	1_0	Xmotor Direction
D8	0_27	YMotor
D9	0_28	YMotor Direction
A0	0_8	SW1
A1	1_6	SW2
A2	1_8	SW3

1.2 Microcontroller

Microcontroller used in this project is a LPC1549. Communication between PC with mDraw and the microcontroller is done through UART port. Communication between the microcontroller and the XY plotter components is done through Pin Driven methods.

All software code is stored in microcontroller. XY plotter lacks any kind of software components.



NXP1549, image 3.

1.3 mDraw

mDraw is a software program specially designed for mDrawBots. mDraw is a open source project. For this project we will be only using the XY plotter mode.

mDraw takes SVG images and converts the image to Gcode commands. These Gcode commands will encompass small line movements (among other things) that happen at a small scale so to a human eye complex images can be made drawn quite smoothly including curves.

Gcode commands are formatted to include a command code and data to go with it.

Commands used in this project are as follows:

- M10, Log of opening a com port on mDraw, mDraw queries data from LPC on current running configuration. LPC replies with needed data

- M11, Limit switch status query, mDraw queries current limit switch state. LPC replies with switch states.

- M2, Save pen up/down position, mDraw send data that has pen up/down values. LPC saves data and replies ok.

- M1, Save stepper directions, plot area, and plotting speed, mDraw send multiple data points, LPC saves data and replies ok.

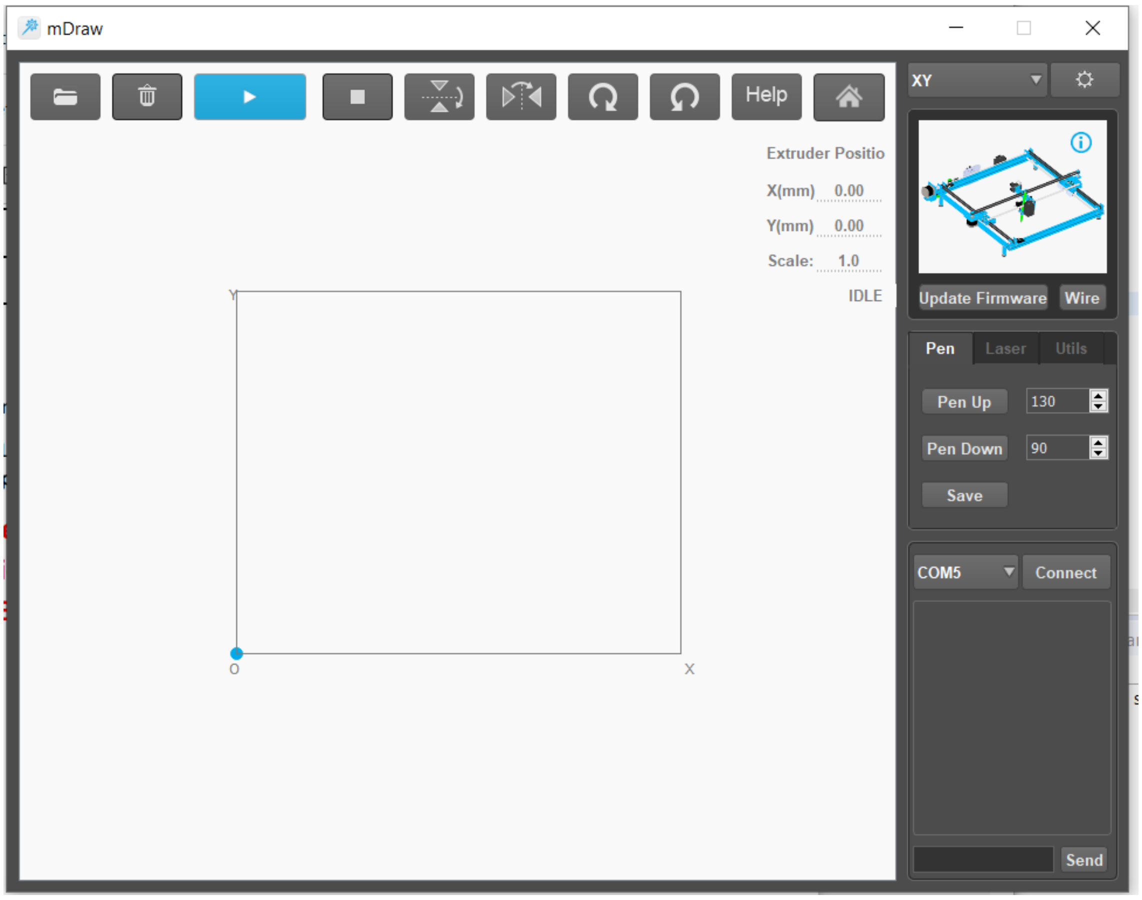
- M4, Set laser power, Sets duty cycle percentage 0-255, LPC saves data and replies ok.

- G28, go to origin, LPC send command internally to move and replies ok.

- G1, go to defined XY coordinate, LPC send command internally to move and replies ok.

Examples and more specific command definitions are included in the project specification pdf.

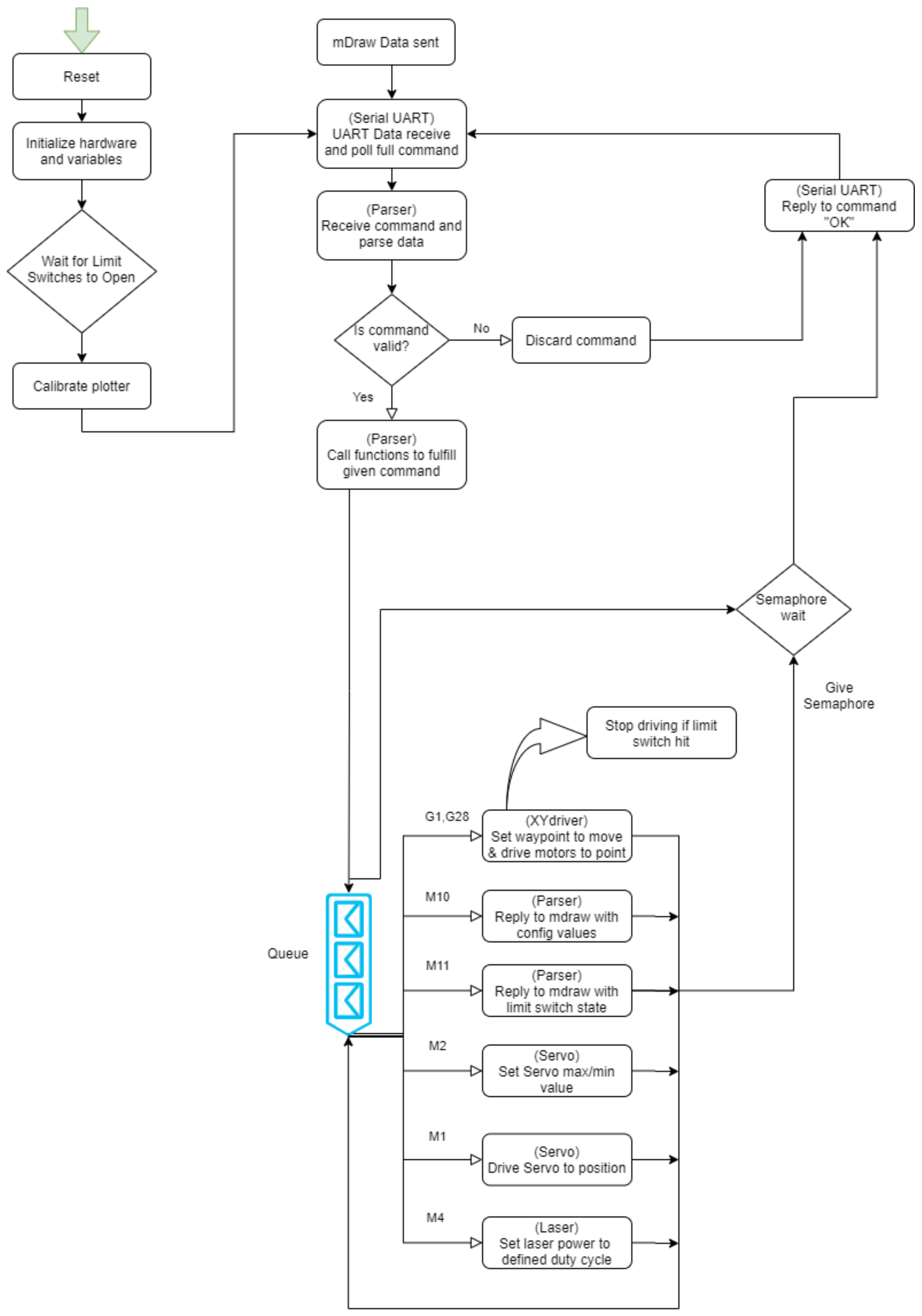
mDraw has a graphic interface to follow where the command output is currently pointing. The graphic interface is independent and will only represent the XY plotter in a working software implementation.



mDraw application, image 4.

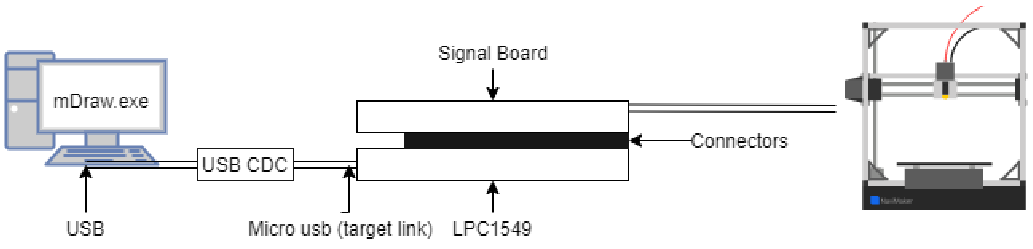
2 Implementation

2.1 Process workflow


Reset
Initialize hardware and variables
Wait for Limit Switches to Open
Calibrate plotter
mDraw Data sent
(Serial UART) UART Data receive and poll full command
(Parser) Receive command and parse data
Is command valid?
No
Discard command
Yes
(Parser) Call functions to fulfill given command
(Serial UART) Reply to command "OK"
Semaphore wait
Give Semaphore
Stop driving if limit switch hit
G1,G28
(XYdriver) Set waypoint to move & drive motors to point
M10
(Parser) Reply to mdraw with config values
M11
(Parser) Reply to mdraw with limit switch state
M2
(Servo) Set Servo max/min value
M1
(Servo) Drive Servo to position
M4
(Laser) Set laser power to defined duty cycle
Queue

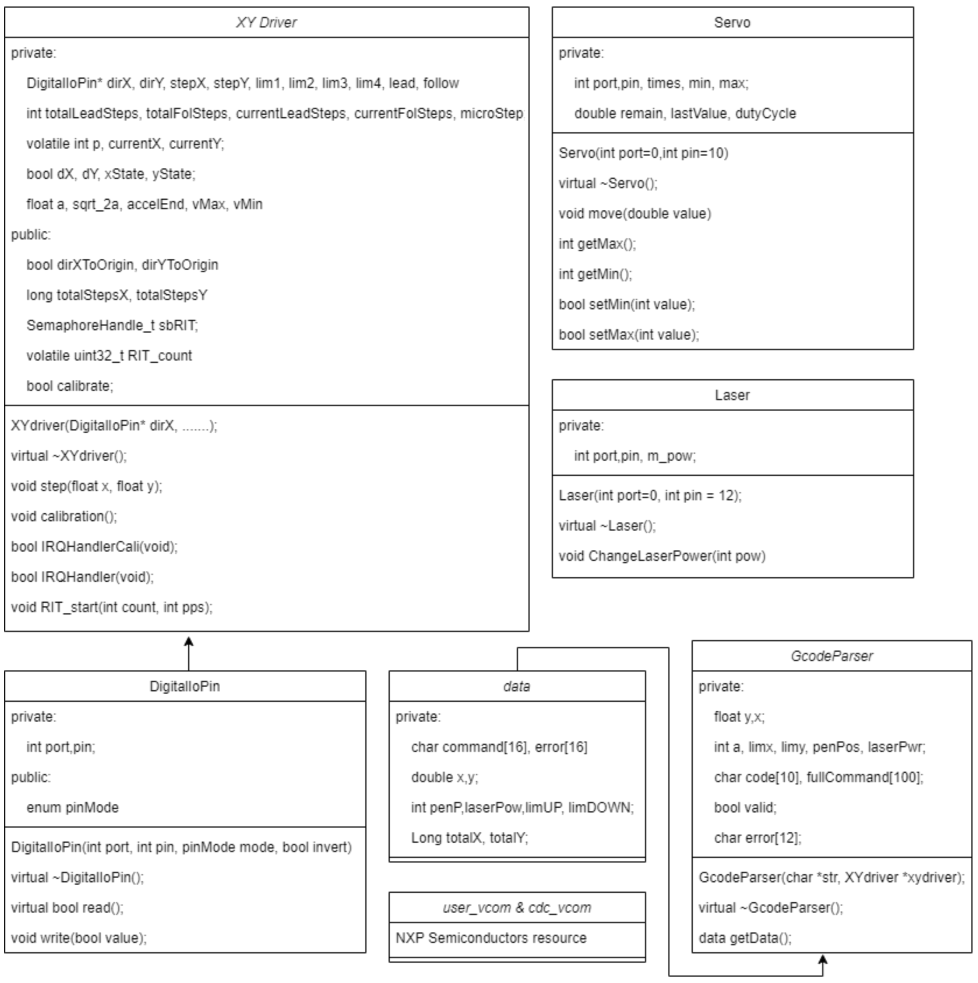
Process workflow, Image 5

2.2 Hardware Diagram



Hardware diagram, image 6

2.3 Class Diagram


XY Driver
private:
DigitalIoPin* dirX, dirY, stepX, stepY, lim1, lim2, lim3, lim4, lead, follow
int totalLeadSteps, totalFolSteps, currentLeadSteps, currentFolSteps, microStep
volatile int p, currentX, currentY;
bool dX, dY, xState, yState;
float a, sqrt_2a, accelEnd, vMax, vMin
public:
bool dirXToOrigin, dirYToOrigin
long totalStepsX, totalStepsY
SemaphoreHandle_t sbRIT;
volatile uint32_t RIT_count
bool calibrate;
XYdriver(DigitalIoPin* dirX, ...);
virtual ~XYdriver();
void step(float x, float y);
void calibration();
bool IRQHandlerCali(void);
bool IRQHandler(void);
void RIT_start(int count, int pps);
Servo
private:
int port,pin, times, min, max;
double remain, lastValue, dutyCycle
Servo(int port=0,int pin=10)
virtual ~Servo();
void move(double value)
int getMax();
int getMin();
bool setMin(int value);
bool setMax(int value);
Laser
private:
int port,pin, m_pow;
Laser(int port=0, int pin = 12);
virtual ~Laser();
void ChangeLaserPower(int pow)
DigitalIoPin
private:
int port,pin;
public:
enum pinMode
DigitalIoPin(int port, int pin, pinMode mode, bool invert)
virtual ~DigitalIoPin();
virtual bool read();
void write(bool value);
data
private:
char command[16], error[16]
double x,y;
int penP,laserPow,limUP, limDOWN;
Long totalX, totalY;
user_vcom & cdc_vcom
NXP Semiconductors resource
GcodeParser
private:
float y,x;
int a, limx, limy, penPos, laserPwr;
char code[10], fullCommand[100];
bool valid;
char error[12];
GcodeParser(char *str, XYdriver *xydriver);
virtual ~GcodeParser();
data getData();

Class Diagram, image 7

3 Testing Plan

3.1 Initial Testing Schedule

Goal in testing would be as follow. Ideally, we would follow this logical progression from input data (foundation) all the way to running the plotter hardware (End-Point for data).

Goal 1.

Parse Gcode, Read mDraw

Goal 2.

Move plotter with hardcode value, Read Limits

Goal 3.

Implement UART, Write commands to UART

Goal 4.

Basic shapes from mDraw, implement angle drawing

Goal 5.

More complicated shapes, improve angle drawing

Goal 6.

Control pen and laser

Goal 7.

Improve program until completion

3.2 Testing finally in brief

02.09.2020 First versions of Gcodeparser and the mDraw functionality with Uart were implemented.	**06.10.2020** First **XYdriver** was built. Testing got at this point to running hardcode values and plotting line at angles.	**11.10.2020** We planned work for the next two days. We planned work for **XYdriver**, **Parser** and **SerialUart.**	**12.10.2020** We continued to work on and test the **XYdriver**, **Parser** and **SerialUart**.
13.10.2020 Better **XYdriver**, **Parser** and **SerialUart** were created. First versions of **Servo** and **Laser** were implemented.	**14.10.2020** Updating and testing the functionality between the programs. Different version on the project was created.	**15.10.2020** Testing and updating the drawing and laser functionalities. **SerialUart** class scrapped for **usb-cdc** implementation.	**16/17.10.2020** Finishing touches on the project.

3.3 Testing outcomes

3.3.1 Gcode Parser

Testing of parser.cpp testing went as follows.
Date 2.9.

Working with the current iteration of Board_UART, problems showed up in reading bulk text copy paste in PuTTY. These issues were avoided by copy pasting single lines at a time. Since single lines would be the working implementation of Mdraw, currently this is fine. Requires further testing with Mdraw itself.

Since parser.cpp is independent from the reading, this issue is not directly a problem for the parser file. Parser file acted as expected and validated the Gcode fine.

In testing it was already noticed that it would be clearer to initialize values to something. This would make debug clearer and possible would be a way to manage moving the needed variables registers.

In summary Parser testing worked as expected, but requires expanding to work as wanted.

Feedback for the program was as follows:

Ok but will require changes to work in LPC. At the moment there is no interface for getting data/executing the g-code.

This is something we will look into during our Mdraw to LPC testing.
Interfaces for UART will be separately added and the way to push the needed variables out in a common manner will either be added directly to parser or in another file.
As our project matures our needs will become clearer for the implementation.

Testing runCommand function had some issues tied to the code array as the code builder was changed multiple times leading to slightly different code builds. Code compare on the other hand was not kept up-to-date with the newest version.

runCommand was scrapped for getData, so command running happened independent from parser, keeping strict jobs clearer.

3.3.2 mDraw reader

First we wanted to be able to connect to the mdraw. We had no idea at first. We saw the com port connection in the mdraw and we had the idea of using Uart. Then we tried our old Uart code from previous exercises and succeeded. We got our first information, which was M10.

After that we tried to get more information out, which gave us more problems. There were problems with “OK” and getting the rest of the coordinates from one line. We also encountered some problems with mdraw, like crashing and the connection with com ports was lost for some reasons. We managed to fix these connection problems by just restarting our mdraw/computer.

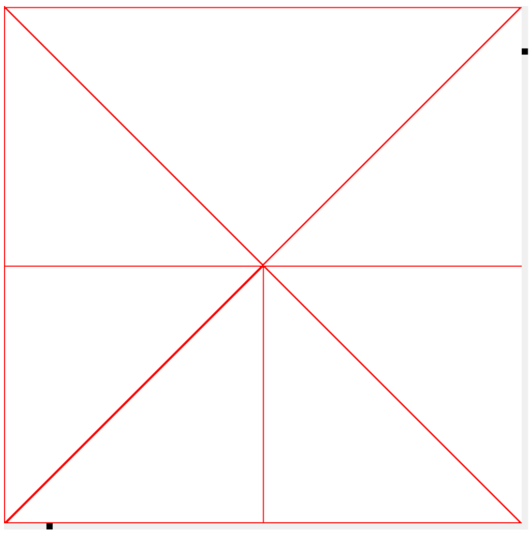
We did get the coordinates and the “OK” to work properly. We managed to do this by understanding the code / trial and error.’

The project ended up using usb-cdc with the virtual com port and the Uart was scrapped.

3.3.3 XY driver

Tested the directions of the plotter. Initial test were done with hardcoded values and acted more like the calibration style running from corners to other corners.

Plotter code initially started a RIT timer for each stepper step. After discovering this was very CPU heavy implementation we moved to a single RIT timer for a whole step command.



Plotter hardcoded drawing, Image 8

3.3.4 Uart/cdc

We used a LpcUart object “vallox” for reading and writing at the beginning. We had problems with the UART implementation and in the end we realised the usb-cdc was a better choice for the debugging and we ended up scrapping the vallox/uart implementation.

3.3.5 Pen servo

Servo brought us few problems. The problems caused the drawing to work wrong. The drawing could just break at one point and cause servo timing violations. It also would also sometimes draw while moving, which would make the drawing a mess. These problems were mostly caused by wrong values and so these problems were fixed by adjusting the values like min and max and the Matchrel values.

3.3.6 Laser Controller

Unexpected outcomes happened with certain values set to the laser and debugging was required. Setting the laser to 0 sets the laser power to 100. Setting the laser to 1 makes the plotter pulse the laser to a point it was visible plotting dots. The problem got solved with testing different values. The off value was set to 2% and that got it working.

4 Documentation

4.1 GCode Parser

Parser program acts as the Gcode parser and validator.
Current iteration validates a char variable and builds an object with the Gcode argument as a variable. Input data is collected independently. Each Gcode command is their own object

Parser can manage M1, M2, M4, M10, M11, G1 and G28. Any hard limits on values are also checked. etc. laser power 255 max, 0 min.

Direct system to move variables to register not planned. Would need more planning on how that would be implemented.

Parser gives a parsed data structure with getData command, which in this case sends data to queue to be managed externally in another task.

4.3 mDraw Reader

How our code works is that it reads characters from virtual com port. When our program reads a defined length of data, it sends a “OK” message to mdraw. This “OK” causes mDraw to send next G-code. This is repeated until all data has been sent by mDraw.

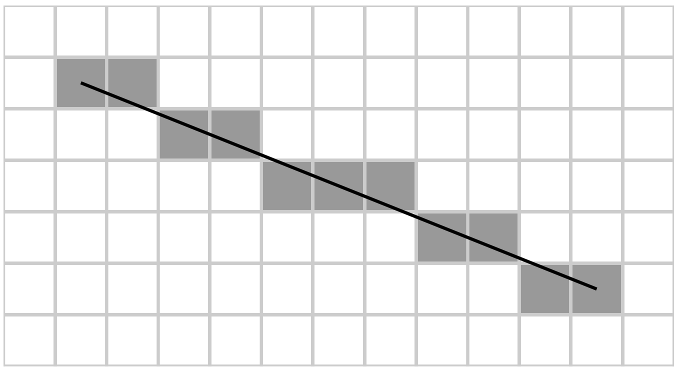
4.4 XY Driver

XY Driver handles both plotter steppers and plotter calibration. Steppers are run with RIT timers to feed the correct pulses for the stepper motors to move the plotter head according to the G-codes.

Calibration is executed after limit switches are released. This will define the borders for plotter after which a limit switch should not get hit.

The code uses the line drawing algorithm is Bresenham’s line algorithm.

https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm



Example of Bresenham’s line drawing algorithm, image 9

4.5 MainP

Main file for project.

ReceiveTask:

Receives data through usb-cdc. Then it parses the data, ads the data to the queue and sends the reply through usb-cdc.

ExecuteTask:

Does the calibration and then it executes commands based on the data object received from the queue.

4.6 Pen Servo

The servo move function makes the servo drive servo to wanted position. This will in effect control pen height, so pen can be lifted and lowered as per mDraw command. Servo also has functions that are used to set and get the servo min and max values.

4.7 Laser Controller

The laser controller drives the laser component to be at a desired PWM frequency. The lasers maximum pwm frequency is 100 kHz. In effect laser will be set on or off by mDraw commanding it when needed and plotter stepper will run the movement for such time. Laser mode must be set on in mDraw for it to drive the component.

References

Image 1, https://www.reichelt.com/gb/en/xy-plotter-robot-kit-v2-0-mb-90014-p255034.html

Image 2, screen capture, emulator

Image 3, https://www.elfadistrelec.fi/fi/lpcxpresso-kehityskortti-lpc1549-lle-nxp-om13056ul/p/30176199

Image 4, screen capture, mDraw

Image 5, Program Flowchart

Image 6, Hardware Diagram

Image 7, Class Diagram

Image 8, Plotter testing with hardcoded values

Image 9, https://en.wikipedia.org/wiki/Bresenham%27s_line_algorithm#/media/File:Bresenham.svg