ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE

Fakulta elektrotechnická

Katedra řídicí techniky

Mikroprocesorové řízení krokových motorů v plotteru

Bakalářská práce

Studijní program: Kybernetika a robotika
Vedoucí práce: Ing. Vít Hlinovský, CSc.

Marek Vlasák

Praha 2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

I. OSOBNÍ A STUDIJNÍ ÚDAJE

Příjmení: **Vlasák** Jméno: **Marek** Osobní číslo: **466306**

Fakulta/ústav: **Fakulta elektrotechnická**

Zadávající katedra/ústav: **Katedra řídicí techniky**

Studijní program: **Kybernetika a robotika**

II. ÚDAJE K BAKALÁŘSKÉ PRÁCI

Název bakalářské práce:

Mikroprocesorové řízení krokových motorů v plotteru

Název bakalářské práce anglicky:

Microprocessor control of stepper motors in plotter

Pokyny pro vypracování:

1) Navrhněte řízení krokových motorů v plotteru a machaniku pisátka.
2) Vytvořte program pro načtení grafických soubrů a jejich následné vykreslení na plotteru.

Seznam doporučené literatury:

[1] Novák, p. - Mobilní roboty-pohony, senzory, žízení. Praha, BEN, 2005
[2] http://www.microcon.cz/
[3] Zbyšek Voda, Průvodce světem ARDUINA, HW kichen, Bučovice, 2017

Jméno a pracoviště vedoucí(ho) bakalářské práce:

Ing. Vít Hlinovský, CSc., katedra elektrických pohonů a trakce FEL

Jméno a pracoviště druhé(ho) vedoucí(ho) nebo konzultanta(ky) bakalářské práce:

Datum zadání bakalářské práce: **06.02.2020** Termín odevzdání bakalářské práce: **22.05.2020**

Platnost zadání bakalářské práce:
do konce letního semestru 2020/2021

Ing. Vít Hlinovský, CSc.	prof. Ing. Michael Šebek, DrSc.	prof. Mgr. Petr Páta, Ph.D.
podpis vedoucí(ho) práce	podpis vedoucí(ho) ústavu/katedry	podpis děkana(ky)

III. PŘEVZETÍ ZADÁNÍ

Student bere na vědomí, že je povinen vypracovat bakalářskou práci samostatně, bez cizí pomoci, s výjimkou poskytnutých konzultací. Seznam použité literatury, jiných pramenů a jmen konzultantů je třeba uvést v bakalářské práci.

. _____
Datum převzetí zadání

_____
Podpis studenta

PROHLÁŠENÍ

„Prohlašuji, že jsem předloženou práci vypracoval samostatně a že jsem uvedl veškeré použité informační zdroje v souladu s Metodickým pokynem o dodržování etických principů při přípravě vysokoškolských závěrečných prací."

V Praze 22. 5. 2020 .....

Marek Vlasák

PODĚKOVÁNÍ

Rád bych poděkoval Ing. Vítu Hlinovskému, CSc. za cenné rady, věcné připomínky a vstřícnost při konzultacích a vypracování bakalářské práce.

ABSTRAKT

Tato bakalářská práce se zabývá návrhem ovládání jednobarevného plotteru, konstrukce pisátka a řešením vektorizace obrazu pro jeho vytištění na plotteru.
Prvním cílem bylo navrhnout konstrukci nového pisátka pro plotter a následně vytvořit program pro řízení plotteru pomocí příkazu GCODE. Dalším cílem bylo najít způsob, jak vytisknout na plotteru obrázek. K tomuto byla použita série funkcí z knihovny openCV pro předzpracování obrazu a jeho následnou vektorizaci. Takto vektorizovaný obraz bylo třeba převést do formátu GCODE příkazů. Dále byl řešen způsob komunikace počítače s plotterem tak, aby byl plotter schopný přijmout korektně jednotlivé příkazy GCODE. Nakonec byla navržena uživatelsky přívětivá aplikace pro komunikaci s plotterem.

Klíčová slova
plotter, vektorizace, detekce hran, GCODE, openCV, tkinter

ABSTRACT

This bachelor's thesis deals with the design of a one-color plotter control, the construction of a pen and the solution of vectorization of image for its printing on a plotter.
The first goal was to design a new plotter pen and then create a program to control the plotter using the GCODE command. Another goal was to find a way to print an image on the plotter. For this, a series of functions from the openCV library was used for image preprocessing and its subsequent vectorization. The vectorized image had to be converted to GCODE command format. Furthermore, the method of communication between the computer and the plotter was solved so that the plotter was able to correctly accept individual GCODE commands. Finally, a user-friendly application for communication with the plotter was designed.

Keywords
plotter, vectorization, edge detection, GCODE, openCV, tkinter

Obsah

Seznam obrázků

1 Úvod

Tématem práce je rozpohybování plotteru ze staré tiskárny a následně navržení způsobu, jak nechat plotter vykreslit libovolný obraz.

V první části práce se zaměřuji na způsob ovládání samotného plotteru. Celé to spočívá v pochopení příkazů GCODE a jejich následné implementaci tak, aby bylo možné celý plotter kontrolovat pouze těmito příkazy. V této části se nejvíce zabývám postupem řešení lineární interpolace, na které stojí celkový pohyb plotteru, a následně pak řeším i kruhovou interpolaci. Cílem je pak zajistit schopnost plotteru reagovat na příkazovou sadu GCODE, tak aby mohl vykreslit objekty popsané „programovacím“ jazykem GCODE.

Poté se v druhé části zabývám principem vektorizace obrazu a jeho následným popsáním pomocí příkazové sady GCODE tak, abych byl schopný daný obraz vytisknout na plotteru. Obraz zpracuji za pomocí knihovny OpenCV. Popíšu zde jednotlivé kroky vektorizace i algoritmy, které jsou použity pro vektorizaci v jednotlivých krocích. Výsledkem by pak měl být textový soubor, který bude obsahovat veškeré příkazy GCODE, které pak bude možno poslat plotteru pro vykreslení.

V poslední části se zaměřím na způsob komunikace plotteru s počítačem a návrh uživatelského rozhraní.

2 Hardware plotteru

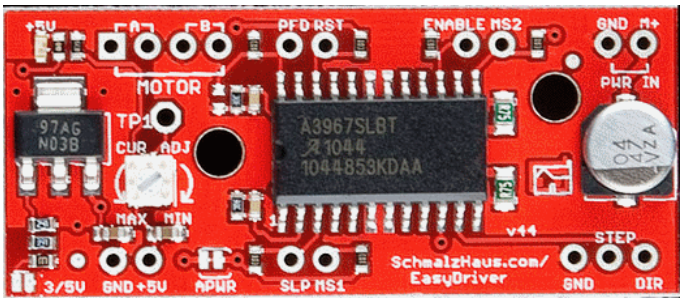
2.1 Řídicí jednotka Arduino Due

Celý ploter řídí vývojová deska Arduino Due s procesorem Atmel SAM3X8E ARM Cortex-M3. Procesor je 32-bitový a jedná se o první Arduino založené na 32-bitové architektuře. Díky tomu je schopné zpracovat až 4B dat v rámci jednoho taktu. Procesor běží na taktu 84 MHz. Vývojová deska disponuje pamětí Flash s kapacitou 512kB a SRAM s velikostí 96 kB. Na desce se nachází 54 digitálních I/O pinů, 12 analogových vstupů, 4 UART, USB umožňující OTG připojení a 2 DAC piny [1].

Obrázek 1: Řídicí jednotka Arduino Due

2.2 Ovladače krokových motorů EasyDriver

Driver krokových motorů A3967 je výstupní modul pro Arduino, též známý pod názvem EasyDriver. Modul slouží k ovládání krokových motorů a je kompatibilní se 4, 6 a 8 vývodovými krokovými motory. Rozlišení kroků lze nastavit pomocí jumperů MS1 a MS2 na: Plný krok, poloviční krok, čtvrtinový krok a osminový krok. V projektu používám osminový krok, tzn. jumpery nejsou zapojené. Rozsah napájecího napětí motorů je 6-30 V. Modul se dá řídit 5V i 3,3V logikou. Pokud by bylo třeba řídit proud jdoucí do motorů je zde umístěn i trimr pro regulaci výstupního proudu z modulu v rozsahu 150-750mA [8].



Obrázek 2: Ovladač krokových motorů EasyDriver

2.3 Krokové motory

U plotteru jsou použity dva krokové unipolární motory [3]. Jeden pro ovládání osy X a druhý pro osu Y. Jedná se o krokové motory se 6 vývody, které jsou napájeny ze zdroje 12 V.

Obrázek 3: Krokový motor

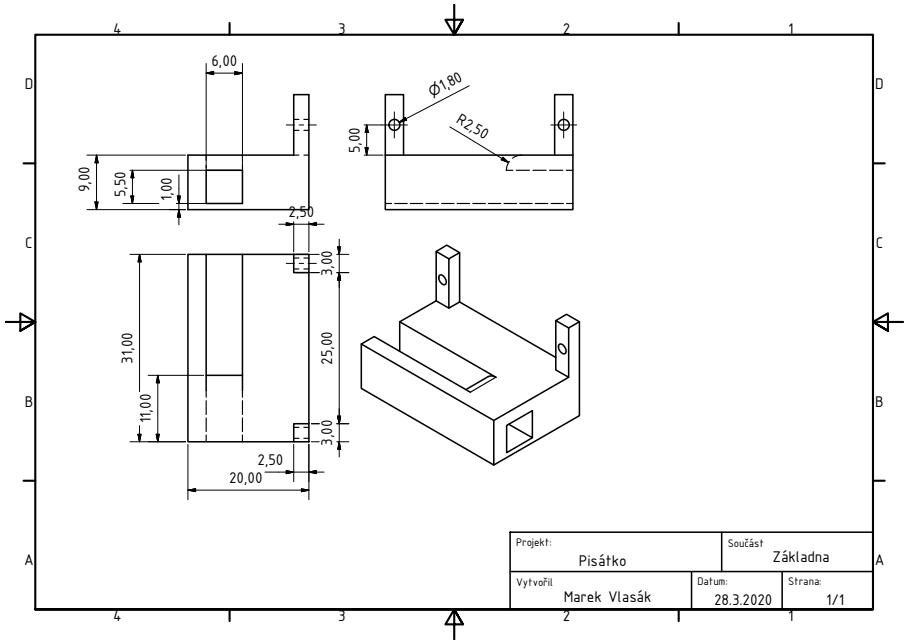
2.4 Servomotor

Pro ovládání pisátka je použit servomotor SG90. Tento servomotor je schopný otáčet se o 180°. Motor disponuje silou 1,2 Kg/cm při napětí 4,8 V, což je pro potřeby ovládání pisátka plně dostačující. Pro jeho řízení se využívá PWM výstup z Arduina. Rozsah napájecího napětí je 4-7,2 V [4].

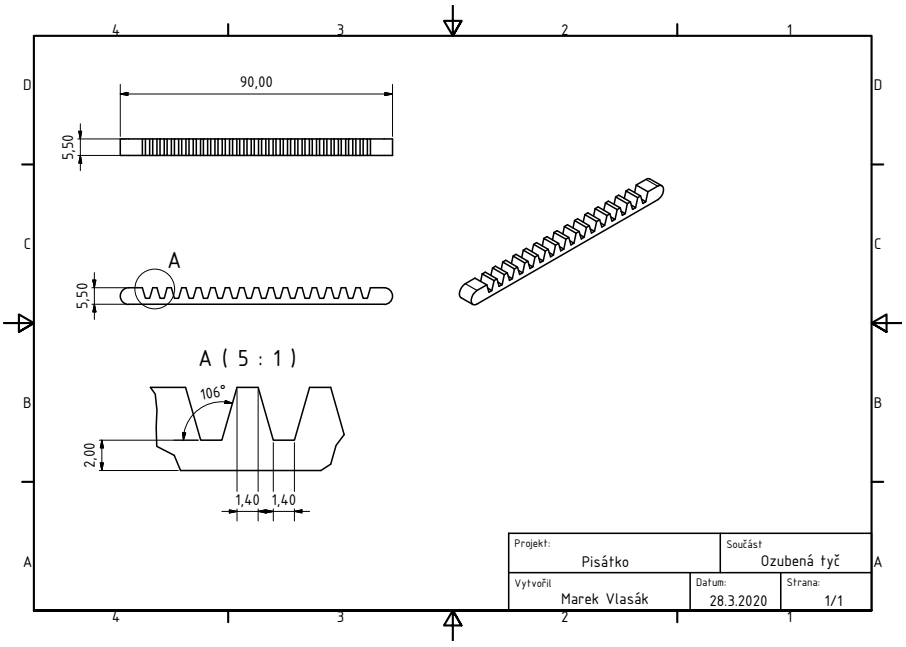
Obrázek 4: Servomotor SG90

2.5 Návrh a konstrukce pisátka

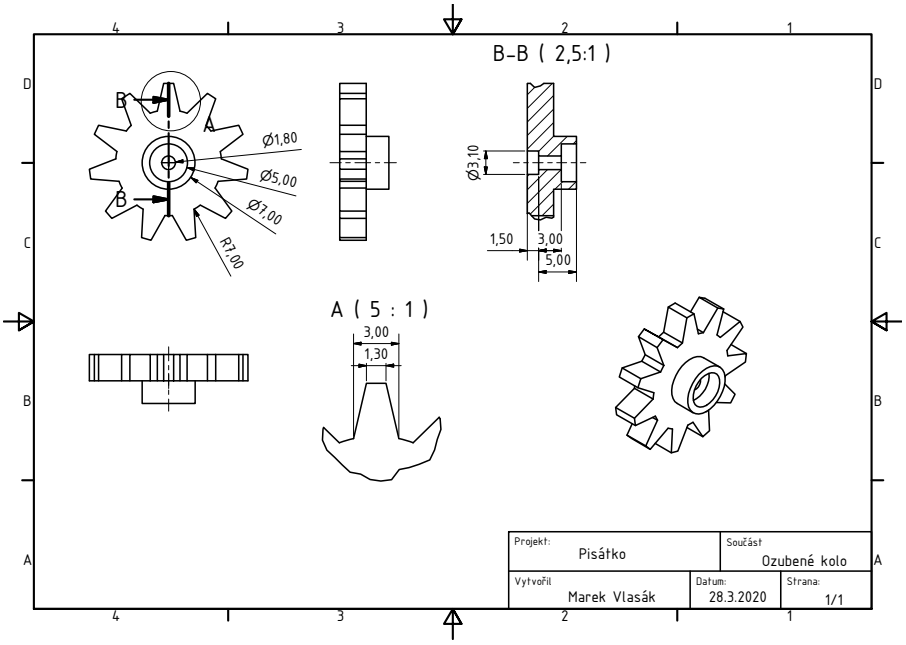
V prvotním zapojení jsem používal pisátko, které již bylo součástí plotteru. Toto jsem ovládal pomocí relé, které spínalo elektromagnetický spínač. Bohužel, tento elektromagnetický spínač přestal během testování fungovat a oprava nebyla možná. Bylo tedy zapotřebí navrhnout nové pisátko. Pro správnou funkci nového pisátka bylo zapotřebí navrhnout konstrukci a zapojení tak, aby docházelo k lineárnímu posunu po ose Z. Pro ovládání pisátka jsem použil servomotor SG90. Tento servomotor je v základním nastavení pouze rotační, a bylo tedy nutné navrhnout převod z rotačního posunu na lineární a uchycení tužky. Výsledný návrh konstrukce lze vidět na následujících technických výkresech.



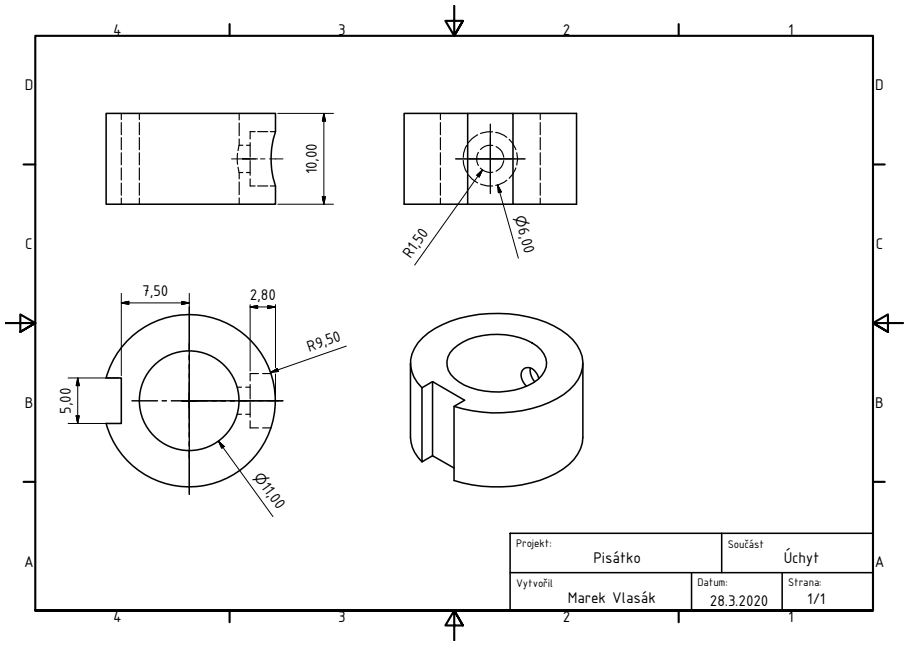
Obrázek 5: Technický výkres základny pisátka



Obrázek 6: Technický výkres ozubeného hranolu



Obrázek 7: Technický výkres ozubeného kola

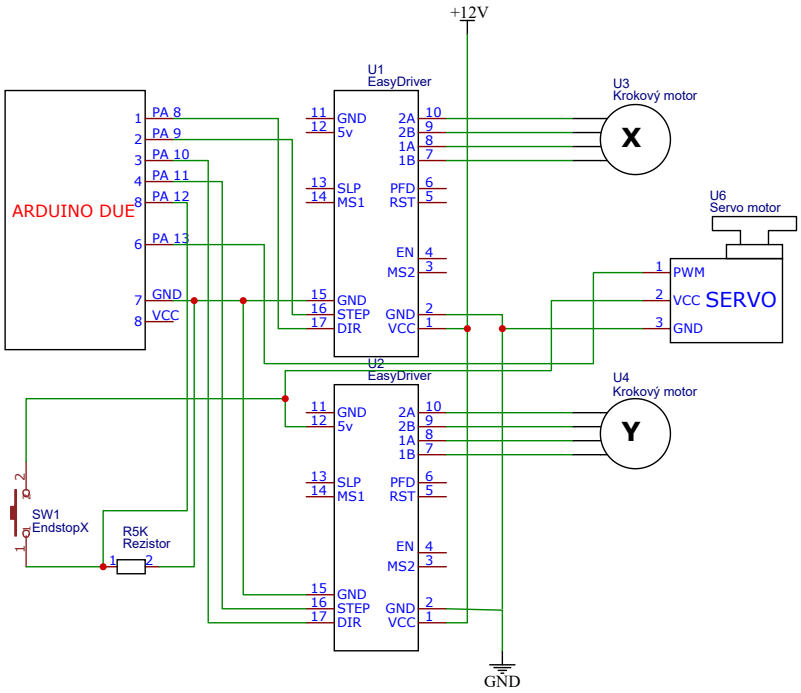


Obrázek 8: Technický výkres úchytů tužky

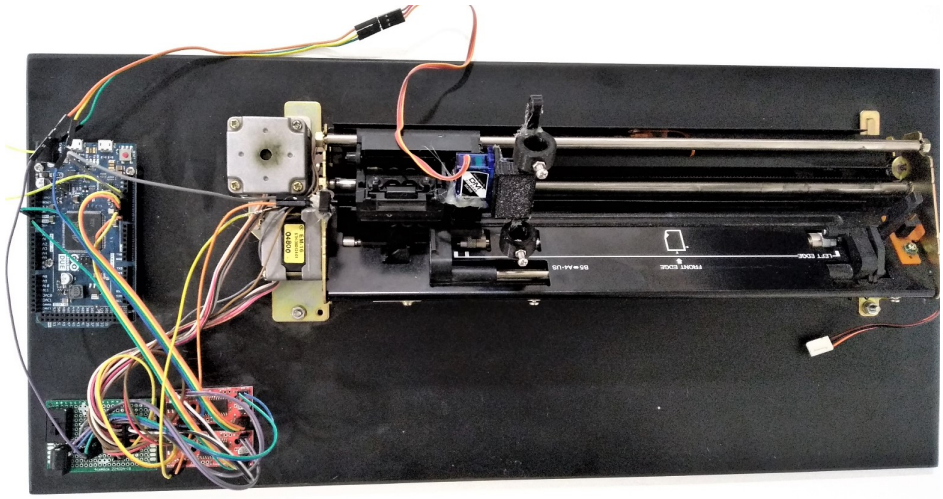
Po dokončení návrhu se jednotlivé modely vytiskly z PLA plastu na 3D tiskárně Anet A8. Vytištěné díly dobře zapadaly, nicméně u základny pisátka byla vůle v lineárním posunu, čímž by mohlo docházelo k nepřesnostem. Tento problém jsem jednoduše vyřešil tím, že jsem místa v drážce základny, kde se posouvá ozubený hranol, upravil přidáním výplně za pomoci 3D pera. Jednotlivé součásti byly pospojovány za pomoci tavné pistole nebo 3D pera a následně připevněny na původní tiskovou hlavu tak, aby nedošlo k jejímu poškození.

2.6 Zapojení plotteru

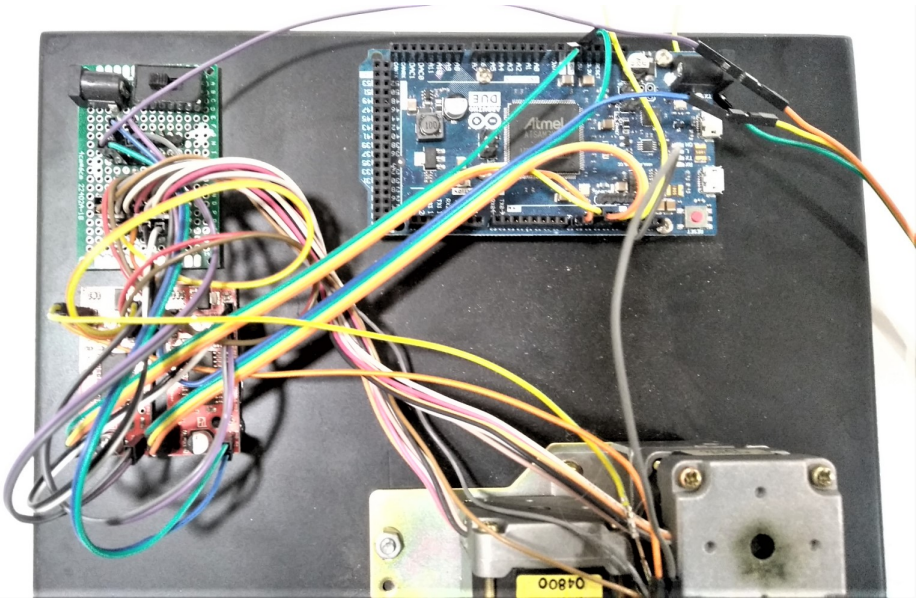
Na následujícím obrázku lze vidět schéma zapojení jednotlivých elektrických součástí a fotografie celého plotteru.



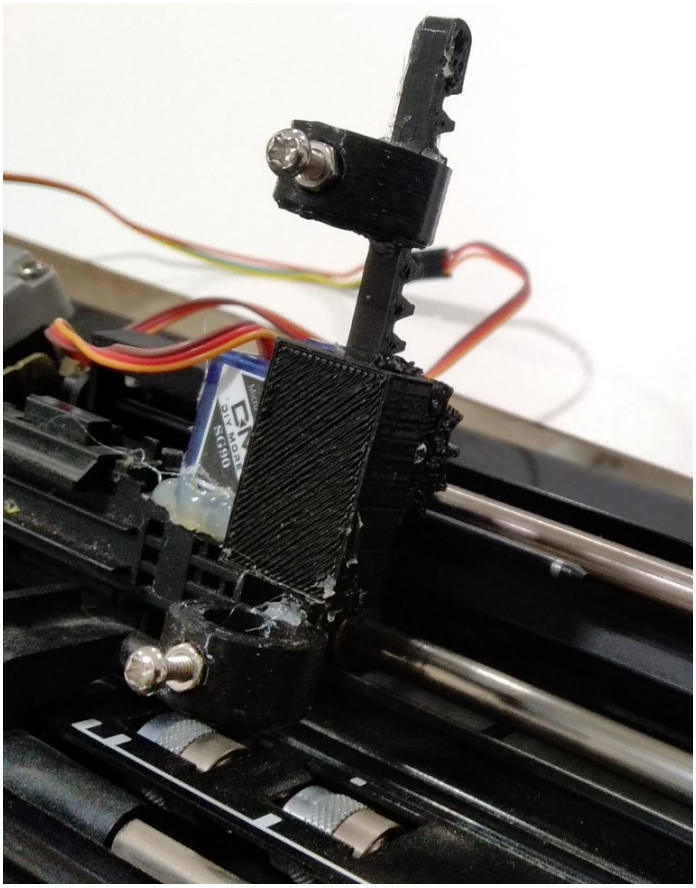
Obrázek 9: Schéma zapojení přípravku



Obrázek 10: Použitý plotter



Obrázek 11: Detail zapojení plotteru



Obrázek 12: Detail konstrukce pisátka

3 Softwarové ovládání plotteru

3.1 Arduino Wiring

Arduino Due lze programovat v jazyce C nebo C++. Nicméně je jednodušší používat knihovnu **Wiring.** V dnešní době se jedná o nejrozšířenější knihovnu pro programování Arduina. Díky své komplexnosti se o ní občas hovoří jako o samostatném programovacím jazyce. Pro programování Arduina se často používá Arduino IDE, což je integrované vývojové prostředí, které je napsáno v jazyce Java. Vychází z výukového prostředí Processing, které bylo mírně upraveno a rozšířeno o nové funkce a v neposlední řadě i o podporu knihovny **Wiring** [5]**.**

Pro programování plotteru jsem se nakonec rozhodl využít tedy knihovnu **Wiring.**

3.2 Příkazová sada GCODE

Pro řízení obráběcích a jim podobných strojů se nejvíce používá GCODE. Jedná se v podstatě o programovací jazyk pro počítačové číselné řízení (dále jen CNC), kterým se tyto stroje programují. Tímto jazykem v podstatě říkáme stroji, jak a co má vyrobit. Říká mu, jakou rychlostí a směrem se mají pohybovat motory, jaký typ nástroje použít aj. GCODE se rozděluje na dvě hlavní skupiny příkazů. Jedna skupina jsou G-kódy, které určují, o jaký pohyb se jedná, například lineární interpolace, kruhová interpolace, absolutní pozicování apod. Druhá sada jsou M-kódy, které ovládají funkce stroje, například spouštění chladící kapaliny, ukončování programu, zastavení stroje aj. GCODE se hojně využívá i u 3D tisku, což mě vedlo k tomu, že pro řízení plotteru používám GCODE [6]. Dalším důvodem pro použití GCODE je fakt, že se jedná o jednoduchý a velice rozšířený jazyk.

3.3 Využívané příkazy v plotteru

Plotteru pro provoz postačí pouze minimum příkazů. Plotter rozeznává následující příkazy:

G00

Pracovní posun do bodu X Y, spouštění pisátka.

Příklad posunu: G00 X30 Y40

Příklad spuštění a zvednutí pisátka: G00 Z0 (spuštění), G00 Z1 (zvednutí)

G01

Lineární interpolace do bodu X Y.

Příklad: G01 X30 Y40

G02

Kruhová interpolace po směru hodinových ručiček do bodu X Y po oblouku se středem v bodě I J. Tento střed je vztažen k počátečnímu bodu.

Příklad: G02 X5 Y0 I5 J0

G03

Kruhová interpolace v protisměru hodinových ručiček do bodu X Y po oblouku se středem v bodě I J. Tento střed je vztažen k počátečnímu bodu.

Příklad: G02 X5 Y0 I5 J0

G28

Přesun na domovskou pozici.

Příklad: G28

G90

Přepnutí na absolutní pozicování.

Příklad: G90

G91

Přepnutí na relativní pozicování.

Příklad: G91

M0

Úplný restart plotteru.

Příklad: M0

3.4 Ovládání krokových motorů a pisátka

V první řadě bylo třeba rozpohybovat jednotlivé osy, které jsou poháněny krokovými motory, a také servomotor pro ovládání pisátka.

Krokové motory jsou ovládány pomocí dvou vstupních pinů EasyDriver modulu, kterými jsou STEP (otočení o definovaný krok) a DIR (směr otáčení). Pro posouvání se na pin STEP posílá obdélníkový signál. Na každou náběžnou hranu obdélníkového signálu motor provede jeden krok v daném směru. Směr se ovládá logickou úrovní napětí přivedeného na již zmíněný pin DIR. Pro vysokou úroveň (3.3 V) máme nastaven pohyb v jednom směru a pro nízkou (0 V) ve směru opačném. Tímto způsobem jsem docílil rozpohybování krokových motorů, které ovládají osy X a Y.

Pro přesný pohyb jsem musel nejdříve zjistit, jaké vzdálenosti odpovídá právě jeden krok. Hodnotu jsem určil experimentálně na obou osách tak, že jsem nechal plotter vykreslit čáru pro zadaný počet kroků a následně za pomoci posuvného měřítka odečetl hodnotu z papíru. Pro 500 kroků jsem na ose X přibližně odečetl hodnotu $12,55\,\text{mm}$ a na ose Y $12,60\,\text{mm}$. Výslednou vzdálenost, kterou vykreslí jeden krok, jsem poté vypočetl následovně:

$$vzdálenost\,na\,1\,krok = \frac{vykreslená\,vzdálenost}{počet\,kroků} \tag{1}$$

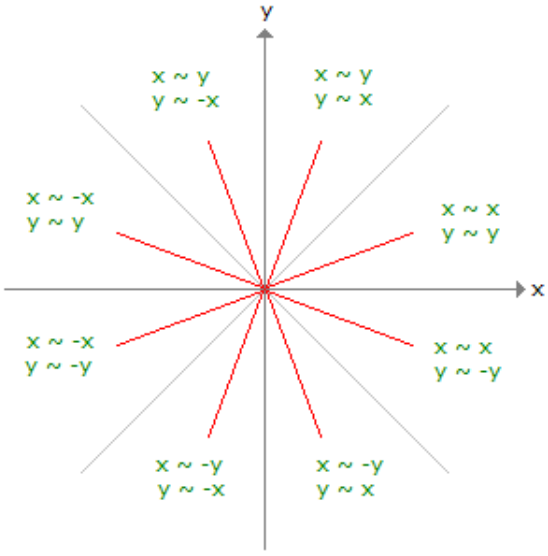
Pro obě osy vyšla přibližná hodnota stejná, a to $0,025\,\text{mm/krok}$.

Osa X je omezená a bylo třeba kontrolovat, aby se tisková hlava nesnažila dostat mimo tiskovou plochu. Tisková plocha je v ose X kvůli konstrukci omezená na 140 mm. Kontrola probíhá hned při spuštění zařízení, kdy tisková hlava dojede ke spínači, který detekuje doraz. Z této pozice se pak tisková hlava přesune na stanovenou počáteční pozici. Osa Y je v podstatě nekonečná a nebylo tedy třeba kontrolovat doraz.

Servomotor pisátka je ovládán pomocí jediného pinu. Pro otočení servomotoru o požadovaný úhel se na tento pin pošle impuls o určité délce, která odpovídá požadovanému úhlu natočení. Správnou délku impulsu pak řeší arduino knihovna **Servo.h**. Ovládání pisátka pak spočívá pouze v pootočení ozubeného kola, které je ovládáno servomotorem.

3.5 Pohyb po přímkách

V předchozím bodě jsme docílili základního ovládání posunu po osách X a Y. Nyní je třeba tento pohyb rozšířit o možnost pohybovat se po úsečkách zadaných počátečními a koncovými body. Zároveň je nutné si uvědomit, že v daný okamžik se můžeme posunout pouze ve směru jedné z os. Pro vyřešení tohoto pohybu jsem se rozhodl využít Bresenhamův algoritmus [7] pro rýsování čar. Nevýhodou tohoto algoritmu je fakt, že funguje pouze pro úsečky, které svírají s osou X úhel 45°. Tento problém jednoduše vyřešíme tak, že pokud se pohybujeme mimo tento prostor, dané souřadnice přemapujeme do prostoru, kde Bresenhamův algoritmus funguje správně, a to způsobem, který je zobrazen na obr. 13



Obrázek 13: Princip mapování souřadnic[8]

3.5.1 Bresenhamův algoritmus

Algoritmus zná počáteční (0, 0) a koncový bod (x_1, y_1). Základní algoritmus na počátku vypočítá směrnici přímky:

$$m = \frac{y_1}{x_1} \tag{2}$$

S takto vypočtenou hodnotou směrnice přímky algoritmus pracuje v plovoucí řádce. Nicméně je lepší použít upravenou verzi algoritmu, která počítá v celých číslech, pro zvýšení rychlosti výpočtů [9]. Toho docílíme jednoduchou úpravou, kdy rovnici (2) vynásobíme $2x_1$. Pro m tedy bude platit:

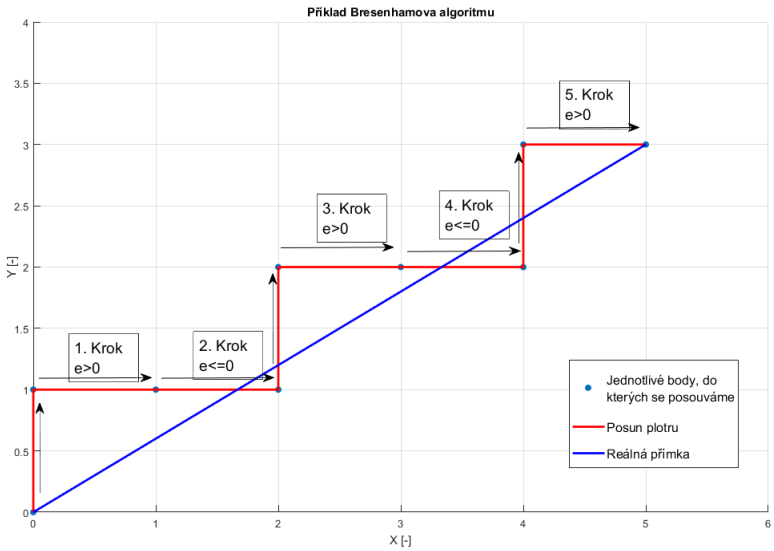
$$m = 2y_1 \tag{3}$$

Na počátku algoritmu si definujeme chybovou konstantu $e = -x_1$, která slouží ke kontrole přechodu přes přímku. Postupně se posouváme po ose X. Při každém kroku se k chybě e přičte hodnota m a následně se rozhoduje, zda-li je chyba e větší než 0. Kontroluje se tím skutečnost, zda jsme již přešli přes přímku. Pokud je podmínka splněna, dojde k posunu po ose X i po ose Y a od chyby e se odečte $2x_1$. V opačném případě, kdy e není větší než 0, dojde pouze k posunu po ose X. Algoritmus končí, jakmile dosáhne koncového bodu x_1.

Pro lepší představu si uvedeme příklad. Uvažujme koncový bod o souřadnicích (5, 3). V následující tabulce jsou vypočteny jednotlivé kroky algoritmu až do jeho konce.

Krok	X	e	$e + m$	$e - 2x_1$	$e + m > 0$	Y
0	0	-5	-	-	Ne	0
1	1	-5	1	-9	Ano	1
2	2	-9	-3	-	Ne	1
3	3	-3	3	-7	Ano	2
4	4	-7	-1	-	Ne	2
5	5	-1	5	-5	Ano	3

Úsečku i jednotlivé body je možné vidět na následujícím grafu. Postupuje se schodovitě tak, abychom se příliš nevzdálili od dané přímky. Tímto způsobem jsem schopný vykreslit libovolnou úsečku.



Obrázek 14: Příklad postupu Bresenhamova algoritmu

3.6 Pohyb po obloucích

V bodě 3.4 jsem vyřešil pohyb po přímkách, je však třeba umět realizovat i pohyb po obloucích. Tento pohyb realizujeme tak, že daný oblouk rozdělíme na mnoho malých úseček, které následně vykreslíme pomocí již zmíněného Bresenhamova algoritmu.

Oblouk je zadán dle GCODE příkazu koncovým bodem B a středem oblouku (I, J), který je vyjádřen vůči počátečnímu bodu A [10]. Tyto údaje postačí k tomu, abychom daný oblouk rozdělili na jednotlivé úsečky.

Nejdříve spočteme vzdálenost počátečního bodu A od koncového bodu B a poloměr oblouku:

$$d = ||A - B|| = \sqrt{(A_x - B_x)^2 + (A_y - B_y)^2} \tag{4}$$

$$r = \sqrt{I^2 + J^2} \tag{5}$$

Dále z těchto znalostí spočteme úhel výseče:

$$\alpha = 2arcsin(\frac{d}{2r}) \tag{6}$$

Nakonec vypočteme délku oblouku:

$$l = \alpha r \tag{7}$$

Z délky oblouku l spočteme počet segmentů n na oblouk, a to tak, že délku oblouku podě-

líme koeficientem rozlišení k. Jako vhodný koeficient rozlišení jsem testováním zvolil hodnotu 0.5 mm. Počet segmentů spočteme následovně:

$$n = \frac{l}{k} \tag{8}$$

Dále získáme úhel β, o který se budeme posouvat po oblouku tak, že celkový úhel α vydělíme počtem segmentů n:

$$\beta = \frac{\alpha}{n} \tag{9}$$

Vyjádříme si počáteční úhel γ_0, od kterého se budeme posouvat, pomocí funkce $arctan2$ ze známých souřadnic středu oblouku. Pokud by výsledná hodnota byla záporná, přičteme 2π.

$$\gamma_0 = arctan2(\frac{-J}{-I}) \tag{10}$$

Jednotlivé body, do kterých se budeme posouvat po úsečkách, dostaneme tak, že k počátečnímu úhlu vždy přičteme úhel posunutí β a spočteme souřadnice následujícího bodu, který leží na daném oblouku. Zároveň zde rozlišujeme, jestli se jedná o pohyb po směru, nebo v protisměru hodinových ručiček. Pokud se pohybujeme v protisměru hodinových ručiček, úhel β přičítáme postupně k úhlu γ_0 a kontrolujeme, zda není nový úhel větší než 2π. Pokud ano, odečteme 2π od následujícího úhlu, abychom zůstali v intervalu $[0, 2\pi]$. Jestliže se pohybujeme po směru hodinových ručiček, daný úhel β odečítáme a kontrolujeme, zda není následující úhel záporný. Pokud je, přičteme k následujícímu úhlu 2π tak, abychom opět zůstali v intervalu $[0, 2\pi]$. Následující souřadnice bodů spočteme následovně:

$$x_{další} = A_x + I + r \cdot cos(\gamma_i) \tag{11}$$

$$y_{další} = A_y + J + r \cdot sin(\gamma_i) \tag{12}$$

Úhel γ_i značí následující úhel, který je již upraven tak, aby byl v intervalu $[0, 2\pi]$ a $i \in \{0, 1, 2, ..., n\}$, kde n značí již zmíněný počet segmentů oblouku.

Tato metoda je vcelku intuitivní, ale má svá omezení. Ta spočívají v tom, že tato metoda funguje pouze pro oblouky s kruhovou výsečí do úhlu 180°. Nicméně to není problém, neboť generátory GCODE rozdělují oblouky na menší segmenty [11].

4 Vektorizace obrázku

4.1 Rozdělení grafických formátů

V základu rozdělujeme grafiku na dva typy, a to na vektorovou a rastrovou.

V rastrové grafice je obrázek popsán pomocí jednotlivých pixelů. Tyto v sobě mají uloženou informaci o barvě dle použitého barevného modelu. Pixely jsou uspořádávány do mřížky, kde má každý definované souřadnice. Nevýhodou je, že se daný obrázek nedá zvětšovat bez toho, aniž by docházelo k rozmazání. Zároveň je to však nejjednodušší způsob pro záznam digitálních fotografií.

Vektorová grafika se liší od rastrové hlavně v tom, že je popsána pomocí definovaných útvarů, jako jsou body, přímky, křivky apod. Podstatná výhoda oproti rastrové grafice spočívá v tom, že daný obrázek vytvořený ve vektorové grafice lze libovolně zmenšovat nebo zvětšovat bez ztráty na kvalitě.

4.2 Postup při vektorizaci

Vektorizace je proces, při kterém z rastrové grafiky získáváme grafiku vektorovou. Předpokladem pro dobrý výsledek je rozlišení - čím větší rozlišení, tím lepší výsledek. Během procesu se nejdříve předzpracuje samotný rastrový obraz tak, aby byl vhodný pro vektorizaci. Následně se provedou níže uvedené kroky pro to, abychom získaly úsečkami popsaný obraz.

Pro zpracování obrázku do vektorové grafiky a následného převodu do GCODE jsem se rozhodl použít programovací jazyk Python a knihovnu OpenCV. Knihovna se používá ke zpracování obrazů a obsahuje spoustu funkcí pro práci s nimi [12]. Pro ukázku použijeme následující obrázek.

Obrázek 15: Ukázková fotka

4.2.1 Převod obrazu

Jelikož plotter využívá pouze jednu barvu pro kreslení, stačí nám z obrazu dostat obrysy. Z počátku tedy načteme obraz a následně jej pomocí funkce **cvtColor** převedeme do odstínů šedi. Takto převedený obraz je vhodný pro převod do binárního obrazu.

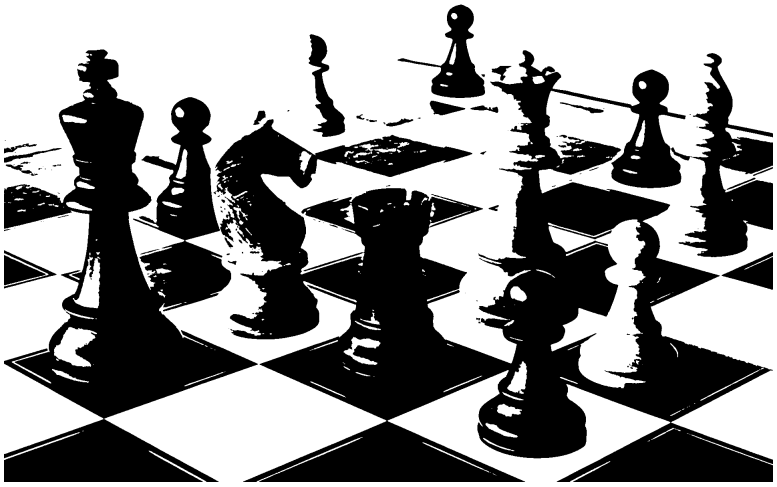
Obrázek 16: Převod do odstínů šedi

4.2.2 Detekce hran

Pro detekci hran je vhodné mít obraz převedený do odstínů šedi. To jsme provedli již předchozím bodě. Knihovna openCV nabízí několik možností, jak získat z obrazu obrysy. Postupně si uvedeme jednotlivé metody a ukážeme si výsledky.

Prahování

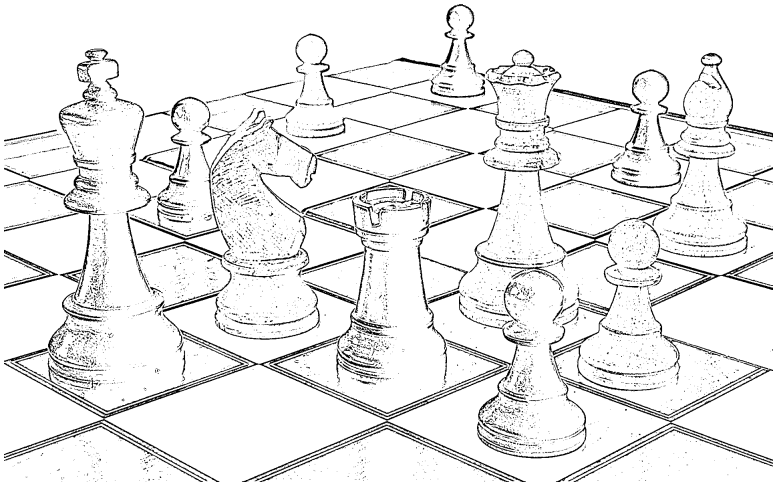
Použijeme-li funkci **threshold** pro detekci obrysů, používáme ji v binárním režimu, tzn. nastavujeme pixelům pouze dvě možné hodnoty barev. Algoritmus je jednoduchý. Postupně procházíme jednotlivé pixely obrazu v odstínech šedi a na základě barvy pixelu a námi zadané hodnoty prahu rozhodneme, zda se pixel nastaví na černou, nebo na jinou (zpravidla bílou) barvu. Výsledek je možné vidět na obr. 17



Obrázek 17: Obraz po použití prahování

Adaptivní prahování

Samotná funkce **threshold** má omezení. Používá se u ní přesně nastavená hodnota prahu, a to může u některých obrazů vést k nekvalitním výsledkům. Zejména jde o obrazy, u kterých je rozdíl v nasvícení jednotlivých částí. Tento problém lze vyřešit pomocí adaptivního prahovování. Tato metoda nepoužívá předem nastavenou hodnotu prahu, ale pro každý pixel spočítá tuto hodnotu zvlášť na základě malého regionu kolem něj. Pro výpočet slouží buď metoda, při které je hodnota prahu rovna hodnotě středu okolních hodnot, od které se odečte zadaná konstanta, nebo se použije metoda, při které je hodnota prahu rovna gausovsky váženému součtu, od kterého je odečtena opět zadaná konstanta. Tím jsme schopni získat rozdílné hodnoty prahu pro rozdílné části obrazu, čímž docílíme lepších výsledků obrazu s rozdílným nasvícením [12]. Výsledek je vidět na obrázku níže



Obrázek 18: Obraz po použití adaptivního prahování

Funkce Canny

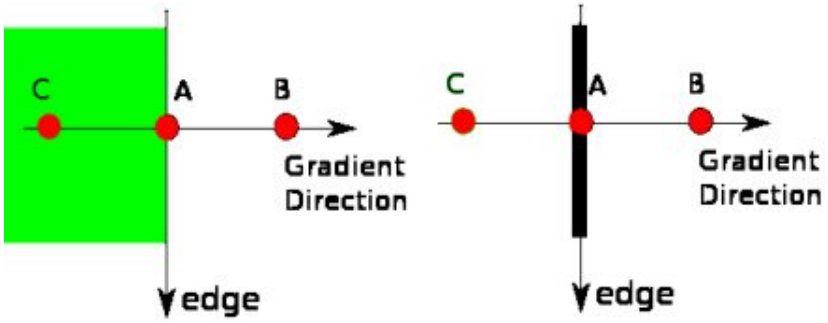
Canny Edge Detection je populární algoritmus pro detekci hran. Tento algoritmus se dělí do několika fází. V první fázi dochází k redukci šumu v obraze. Pro jeho potlačení se používá Gaussovský filtr. Šum je třeba potlačit, neboť má špatný vliv na detekování hran.

V takto vyhlazeném obraze se najde gradient intenzity tak, že se obraz profiltruje jádrem Sobel v horizontálním i vertikálním směru, a tím dostaneme první derivaci v horizontálním (G_x) a vertikálním (G_y) směru [12]. Z těchto dvou obrazů získáme gradient i směr hrany pro každý pixel následovně:

$$G = \sqrt{G_x^2 + G_y^2} \tag{13}$$

$$\theta = arctan(\frac{G_y}{G_x}) \tag{14}$$

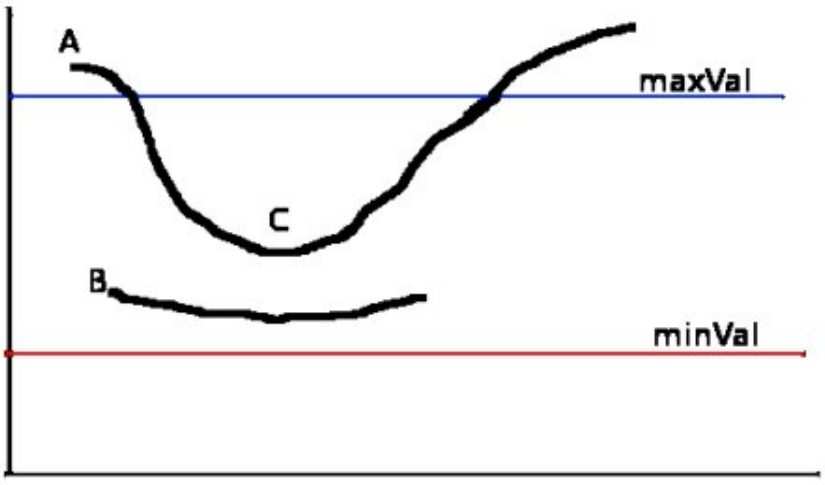
Po získání velikosti a směru gradientu se provede úplné skenování obrazu, aby se odstranily všechny nežádoucí pixely, které nemusí tvořit okraj. Za tímto účelem se na každém pixelu kontroluje, jestli se jedná o lokální maximum v jeho sousedství ve směru gradientu, viz obrázek 19.



Obrázek 19: Princip rozhodování o hraně [12]

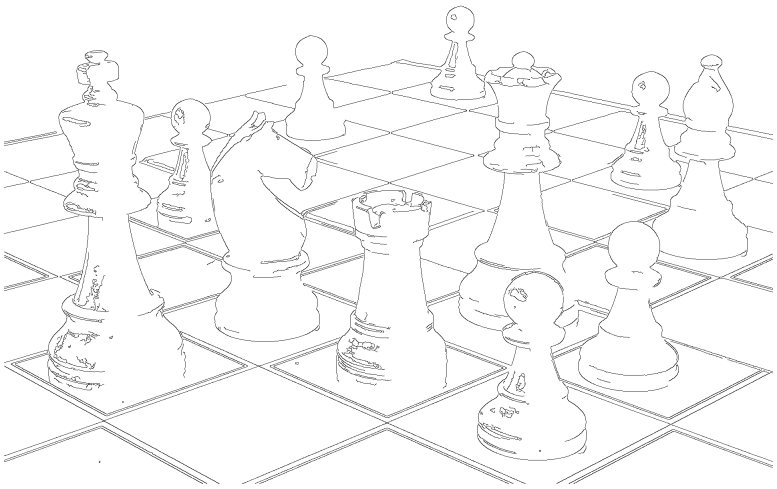
Bod A se nachází na hraně, která svírá kolmici se směrem gradientu. Body B a C jsou ve směru gradientu. Bod A je kontrolován pomocí bodů B a C, aby se zjistilo, zda tvoří lokální maximum. Pokud ano, uvažuje se o další fázi, jinak je bod potlačen. Ve výsledku dostáváme binární obraz s hranami. V poslední fázi se kontroluje, zda-li se opravdu jedná o hrany, nebo ne. K tomuto rozhodnutí potřebujeme dvě prahové hodnoty MIN a MAX, podle kterých se následně rozhodujeme. Je-li gradient intenzity větší než MAX, je hrana ohodnocena jako „silná“ a je považována za hranu. Pokud je gradient intenzity menší než MIN, je hrana automaticky vyřazena. Nakonec, pokud se nachází gradient intenzity hrany mezi hodnotami MIN a MAX, jsou tyto hrany ohodnoceny na základě konektivity k ostatním pixelům. Jestliže je hrana připojena k některému pixelu hrany, která byla ohodnocena jako „silná“, pak se tato hrana považuje za součást hrany. V opačném případě, kdy hrana není napojena na „silnou“ hranu,

je opět vyřazena [12], viz obrázek 20.



Obrázek 20: Hranová hystereze [12]

U této metody se musí zvolit prahové konstanty MIN a MAX pro správnou funkčnost. Doporučuje se poměr hodnot 2:1 nebo 3:1 [13]. Výsledek ukazuje následující obrázek.



Obrázek 21: Obraz po použití funkce Canny

Poté, co jsem vyzkoušel jednotlivé metody, jsem došel k závěru, že základní metoda prahování není použitelná. U mnoha druhů obrazů dochází ke splynutí jednotlivých ploch a následně k získání jen několika významných obrysů. Nejlépe si tato metoda vedla u obrázků, které byly kreslené, což je ale nedostačující. Lépe na tom byla metoda adaptivního prahování. Mnohem lépe detekovala obrysy v obraze a nedocházelo ke slévání ploch. U kreslených objektů metoda obstála bez větších problému, ale u klasických fotografií docházelo k menšímu šumu. Nejlépe dopadl algoritmus canny, který je přímo určený pro detekci hran. Jak u fotografií, tak i u kreslených obrázků dosahoval výborných výsledků. Pouze u kreslených obrázků, kde jsou jed-

notlivé objekty ohraničené černě, dochází ke zdvojení hran. V tomto případě je lepší použít adaptivní prahování.

4.2.3 Získání obrysů

Poté, co jsem získal binární obraz, ve kterém jsou pouze hrany, bylo zapotřebí z tohoto obrazu získat jednotlivé obrysy a k nim příslušnou posloupnost bodů. K tomuto účelu slouží z openCV funkce **getCountours**.

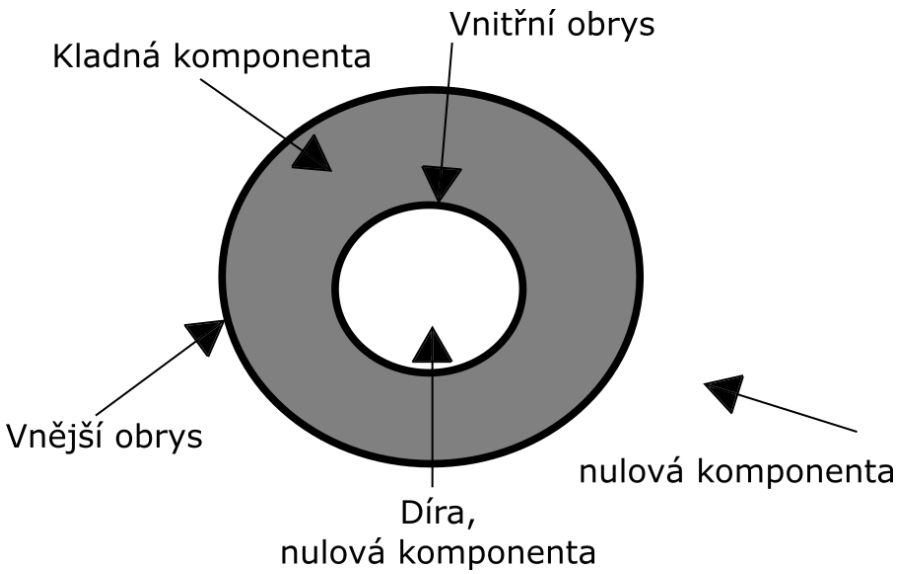
Tato funkce ze zadaného obrazu vrátí seznam jednotlivých nalezených obrysů a k nim příslušnou posloupnost bodů tak, aby mohl být daný obrys z těchto dat vykreslen. Algoritmus, který se skrývá za touto funkcí, je založen na principu Suzukiho algoritmu pro sledování hranic.

Princip Suzukiho algoritmu

Algoritmus funguje na principu sledování hranic, je tedy nutné mít obraz v binárním obraze. *„Pro účely vyhledávání obrysů jsou jednotlivé body obrázku v průběhu algoritmu označovány čísly. Význam číslic je následující“* [13]:

- Body s hodnotou 0 - Jedná se o pozadí nebo díru
- Body s hodnotou 1 - Body ještě nezpracovaných hran
- Body s jinou hodnotu jsou již detekované obrysy

Rozlišujeme dvě komponenty. První komponenta, nulová, je složená z bodů s hodnotou 0. Druhá komponenta je komponenta kladná, a ta se skládá z kladných hodnot. Dále rozdělujeme vnitřní a vnější obrys. Vnější obrys je dán jako skupina hraničních bodů mezi nulovou a kladnou komponentou. Kladná komponenta musí být v tomto případě zcela obklopena nulovou komponentou. Vnitřní obrys je dán jako skupina hraničních bodů mezi dírou a kladnou komponentou. V tomto případě je díra zcela obklopená kladnou komponentou [13, 14].



Obrázek 22: Popis vnitřního a vnějšího obrysu

Průběh algoritmu je pak následovný. Binárním obrazem je procházeno od levého horního rohu po řádcích zleva doprava. Procházení je zastaveno, jakmile se narazí na bod (x, y), u něhož je splněna podmínka pro počáteční bod vnitřní, nebo vnější hrany. Má-li bod (x, y) hodnotu 1 a bod (x-1, y) hodnotu 0, jedná se o vnější obrys. V případě, kdy má bod (x, y) hodnotu ≥ 1 a bod (x+1, y) hodnotu 0, jedná se o vnitřní hranu. Pokud nalezený bod (x, y) splňuje obě tyto podmínky, je považován za vnější hranu. Nakonec je bodu přiřazeno unikátní číslo NBD.

Postupně procházíme a označujeme body nalezeného obrysu. „*Při procházení se udržuje hranice dvou komponent na jedné straně a pokračuje se tak dlouho, než se opět narazí na počáteční bod procházeného obrysu*“ [13]. Během procházení jsou body ohodnocovány dle následujících pravidel:

1. Jestliže je sledovaný obrys mezi nulovou komponentou, která obsahuje bod (p, q+1) a kladnou komponentou, která obsahuje bod (p, q), tak se hodnota bodu (p, q) změní na hodnotu -NBD [14]

2. V ostatních případech se hodnota bodu (p, q) nastaví na hodnotu NBD, pokud bod ještě nebyl označen jako obrys [14].

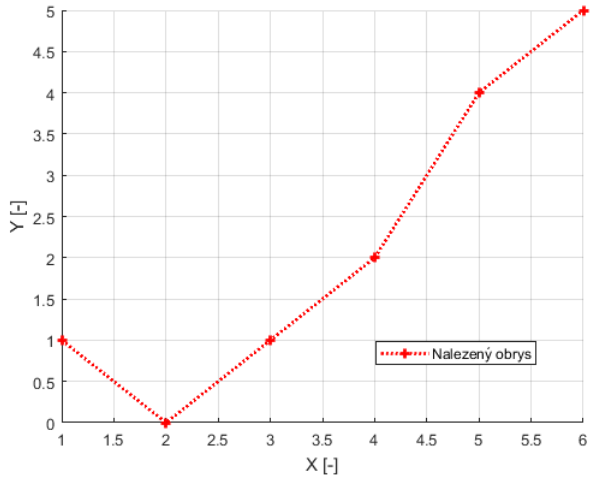
Takto se postupně prochází obraz, dokud se algoritmus nedostane k dolnímu pravému rohu. Po dosažení tohoto bodu algoritmus končí. Výsledné obrysy jsou poté reprezentovány jako posloupnosti bodů.

Optimalizace získaných obrysů

V předchozím bodě jsme získali jednotlivé obrysy popsané posloupností bodů. Tyto obrysy jich však mohou obsahovat příliš mnoho, tudíž by byl výsledný počet GCODE příkazů příliš

velký. Je tedy vhodné každý nalezený obrys ještě optimalizovat. Základní jednoduchou optimalizaci provádí již zmíněná funkce **getCountours**, která, pokud najde přesnou horizontální nebo vertikální úsečku, vrátí pouze počáteční a koncový bod. Pro zbývající obrysy jsem navrhl další optimalizaci.

Každý obrys je prezentován posloupností bodů. Příklad takové posloupnosti může být na následujícím obrázku.



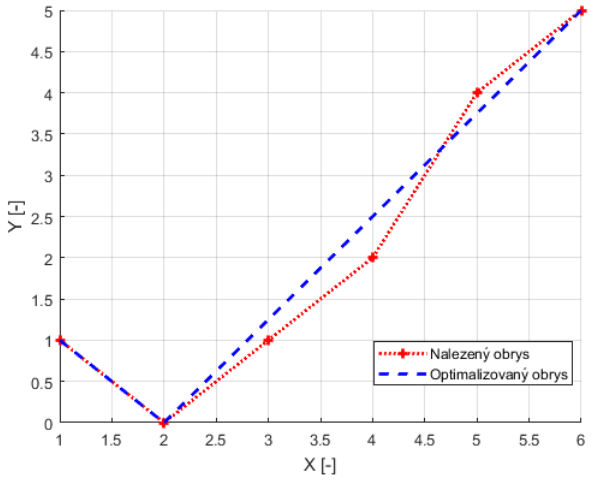
Obrázek 23: Příklad nalezeného obrysu

Z obrázku je patrné, že každý bod je nutné spojit s následujícím bodem úsečkou a nelze zde najít žádnou úsečku, která by procházela všemi body. Můžeme se pokusit alespoň část bodů proložit přímkou, a tím odstranit pár bodů. Tuto optimalizaci si můžeme dovolit, pokud zachováme pouze malou chybu od ostatních bodů. Nejedná se o klasické proložení bodů přímkou. Kvůli návaznosti obrysů musíme zachovat počáteční a koncový bod jak celého obrysu, tak i jednotlivých optimalizovaných úseček.

V algoritmu vezmeme první dva body, které budou tvořit prvotní úsečku, kterou se budeme snažit rozšiřovat. V následujícím kroku se pokusíme vzít další bod, který označíme jako další možný koncový bod úsečky. Z tohoto a počátečního bodu spočteme rovnici přímky a následně spočteme jednotlivé vzdálenosti bodů, které se nachází mezi počátečním a možným koncovým bodem, od přímky.

Pokud se všechny vypočtené vzdálenosti nachází v nízké toleranci, pak označíme námi testovaný koncový bod jako nový koncový bod úsečky. V opačném případě algoritmus skončí a jako počáteční a koncový bod úsečky se nastaví body z předchozí iterace. Poté se algoritmus spouští znovu, přičemž počátečním bodem bude koncový bod předchozí úsečky.

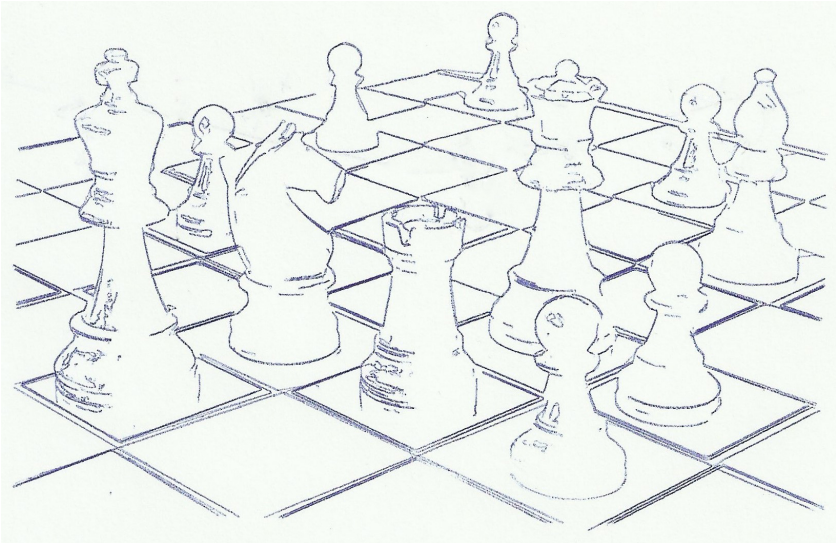
Takto procházíme postupně celý obrys, dokud nenarazíme na konec. Tímto způsobem odstraníme některé body bez újmy na výsledném obraze.



Obrázek 24: Příklad optimalizovaného nalezeného obrysu

4.2.4 Převod do GCODE příkazů

Pomocí předchozích bodů jsem získal souřadnice jednotlivých úseček, které se již pouze převedou do jednotlivých příkazů GCODE. Jelikož na plotteru používám jednotky v milimetrech, rozhodl jsem se použít přepočet 1 mm = 10 px, čímž docílím dobrého výsledného rozlišení. Postupně procházím jednotlivé souřadnice bodů, které následně ukládám ve formátu GCODE do textového souboru. Z počátečního bodu obrysu vím, že se musím na tento bod přesunout a poté spustit pisátko pro kreslení. Dojedu-li na konec obrysu, tak v posledním bodě pisátko vypnu. Takto připravený textový soubor je možné poslat do plotteru k jeho vytištění. Vytištěný finální obrázek vypadá následovně:



Obrázek 25: Výsledný vytištěný obraz

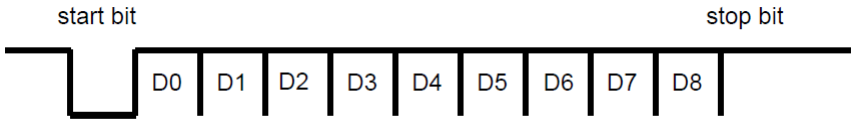
5 Komunikace plotteru s počítačem

5.1 Použité komunikační rozhraní

Pro komunikaci mezi počítačem a plotterem jsem se rozhodl použít sériovou komunikaci pomocí UART přes USB.

UART (Universal asynchronous receiver-transmitter) je součástka, která slouží k asynchronní komunikaci zpravidla za použití dvou pinů RX a TX. Slouží pro přenos dat mezi zařízeními, a to oběma směry, tedy v režimu plný duplex.

Pro přenášení dat se používají rámce. Tyto rámce mohou mít 5-9 bitů a jsou od sebe rozlišeny start bitem a jedním nebo dvěma stop bity [15] dle použité konfigurace viz obr. 26.



Obrázek 26: Uspořádání rámce [15]

U přenosu lze nastavit rychlost od 1200 bps až do 250 kbps [15]. U plotteru jsem zvolil rychlost 115,2 kbps.

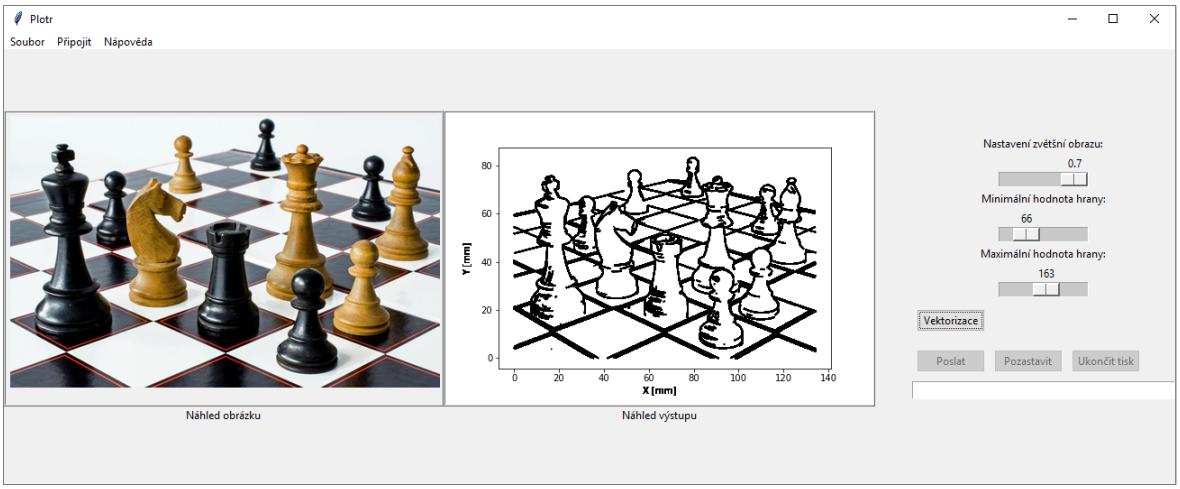
5.2 Princip komunikace

Komunikace mezi plotterem a počítačem je tedy zajišťována sériovou komunikací přes USB. Během posílání jednotlivých příkazů GCODE je třeba kontrolovat, aby se vykonávaly postupně ve správném pořadí. Mezi počítačem a plotterem probíhá jednoduchá komunikace. Z počítače se pošle jeden GCODE příkaz a čeká se na odpověď od plotteru. Plotter přijme příkaz, dekóduje ho a provede příslušné operace. Po úspěšně provedeném příkazu zašle plotter počítači zprávu, po které počítač pošle následující řádek GCODE. Zpráva obsahuje číslo momentálně vykresleného řádku. Tuto komunikaci lze během postupného posílání jednotlivých příkazů pozastavit, případně i zastavit.

5.3 Návrh uživatelského rozhraní

Testování funkčnosti kódu, jak převodu obrázku, tak i komunikace s plotterem, probíhalo čistě přes terminál. Nejedná se tedy o příliš uživatelsky přívětivé řešení. Bylo tedy vhodné navrhnout aplikaci s grafickým rozhraním. Hlavním důvodem vytvoření okenní aplikace byl fakt, že je třeba znát dopředu, jak bude obrázek následně vykreslen na plotteru.

Pro návrh aplikace jsem se rozhodl použít knihovnu **tkinter** [16] pro python. Grafické prostředí je vidět na následujícím obrázku.



Obrázek 27: Náhled ovládací aplikace

V horní části aplikace se nachází menu. Pod položkou „Soubor“ najdeme možnost otevřít obrázek a nebo případně již vytvořený GCODE soubor. V menu je dále položka „Připojit“, která slouží k vybrání COM portu a následného pokusu o připojení. Poslední položka slouží k zobrazení nápovědy k aplikaci.

Z obrázku 27 je vidět, že aplikace má dvě náhledová okna pro obrázek. V prvním okně je zobrazen otevřený obraz pro vektorizaci a ve druhém je pak náhled výsledné vektorizace za pomocí knihovny **matplotlib** pro vykreslování grafů.

V pravé části aplikace se nachází ovládací prvky. Je zde možno nastavit zvětšení/zmenšení výsledného obrazu. Maximální hodnota zvětšení je proměnlivá, neboť je třeba kontrolovat, aby výsledný obraz nebyl v ose X větší než 140 mm. Dále je zde možné nastavit minimální a maximální hodnotu pro Canny algoritmus uvedený ve 4.2.2. Nakonec se zde nachází ovládací tlačítka pro sériovou komunikaci a textové okno, ve kterém se vypisují právě prováděné GCODE příkazy.

Ovládání aplikace je vcelku intuitivní. Nejdříve je třeba vybrat, jestli chceme použít již vygenerovaný GCODE soubor, nebo vytvořit nový z obrázku. Pokud jsme si vybrali obrázek, který chceme vykreslit na plotteru, tak pomocí tlačítka „Vektorizace“ připravíme vektorizovaný obraz. Jestliže nejsme s výstupem spokojeni, je možné výsledný obraz zvětšit/zmenšit, případně zkusit použít jinou konfiguraci Canny algoritmu pomocí posuvníků pro nastavení minimální a maximální hodnoty hrany. To se hodí, pokud ve výsledném obrazu nejsou vidět některé, zpravidla tenčí, hrany. Abychom se pokusili je zviditelnit, stačí posouvat oběma posuvníky směrem dolů, přičemž je vhodné zachovat doporučený poměr 1:2, 1:3. Takto vektorizovaný soubor je připraven k vykreslení na plotteru. Nejdříve je třeba připojit se k plotteru přes položku „Připojit“. Zde vybereme příslušný COM port a stiskneme tlačítko připojit. Pokud připojení proběhlo úspěšně, objeví se hláška o úspěšném připojení a zároveň se vyresetuje plotter. Poté stačí daný obraz poslat. Obdobný postup je v případě otevření již vygenerovaného GCODE souboru. V tomto případě aplikace nabízí pouze poslání souboru do plotteru.

6 Závěr

Cílem práce bylo navrhnout způsob ovládání plotteru, konstrukci pisátka a také aplikaci pro převod obrazu tak, aby jej poté bylo možné plotterem vytisknout. Z počátku jsem se rozhodoval, jak přistoupit k ovládání a pozicování plotteru. Jestli použít jedinečnou sadu příkazů, která by řešila pouze komunikaci mezi počítačem s daným plotterem, nebo se snažit použít sady příkazů GCODE. Během vývoje programu pro ovládání plotteru se již z počátku ukázalo, že bude vhodnější použít sadu příkazů GCODE tak, aby byl plotter schopen číst vygenerované GCODE soubory, například z aplikace Inkscape. Ovládací program je nakonec schopný číst tuto sadu příkazů a vykonávat patřičné instrukce.

Při návrhu pisátka bylo třeba vyzkoušet více návrhů a vybrat ten nejefektivnější a zároveň nejkompaktnější. Během testování výsledných výtisků docházelo k problémům s přesností vytištěných součástí, a bylo tedy nutné součásti doupravovat. To také nakonec vedlo k drobným vůlím pisátka. Tyto vůle v konečné fázi neměly příliš velký dopad na výsledný výtisk.

V poslední části jsem se zabýval návrhem aplikace a řešením zpracování obrazu tak, aby jej následně bylo možné jednoduše vytisknout na plotteru. Pro řešení jsem se rozhodl použít funkce z knihovny **openCV**. Díky nim jsem byl poté schopný v obraze detekovat hrany a následně získat body obrysů, které mohly být dále zpracovány do formátu GCODE příkazů. Pro vektorizaci bylo vhodné zvolit obrazy v dobrém rozlišení. Zároveň jsem během testování zjistil, že některé druhy obrázků nejsou vhodné pro vektorizaci. zejména se jednalo o příliš složité obrazy jako například obrazy přírody. V těchto případech bylo ve výsledném náhledu výkresu tolik čar, že bylo obtížné v obraze rozlišit jednotlivé objekty. Celkový návrh uživatelsky přívětivého prostředí pak probíhal za pomocí knihovny **tkinter**. V konečné fázi jsem byl tedy schopný načíst obraz, ten posléze vektorizovat a nakonec ho nechat plotterem vykreslit.

Literatura

[1] Arduino Documentation. *Arduino* [online]. Dostupné z: https://www.arduino.cc/reference/en

[2] LUBOŠ M. Driver krokových motorů A3967. *Arduino návody* [online]. [cit. 2020-04-19]. Dostupné z: https://navody.arduino-shop.cz/navody-k-produktum/driver-krokovych-motoru-a3967.html

[3] Zapojení vinutí. *Microcon* [online]. [cit. 2020-05-03]. Dostupné z: http://www.microcon.cz/zapojenivinuti2012web/vinutihlstranka14.asp

[4] SERVO MOTOR SG90 DATASHEET. *Department of Electrical and Electronic Engineering* [online]. [cit. 2020-04-19]. Dostupné z: http://www.ee.ic.ac.uk/pcheung/teaching/DE1_EE/stores/sg90_datasheet.pdf

[5] VODA, Zbyšek. *Průvodce světem Arduina. 2.* Bučovice: Martin Stříž, 2015. ISBN 978-80-87106-90-7.

[6] G-code. *Wikipedia* [online]. [cit. 2020-04-19]. Dostupné z: https://en.wikipedia.org/wiki/G-code

[7] BRESENHAM Jack E. Algorithm for computer control of a digital plotter. *IBM Systems Journal*. vol. 4. 1965. stránky 25-30

[8] Rasterizace úsečky. *IT Network* [online]. [cit. 2020-04-20]. Dostupné z: https://www.itnetwork.cz/navrh/algoritmy/algoritmy-graficke/algoritmus-grafika-rasterizace-usecky/

[9] W. RANDOLPH Franklin. *Bresenham Line and Circle Drawing* [online]. [cit. 2020-04-23]. Dostupné z: https://wrf.ecse.rpi.edu/Research/Short_Notes/bresenham.html

[10] Make G02 & G03 Easy, Avoid Mistakes. *CNC CookBook* [online]. [cit. 2020-04-24]. Dostupné z: https://www.cnccookbook.com/cnc-g-code-arc-circle-g02-g03/

[11] Cnc Radius Programming. *Cnc Philosophy* [online]. [cit. 2020-04-26]. Dostupné z: https://cncphilosophy.com/circular-interpolation-in-cnc/

[12] Open Source Computer Vision. *OpenCV* [online]. [cit. 2020-04-19]. Dostupné z: https://docs.opencv.org/master/index.html

[13] FRYDRYCH Jan. *Rozpoznávání dopravních značek pro řízení modelu auta.* Ostrava, 2017. Diplomová práce. Technická univerzita Ostrava.

[14] Suzuki, Satoshi a Abe, Keiichi. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*. 1985. stránky 32-46. ISSN 0734189x

[15] UART. *Zavavov* [online]. [cit. 2020-04-28]. Dostupné z: http://www.zavavov.cz/cz/elektrotechnika/komunikacni-sbernice/67-uart-usart-komunikujte-seriove-po-dvou-vodicich/

[16] Tkinter Documentation. *TkDocs* [online]. [cit. 2020-05-02]. Dostupné z: https://tkdocs.com/index.html

Příloha A: Obsah přiloženého CD

BP_Vlasák.pdf - Bakalářská práce

arduino_plotter.cpp - Zdrojový kód pro ArduinoDue k ovládaní plotteru

app_plotter.py - Zdrojový kód okenní aplikace pro vektorizaci a komunikaci s plotterem