Česká zemědělská univerzita v Praze

Provozně ekonomická fakulta

Katedra informačního inženýrství

Diplomová práce

Řídící jednotka pro dvou pístové čerpadlo s lineárním průtokem

Bc. Jan Severyn

© 2020 ČZU v Praze

ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Jan Severyn

Systémové inženýrství a informatika
Informatika

Název práce

Řídící jednotka pro dvoupístové čerpadlo s lineárním průtokem

Název anglicky

Control unit for two-piston linear flow pump

Cíle práce

Cílem práce je vytvořit program pro řízení dvoupístového čerpadla s využitím platformy Arduino. Součástí práce je také vytvoření funkčního prototypu řídící jednotky.

Metodika

Pro dosažení cíle práce bude využita platforma Arduino. Chování řadiče bude popsáno za pomoci jazyka C s využitím předpřipravených knihoven pro práci s externími prvky připojenými k mikrořadiči. Mikrořadič bude komunikovat především s krokovým motorem ovládajícím čerpadlo, tlakoměrem a dalšími senzory. Využity budou také zobrazovací zařízení, ovládací prvky a propojení s PC.

Doporučený rozsah práce

50-60 stran

Klíčová slova

arduino, čerpadlo, řidící jednotka, krokový motor, jazyk C

Doporučené zdroje informací

BELL, C A. *Beginning sensor networks with Arduino and Raspberry Pi.* [New York, New York]: Apress, 2013. ISBN 1430258241.

PURDUM, J. Beginning C for Arduino, Second Edition: Learn C Programming for the Arduino. Apress; 2nd ed. edition, 2015. ISBN 978-1484209417

Selecký, M. Arduino – uživatelská příručka. Computer Press, 2016. ISBN 978-80-251-4840-2

VODA, Z. Průvodce světeme Arduina.Bučovice Nakladatelství Martin Stříž, 2017. ISBN: 978-80-87106-93-8

Předběžný termín obhajoby

2019/20 ZS – PEF (únor 2020)

Vedoucí práce

Ing. Marek Pícka, Ph.D.

Garantující pracoviště

Katedra informačního inženýrství

Elektronicky schváleno dne 24. 1. 2019

Ing. Martin Pelikán, Ph.D.

Vedoucí katedry

Elektronicky schváleno dne 24. 1. 2019

Ing. Martin Pelikán, Ph.D.

Děkan

V Praze dne 06. 04. 2020

Čestné prohlášení

Prohlašuji, že svou diplomovou práci " Řídící jednotka pro dvou pístové čerpadlo s lineárním průtokem" jsem vypracoval(a) samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou citovány v práci a uvedeny v seznamu použitých zdrojů na konci práce. Jako autor(ka) uvedené diplomové práce dále prohlašuji, že jsem v souvislosti s jejím vytvořením neporušil autorská práva třetích osob.

V Praze dne 6.4.2020 _____

Poděkování

Rád(a) bych touto cestou poděkoval(a) Ing. Marku Píckovi, Ph.D. za odborné vedení práce, věcné a odborné připomínky. Mé poděkování patří také Petru Severynovi za námět práce a konzultace během řešení práce.

Řídící jednotka pro dvou pístové čerpadlo s lineárním průtokem

Abstrakt

Diplomová práce se zabývá tvorbou programu pro řídící jednotku dvou pístového čerpadla poháněného bipolárním krokovým motorem. Jádrem řídící jednotky je vývojová deska Arduino UNO ovládající krokový motor čerpadla dle uživatelem nastavených parametrů komunikující s uživatelem pomocí LCD displeje nebo sériové linky. V rámci úvodní rešerše práce je ve stručnosti popsáno současné známé řešení ovládání čerpadla, současné možnosti v oblasti vývojových desek Arduino, řízení krokových motorů i možnosti využití zobrazovacích zařízení s vývojovými deskami Arduino. Hlavní část práce je rozdělena na čtyři dílčí části. Nejprve je popsána analýza funkčních a nefunkčních požadavků, včetně popisu případů užití na základě které, byla navrhnuta architektura cílového řešení. Druhou popisovanou částí je vytvoření programu pro vývojovou desku Arduino UNO a popis vytvoření prototypu řídící jednotky připojené k motoru čerpadla pomocí nepájivého pole. Konkrétně je v této části uveden popis jednotlivých komponent řešení a popis SW řešení. Následně je popsán rozsah, průběh a výsledek testování implementovaného řešení a na závěr hlavní části je uvedena cenová kalkulace navrhovaného a implementovaného řešení. Na konci práce je uvedeno celkové zhodnocení a popis dalšího možného rozvoje práce.

Klíčová slova: arduino, čerpadlo, řídící jednotka, krokový motor, jazyk C, bipolární krokový motor, displej, sériová linka, Arduino UNO

Control unit for two-piston linear flow pump

Abstract

The thesis describe creating software for control unit of two-piston pump driven by bipolar stepper motor. Arduino UNO is used as the core of the control unit, which control stepper motor of pump based on user defined parameters. Control unit communicate with user using LCD display or serial line.

In literary research of this thesis are briefly describe current known pump control solution, current options in Arduino development boards, stepper motor controls and options for using displays with Arduino development boards. The main part of this thesis is divided to four subsections. At first functional and non – fuctional requirements analysis is describe. It includes Use-Case description also. Based on this analysis was created architecture design of goal solution. Second described subsection is creating software for Arduino UNO and description of creating of prototype of control unit using breadboard. Specifically, descripotion of each compoment of the solution and description of SW solution is describe in this subsection. Then the testing scope, testing process and testing results of implemented solution is described. At the end of main part solution pricing is describe. Evalution of solution and description of possible further development is given at end of this thesis.

Keywords: Arduino, pump, control unit, stepper motor, C language, bipolar stepper motor, display, serial line, Arduino UNO

Obsah

Seznam obrázků

Seznam tabulek

Seznam příloh

1 Úvod

Firma Welco vyrábí a prodává čerpadla poháněné různorodými druhy motorů. Jedním z typů motorů, která pro svůj pohon využívá čerpadlo WP1000 je bipolární krokový motor. Čerpadla jsou osazena pouze motorem bez programové logiky. Firma Watrex tato čerpadla využívá pro vytvoření čerpadla včetně programové logiky pro kontinuální čerpání. Nevýhodou těchto čerpadel jsou vyšší náklady na výrobu nemožnost využití více režimů čerpání.

V rámci práce se budu věnovat využití vývojové desky Arduino k sestavení řídící jednotky pro čerpadlo od firmy Welco, které je poháněné bipolárním krokovým motorem. Hlavním cílem práce je vytvořit takové program pro vývojovou desku Arduino, který umožní spuštění čerpadla ve dvou režimech čerpání s možností nastavení objemu a průtoku, umožní kalibraci čerpadla a nastavení výchozích hodnota. Veškeré funkce čerpadla bude možné ovládat pomocí LCD displeje a tlačítek nebo pomocí připojeného počítače. Řešení bude sestaveno do funkčního prototypu pomocí nepájivého pole.

Výstupy práce budou analýza požadavku, která bude předcházet samotné implementaci řešení, otestovaný a funkční program pro vývojovou desku Arduino, sestavený prototyp řešení a cenová kalkulace implementovaného řešení, kterou bude možné porovnat s kalkulací současného řešení.

Text práce má následující strukturu. V druhé kapitole definuji Cíle práce a popisuji metodiku použitou pro dosažení těchto cílů. Třetí kapitola se zabývá popisem současné problematiky a literární rešerší možností řízená krokových motorů, možností využití zobrazovacích zařízení a popisem jednotlivých typů desek Arduino. Čtvrtá kapitola je jádrem celého řešení. Obsahuje analýzu požadavků na implementaci řídící jednotky, popis implementovaného řešení, popis průběhu a výsledků testování celého řešení a cenovou kalkulaci řešení. V rámci páté kapitoly popisuji výsledný stav práce, hodnotím naplnění definovaných cílů a uvádím další možnosti rozvoje implementovaného řešení. V poslední kapitole stručně shrnuji výsledky a průběh celé práce.

2 Cíl práce a metodika

2.1 Cíl práce

Cílem práce je vytvořit program pro řízení dvou pístového čerpadla WP1000 od firmy Welco s využitím platformy Arduino. Vytvořená řídící jednotka bude implementovat níže popsané funkcionality, které budou podrobněji popsány v analytické fázi práce.

Řídící jednotka by měla umožnit ovládat čerpadlo a poskytovat jednoduché uživatelské rozhraní pomocí LCD displeje a vstupních ovládacích prvků (tlačítek nebo klávesnice) a zároveň pomocí sériové linky po připojení jednotky k PC pomocí USB rozhraní.

Čerpadlo bude možné spustit ve dvou základních režimech průběžného a přesného čerpání s nastavenými parametry průtoku a objemu. Výchozí nastavení těchto parametrů bude možné uložit do trvalé paměťově nezávislé paměti.

Řídící jednotka bude obsahovat funkci pro kalibraci pro správné nastavení převodových hodnot mezi rychlostí motoru a průtokem.

Nedílnou součástí práce je také vytvoření funkčního prototypu řídící jednotky, jehož jádrem bude vývojová deska Arduino. Výběr vhodné desky bude proveden v rámci analytické části práce vzhledem k definovaným parametrům.

Součástí práce také bude cenová kalkulace nákladů navrhovaného řešení.

Nově vytvořená řídící jednotka by mohla sloužit jako alternativa k čerpadlům, která jsou dnes vyráběny a využívána firmou Watrex.

2.2 Metodika

Pro dosažení cíle práce bude využita platforma Arduino. Chování řadiče bude popsáno za pomoci jazyka C s využitím předpřipravených knihoven pro práci s externími prvky připojenými k mikrořadiči. Mikrořadič bude komunikovat především s krokovým motorem ovládajícím čerpadlo, zobrazovacím zařízením a ovládacími prvky.

Pro propojení s PC bude využito UART rozhraní vývojové desky a propojení s PC bude zajištěno pomocí USB rozhraní.

Základem pro sestavení celého prototypu řadiče bude nepájivé pole a součástí práce bude detailní schéma zapojení celého řešení.

Pro cenovou kalkulaci budou využity informace o cenách na českém trhu v době tvorby řešení.

3 Současný stav poznání řešené problematiky

V následujícím textu nejdříve popíšu aktuální stav řešené problematiky. Konkrétně se zaměřím na v současné době využívané řešení pro řízení čerpadla pomocí bipolárního krokového motoru a příležitosti pro vytvářené řešení. Dále představím jednotlivé možnosti řízení krokového motoru a možnosti využití zobrazovacích zařízení vzhledem k potřebám této práce. Na závěr kapitoly popíšu vývojové desky Arduino a jejich různé varianty.

3.1 Současné využívané řešení

Firma Watrex v současné době produkuje čerpadlo (včetně řídící logiky) DeltaChrom™ Membrane Suppression Unit MSU 010 (Watrex) . Čerpadlo pracuje na jednoduchém principu. Jádrem čerpadla je řadič krokového motoru vlastní výroby. Změnou rychlosti otáček tohoto motoru dochází ke změně průtoku čerpadla. Veškeré nastavení probíhá prostřednictvím připojení řadiče k PC. Zmíněné čerpadlo neobsahuje LCD displej.

3.1.1 Omezení architektury

Architektura současného řešení neumožňuje realizaci rozdílných využití čerpadla v rámci jedné řídící jednotky. V současné době řešení podporuje pouze režim kontinuálního čerpání. Čerpadlo není možné využít pro přesné čerpání. Stručný popis rozdílů dvou zmíněných režimů uvádím zde:

- **Kontinuální čerpání** – slouží pro nepřerušené čerpání kapaliny s definovaným průtokem určitou, ne vždy předem známou, dobu
- **Přesné čerpání –** slouží pro přesné dávkování předem určeného množství kapaliny

3.1.2 Možnosti rozvoje řešení

Rozvoj řídících jednotek je v současném situaci omezen na jednoho firemního specialistu, který má vývoj řídících jednotek na starost. Existuje tu tedy riziko vzniku tzv. **Vendor locku** [1]a pro rozvoj je omezena zastupitelnost.

3.1.3 Cenová kalkulace současného řešení

V následující tabulce (Tabulka 1) uvádím kalkulaci nákladů současného vyráběného řešení, které jsem obdržel od zástupce firmy Watrex.

Tabulka 1: Cenová kalkulace současného řešení

Název dílu	Cena (včetně DPH)
Řadič krokového motoru (vlastní výroba)	4000,-
Konektory, LED, vnitřní kabeláž	100,-
DC adaptér 24 V, 2.7A	650,-

[1] Vendor lock je označení situace, kdy se zákazník při potřebě úpravě systému nebo SW nemůže vymanit ze závislosti na konkrétním dodavateli. (Látal, 2018)

Stolní pouzdro ABS,	1700, -
Výroba čelního panelu	250,-
Suma	**6700**, -
Suma bez boxu	**4750, -**

Zdroj: vlastní sestavení na základě podkladů do Watrex

3.1.4 Předpokládané výhody a rozdíly navrhovaného řešení

Navrhované řešení, jehož analýza a popis realizace je podrobněji popsána v následujících kapitolách předpokládá výhody uvedené v tabulce níže:

Obrázek 1: Předpokládané výhody a rozdíly navrhovaného řešení

Výhoda	Popis
Dva režimy čerpání	Nová řídící jednotka bude podporovat oba režimy čerpání.
Komfortní ovládání	Navrhované řešení předpokládá využití LCD displeje a tlačítek pro nastavení a ovládání čerpadla.
Rozvoj řešení	Rozvoj nového řešení bude možně z velké části provést pouze úpravou programu v jazyce C, využívající veřejně dostupné knihovny pro platformu Arduino.
Náklady na řešení	Nové řešení, předpokládá úsporu nákladů na použité komponenty.

3.2 Možnosti řízení krokových motorů

Vzhledem k tomu, že základním úkolem řídící jednotky čerpadla bude ovládání krokového motoru, představíme si v této kapitole nejdříve způsoby řízení krokového motoru a následně možné způsoby implementace.

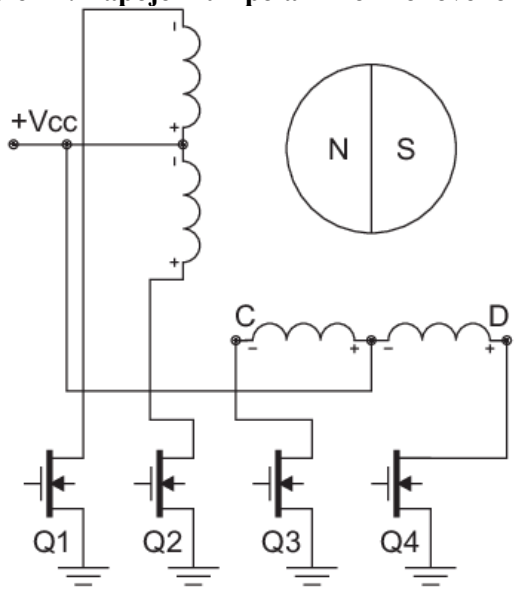
3.2.1 Způsoby řízení

Jak uvádí Petr Maňák ve své práci (Maňák, 2014), řízení krokových motorů spočívá ve spínání fázových proudů v předem určených sekvencích, které se liší dle počtu fází motoru a jejich zapojení. Základní zapojení krokového motoru může být bipolární nebo unipolární.

Autor dále uvádí, rozdíly řízení těchto způsobů zapojení. Při unipolárním zapojení motoru prochází proud jednotlivými cívkami pouze jedním směrem. Pro zapojení řídící elektroniky pro čtyř fázový krokový motor stačí tedy pouze 4 tranzistory, které spínají proud jednotlivými fázemi. Schéma unipolárního zapojení krokového motoru je znázorněno na obrázku - Obrázek 2. Každá cívka je spínána jedním z tranzistorů Q1 – Q4.

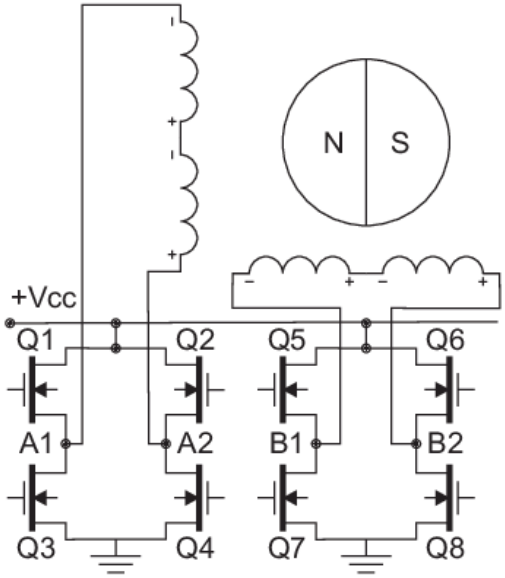
U bipolárního zapojení může proud cívkami protékat oběma směry. Řídící jednotka pro bipolární zapojení motoru je tak tedy složitější. Každou fází (cívku) je nutné ovládat pomocí H-můstku, který se skládá ze čtyř tranzistorů. Pro dvou fázový bipolárně zapojený krokový motor je potřeba celkem 8 tranzistorů. Schéma bipolárního zapojení pomocí H-můstku je znázorněno na obrázku - Obrázek 3. Jednotlivé cívky jsou spínány vždy dvojicí tranzistorů dle potřebného směru proudu (Q1 a Q4, Q2 a Q3, Q5 a Q8 nebo Q6 a Q7)

Obrázek 2: Zapojení unipolárního krokového motoru



Zdroj: Leuchter, 2018

Obrázek 3: Zapojení bipolárního krokového motoru



Zdroj: Leuchter, 2018

Existuje více možností postupného spínání jednotlivých cívek krokového motoru:

- Čtyřtaktní s jednou aktivní fází
- Čtyřtaktní s dvěma aktivními fázemi
- Osmitaktní

3.2.1.1 Čtyřtaktní řízení s jednou aktivní fází

Jedná se nejjednodušší z uvedených principů. V každém ze čtyř střídajících se taktů dochází k sepnutí vždy pouze jedné cívky. U unipolárního zapojení dochází ke střídání sepnutých cívek a postupně dochází ke spínání vždy pouze jednoho tranzistoru. U bipolárního řízení dochází ke střídání sepnutí cívek včetně střídání jejich polarity. Dochází postupně ke spínání dvojic tranzistorů. Popis spínání tranzistorů pro jednotlivé takty popisuje tabulka níže, ve které využívám označení tranzistorů z obrázků Obrázek 2 a Obrázek 3. (Maňák, 2014)

Tabulka 2: Spínání tranzistorů – čtyřtaktní řízení s jednou aktivní fází

Číslo taktu	**Sepnuté tranzistory**	
	Unipolární zapojení	Bipolární zapojení
1	Q1	Q1 a Q4
2	Q2	Q5 a Q8
3	Q3	Q2 a Q3
4	Q4	Q6 a Q7

Zdroj: vlastní zpracování získaných dat (Maňák, 2014)

3.2.1.2 Čtyřtaktní řízení s dvěma aktivními fázemi

U tohoto způsobu řízení jsou v každém ze čtyřtaktu běhu motoru aktivní dvě cívky. Dochází tak vždy k sepnutí dvou tranzistorů v případě unipolárního zapojení a čtyři cívky při zapojení bipolárním. Tento způsob řízení umožňuje vyšší krokovací frekvenci, ale zvyšuje spotřebu motoru. Rotor je vychýlen vždy o polovinu velikosti kroku.

Popis spínání tranzistorů pro jednotlivé takty popisuje tabulka níže, ve které využívám označení tranzistorů z obrázků Obrázek 2 a Obrázek 3. (Maňák, 2014)

Tabulka 3: Spínání tranzistorů – čtyřtaktní řízení s dvěma aktivními fázemi

Číslo taktu	**Sepnuté tranzistory**	
	Unipolární zapojení	Bipolární zapojení
1	Q1, Q2	Q1 a Q4, Q5 a Q8
2	Q2, Q3	Q5 a Q8, Q2 a Q3
3	Q3, Q4	Q2 a Q3, Q6 a Q7
4	Q4, Q1	Q6 a Q7, Q1 a Q4

Zdroj: vlastní zpracování získaných dat (Maňák, 2014)

3.2.1.3 Osmitaktní řízení

Poslední z uvedených způsobů řízení je kombinací obou předchozích způsobů. Střídavě jsou v taktech sepnuty dvě a jedna cívka. Výhodou tohoto řízení dvojnásobení počtu kroků motoru. Docílíme tím zmenšení kroku motoru na polovinu.

Popis spínání tranzistorů pro jednotlivé takty popisuje tabulka níže, ve které využívám označení tranzistorů z obrázků Obrázek 2 a Obrázek 3. (Maňák, 2014)

Tabulka 4: Spínání tranzistorů – osmitaktní řízení

Číslo taktu	**Sepnuté tranzistory**	
	Unipolární zapojení	**Bipolární zapojení**
1	Q1, Q2	Q1 a Q4, Q5 a Q8
2	Q1	Q1 a Q4
3	Q4, Q1	Q6 a Q7, Q1 a Q4
4	Q4	Q6 a Q7
5	Q3, Q4	Q2 a Q3, Q6 a Q7
6	Q3	Q2 a Q3
7	Q2, Q3	Q5 a Q8, Q2 a Q3
8	Q2	Q5 a Q8

Zdroj: vlastní zpracování získaných dat (Maňák, 2014)

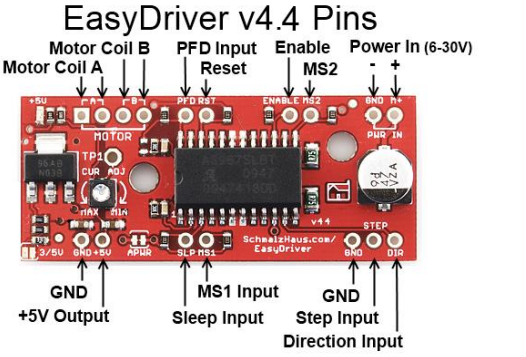
3.2.2 Možnosti implementace řízení krokového motoru

Pro implementaci řízení krokového motoru prostřednictvím Arduino je možné využít několik způsobů. V rámci své práce jsem narazil a porovnával možnosti, které stručně popíšu v následujících podkapitolách. Vzhledem že v mé práci budu pracovat s bipolárním krokovým motorem zaměřil jsem se na implementace řízení bipolárních motorů.

3.2.2.1 Řadič krokového motoru EasyDriver A3967

Pro velice jednoduché řízení bipolárních krokových motorů je možné využít modul EasyDriver. Řízení krokových motorů je zajištěno pomocí integrovaného obvodu A3967. Popis řadiče je zobrazen na obrázku - Obrázek 4.

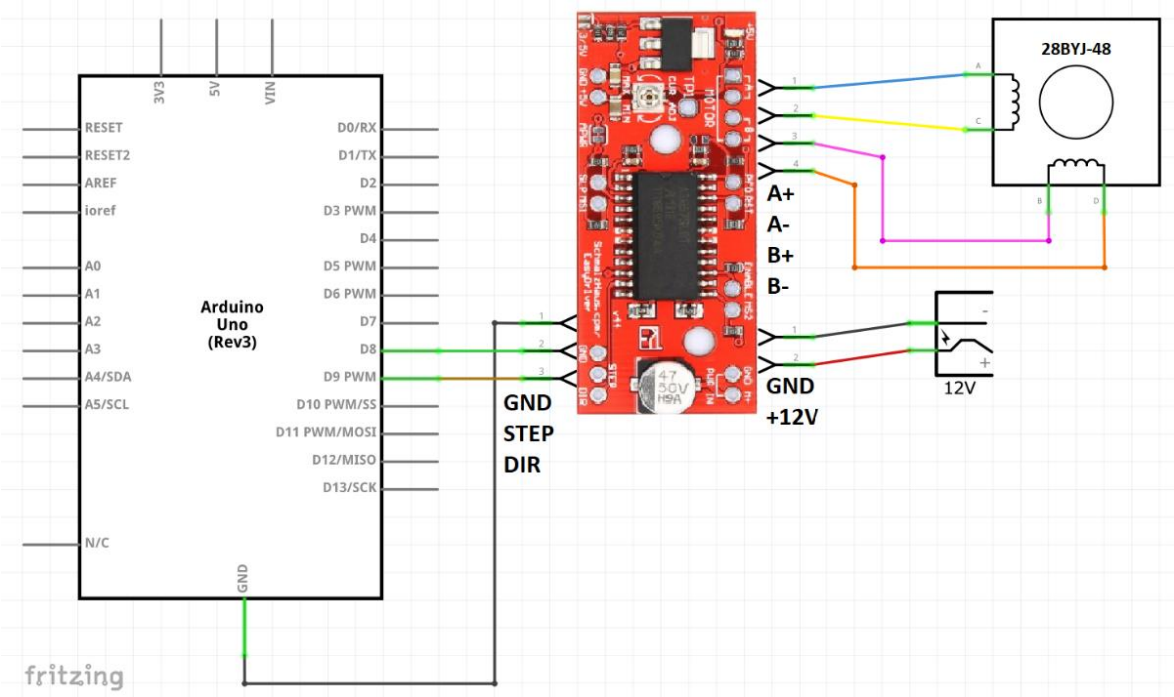
Obrázek 4: Řadič krokového motoru EasyDriver v4.4



Zdroj: Schmalz, 2015

Zapojení i ovládání tohoto řadiče je velice jednoduché. Příklad využití tohoto řadiče je uveden například v článku na stránce navody.arduino-shop.cz (Driver krokových motorů A3967, 2019) a znázorněn na obrázku - Obrázek 5. Motor se připojí pomocí 4 pinů označených jako A+, A-, B+ a B-. Deska Arduino se propojí s řadičem pomocí pinů STEP, DIR a GND. Nakonec je k řadiči nutné připojit napájení.

Obrázek 5: Příklad zapojení řadiče EasyDriver



Zdroj: (Driver krokových motorů A3967, 2019)

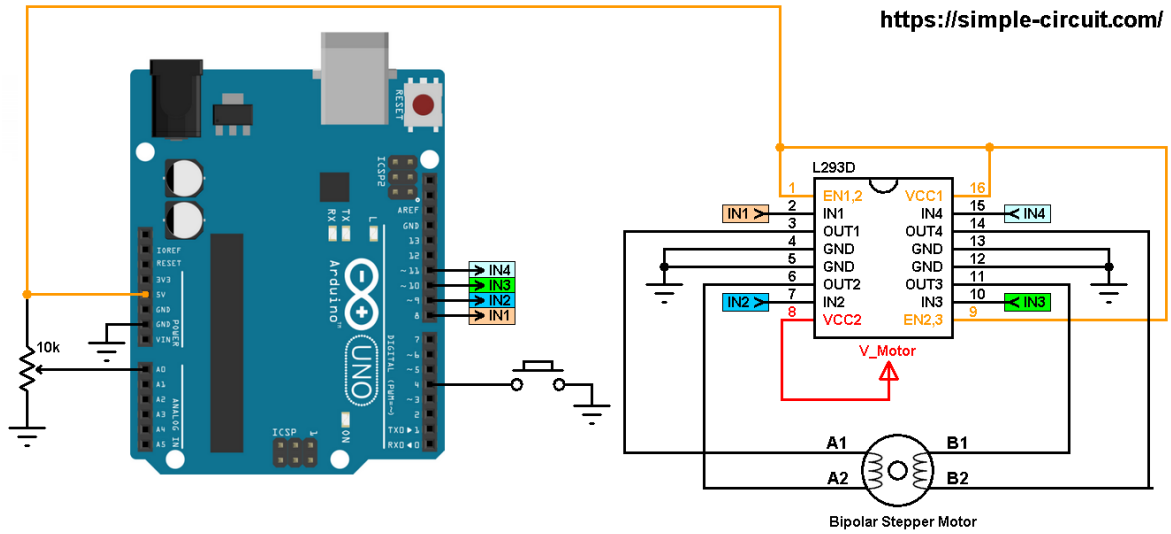
Krokový motor se následně ovládá pouze pomocí pinu STEP. Každá náběžná hrana signálu způsobí, že se motor otočí o jeden krok. Pomocí pinu DIR se pouze přepíná směr otáčení krokového motoru (Driver krokových motorů A3967, 2019)

Výhodou tohoto řešení je jednoduchost zapojení i ovládání. Nevýhodou je menší kontrola nad řízením motoru. Nemáme možnost zvolit způsob řízení motoru popsaných v kapitole 3.2.1. Další nevýhodou je nemožnost jednoduchého nastavení rychlosti motoru.

3.2.2.2 H-můstek s využitím knihovny Stepper

Jak jsem zmínil v kapitole 3.2.1 pro řízení bipolárních krokových motorů je možné využít h-můstek. Schéma zapojení s využitím h-můstku (v tomto případě konkrétně obvod L293D) je zobrazeno na obrázku - Obrázek 6. Kromě napájecích pinů je z Arduino nutné propojit 4 konektory na vstupní piny IN1 – IN4. Krokový motor je pak připojen pomocí pinů OUT1 – OUT4. Při přivedení signálu na jeden z pinů IN1 – IN4 dojde k sepnutí dvojice tranzistorů dle principu popsaného v kapitole 3.2.1.

Obrázek 6: Zapojení bipolárního krokového motoru s h-můstkem



Zdroj: Arduino Bipolar Stepper Motor Control, 2018

K ovládání takto zapojeného motoru je pak možné využít integrovanou knihovnu Stepper (Gassner, 2019). Pomocí konstruktoru **Stepper (steps, pin1, pin2, pin3, pin4**) je nejprve nutné vytvořit nový objekt motoru. Argument „steps“ udává kolik kroků motor musí udělat pro otočení o 360°. Argumenty „pin1“ – „pin4“ slouží k definici pinů na které je motor prostřednictvím h-můstku připojen.

Po vytvoření objektu motoru je možné pomocí metody **setSpeed(rpm)** nastavit rychlost krokového motoru v otáčkách za minutu. Pomocí zavolání metody **step(steps)** následně pohneme motorem o počet kroků uvedeném v argumentu *steps*.

Výhodou tohoto zapojení zůstává stejně jako u použití EasyDriveru celkem jednoduché ovládání a také cena. Cena h-můstku je oproti EasyDriveru výrazně nižší. Nevýhodou může být poněkud složitější zapojení a zároveň také nemáme možnost ovlivnit způsob řízení motoru. Knihovna Stepper řídí motor čtyř-taktně s dvěma aktivními fázemi (viz kapitola 3.2.1.2).

3.2.2.3 H-můstek s využitím vlastní implementace

Poslední metoda využívá stejného zapojení jako je uvedeno v předchozí kapitole 3.2.2.2. Místo knihovny Stepper se však zaměřím na možnost vlastní implementace řízení krokového motoru. Ve článku Arduino – řízení krokového motoru (Syrový, 2014) kde se autor zabýval řízením unipolárního krokového motoru je zmíněno, že při použití knihovny Stepper pohyb motoru nebyl plynulý.

Je možné tedy pro řízení motoru využít vlastní implementaci podobně jako Petr Maňák ve své práci (Maňák, 2014). Výhodou je stejně jako v přechozí kapitole cena tohoto řešení. Druhou výhodou je možnost vybrat si libovolný způsob řízení motoru (viz kapitola 3.2.1.). Zásadní nevýhodou je však implementační náročnost takového řešení. V případě tohoto řešení je nutné vytvořit programovou logiku spínaní jednotlivých fázi včetně nastavení rychlosti. Jelikož cílem této práce není implementace různých režimů řízení krokových motorů pravděpodobně bude vhodnější zvolit jednu z předchozích variant řešení.

3.3 Možnosti využití zobrazovacích zařízení

3.3.1 Typy zobrazovacích zařízení

V první kapitole vás nejprve seznámím s různými typy displejů, které by bylo možné pro mé řešení využít. V rámci kapitoly se nebudu věnovat segmentovým displejům a led maticím, které umožnují zobrazení jen základních informací v malém rozsahu a pro připravované řešení by nebyly vhodné. Pro každý typ displeje uvedu stručnou charakteristiku a jeho výhody a nevýhody.

3.3.1.1 Alfanumerické LCD displeje

Alfanumerické displeje jsou rozděleny na jednotlivá políčka, která vždy odpovídají jednomu alfanumerickému znaku. Každý znak je tvořen mřížkou o rozměrech 5x8 pixelů. Displeje disponují pamětí, v které je uložena předdefinovaná sada znaků, kterou může programátor využít. Zároveň má také možnost vydefinovat omezené množství vlastních znaků. (Specification of LCD module)

Displeje tohoto typu se vyrábějí ve velkém množství rozměrů. Například: 8x1, 8x2, 10x2, 16x2, 20x4 a podobně. První číslo vždy určuje počet znaků v jednom řádku a druhé číslo pak udává počet řádků. (16x2 LCD Module, 2017)

Obsluha LCD displeje je velice jednoduchá, jelikož je možné využít knihovnu LiquidCrystal, která implementuje veškeré základní funkce displeje.

Výhodou využití Alfanumerického displeje je velmi nízká cena a jednoduché ovládání pomocí knihovny LiquidCrystal. Cena LCD displeje o rozměrech 16x2 nebo 20x4 se pohybuje mezi 100–300 Kč (údaj platný v březnu 2020, e-shop: www.gme.cz, https://www.laskarduino.cz nebo https://arduino-shop.cz).

Naopak nevýhodou může být omezená znaková sada v případě lokalizace výpisu na displej do více jazyků se specifickými znakovými sadami. Další nevýhodou je nemožnost úpravy velikosti písma.

Příklad alfanumerického displeje o rozměrech 16x2 je vidět na obrázku níže - Obrázek 7.

Obrázek 7: Alfanumerický displej 16x2

Zdroj: Santy.cz, 2020

3.3.1.2 Grafické monochromatické LCD displeje

Pokročilejší možností zobrazení informací je využití Grafického monochromatického LCD displeje. Jako příklad takového displeje uvádím Grafický LCD displej 128x64 s řadičem ST7920. Existuje ovšem velké množství variant displejů, odlišných dle rozlišení

či barevného podsvícení. Výhodou využití takového displeje je větší variabilita výstupu. Displeje umožňují kromě zobrazení textu také například vykreslení jednoduchých obrázků či grafů. (Grafický LCD display 128x64 ST7920, 2017)

Pro zobrazení textu je možné využít knihovnu U8g2. Složitost vypsání textu na displej s použitím této knihovny je pak srovnatelné jako u výpisu na alfanumerický displej.

Výhodou je možnost využití více různých fontů a velikosti písmen. Je možné ovlivnit množství zobrazeného textu změnou fontu a velikosti písma. Knihovna zároveň podporuje znakovou sadu UTF8 a poskytuje tedy podporu pro lokalizaci do více jazyků. (U8g2, 2019)

Nevýhodou oproti Alfanumerickým displejům může být cena. Cena grafických displejů s rozlišením 128x64 nebo se pohybuje mezi 250–500 Kč (údaj platný v březnu 2020, e-shop https://www.laskarduino.cz).

Příklad grafického LCD displeje můžete vidět na obrázku níže - Obrázek 8.

Obrázek 8: Monochromatický grafický displej 128x64

Zdroj: (Grafický LCD display 128x64 ST7920, 2017)

3.3.1.3 Dotykové LCD displeje

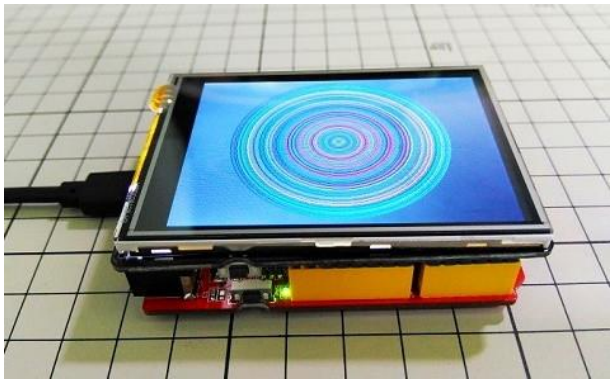
Dotykové LCD displeje představují kombinaci vstupní a výstupní periferie. Příkladem takového displeje vhodného pro použití s Arduino je 2.8" TFT Touch shield. Na rozdíl od monochromatických displejů, umožní TFT dotykové displeje zobrazení barevného obsahu včetně možnosti zpětné interakce pomocí dotykové plochy.

Práce s dotykovým displejem je již poněkud více náročná, a to hned z několika důvodů. Za prvé musíme pracovat s minimálně dvěma knihovnami. Jednu knihovnu budeme potřebovat pro práci s dotykovou vrstvou a druhou pro výpis dat na obrazovku. Pro zmíněný displej můžeme využít například knihovny *TouchScreenDriver* a *TFT Touch Shield*. Druhým důvodem je nutnost kalibrace dotykové plochy displeje pro správné získávání souřadnic. Pro různé displeje je však nutné nalézt kompatibilní knihovnu. Při použití správné knihovny je pak výpis textových či grafických dat srovnatelně náročný s předchozími příklady. Samozřejmě doplnění o možnost barevného výpisu. (Voda, 2017)

Výhodou dotykových LCD displejů je tedy jednoznačně možnost využití jako vstupní periferie. Druhou výhodou je možnost využití barev. Obě výhodu obecně zvyšují uživatelský komfort.

Nevýhodou oproti ostatním typům může být cena, která se pohybuje od 500 Kč výše (údaj platný v březnu 2020, e-shop: https://www.hwkitchen.cz/) a v případě využití pouze k výpisu textových informací může být využití tohoto typu displeje zbytečně nákladné.

Obrázek 9: LCD TFT Shield



Zdroj: 2.8" TFT TOUCH SHIELD V2.0, 2020

3.3.2 Způsoby propojení

Většinu displejů je možné zapojit několika možnými způsoby. V rámci této kapitoly představím dva základní způsoby:

- Paralelní zapojení všech pinů na jednotlivé piny Arduino
- Sériové zapojení pomocí I2C sběrnice

Rozdíly zapojení popíšu na jednoduchém příkladu zapojení LCD displeje 16x2.

3.3.2.1 Paralelní zapojení pinů

Paralelní zapojení displeje je méně úspornou variantou, co se týká využití pinů na desce. Pro připojení alfanumerického displeje je nutné zapojit celkem 10 vodičů (z toho 6 datových).

Výhodou tohoto zapojení je to, že displej je možné zapojit přímo bez prostředníka ve formě I2C převodníku.

Nevýhodou je velká spotřeba pinů desky Arduino, které může být potřebné k jinému využití dle povahy projektu. (LCD displej, 2016)

3.3.2.2 Zapojení pomocí I2C sběrnice

Sériové zapojení displeje pomocí I2C sběrnice na rozdíl od předchozí varianty využívá pro připojení s deskou Arduino pouze 4 vodiče (z toho 2 datové). Pro zapojení k Arduinu je však zapotřebí mezi displej a Arduino umístit I2C převodník (například PCF8574).

Výhodu tohoto řešení jsem zmínil hned na začátku, jde o úsporu vstupně/výstupních pinů Arduina, které je následně možné využít pro jiné účely. Druhou výhodou je možnost připojení více displejů pomoc I2C sběrnice rozlišených adresou.

Nevýhodou je nutnost zapojení převodníku pro každý displej, což může nepatrně zvýšit náklady. (LCD displej, 2016)

3.3.3 Dostupné možnosti pro implementace menu aplikace

Vzhledem k požadavku na ovládání řídící jednotky pomocí vstupních prvků a displeje zaměřil jsem se také na průzkum možností implementace menu v prostředí Arduino. Svůj průzkum jsem zaměřil na existující knihovny, pomocí kterých by bylo

možné menu implementovat. V rámci průzkumu jsem nalezl hned několik knihoven, které umožnují implementaci menu a následné zobrazení na LCD displeji.

3.3.3.1 MD_menu

MD_menu je knihovna sloužící ke tvorbě menu jako Front-endové části aplikace sloužící pro nastavení parametrů. Knihovna je uzpůsobená pro práci s alfanumerickými LCD displeji s jedním nebo dvěma řádky. Základní funkce knihovny:

- Statická definice menu
- Callback
- Timeout v případě neaktivity
- Podpora vstupních parametrů typů
    - Ano/Ne
    - Výběr ze seznamu
    - Celé číslo
    - Desetinné číslo

Uváděnou výhodou menu je jeho optimalizace využití paměti. Nevýhodou je nutnost práce pouze se statickými hodnotami. (Colli, 2017)

3.3.3.2 ArduinoMenu 4

ArduinoMenu 4 je rozsáhlá knihovna umožňující vytvoření libovolné struktury menu pomocí předem definovaných objektů a maker. Menu obsahuje velké množství předpřipravených tříd pro podporu:

- Tvorby menu a submenu v libovolném množství úrovní
- Tvorby prvků menu pro nastavení proměnné:
    - Z předem definovaného seznamu
    - Staticky nebo dynamicky nastaveného rozsahu
- Tvorbu prvků menu pro Callback funkcí

Knihovna také podporuje výpis na širokou škálu zařízení, např.:

- Alfanumerické displeje
- Grafické displeje
- Sériovou linku

Knihovna nabízí podporu pro různé typy ovládání včetně vstupu pomocí tlačítek, streamu ze sériové linky nebo klávesnic.

Výhodou knihovny je velká univerzálnost a široká škála využití, možnost nastavení proměnné v předem nastaveném rozsahu s možností dynamické deklarace rozsahu.

Nevýhodou naopak je poněkud složitější definice menu v případě využití dynamického rozsahu. V případě tvorby statického menu je možné využít předem definovaná makra, které zjednodušují definici. Poslední nevýhodu je vyšší náročnost na paměť než například u MD_menu. (Azevedo, 2019)

3.3.3.3 LiquidMenu

Knihovna umožňující definici a používání menu zobrazené na Alfanumerickém displeji. Knihovna umožňuje vytvářet menu pomocí hierarchicky uspořádaných tříd. Rozděluje menu do následujících struktur:

- LiquidLine – třída reprezentující řádek textu nebo čísel na displeji, umožňuje výpis a editaci proměnné
- LiquidScreen – třída reprezentující kolekci řádků zobrazených na jedné obrazovce zároveň
- LiquidMenu – třída kombinující obrazovky do jednoho menu, umožnuje ovládání menu pomocí metod třídy

Základní podporované vlastnosti menu:

- Výběr hodnoty z nabídky
- Callback
- Podpora připojeného displeje pomocí I2C

Výhodou knihovny je jednoduchá a strukturovaná definice menu. Nevýhodou je omezení výstupu menu pouze pro Alfanumerické displeje a omezení možnosti výběru hodnot pouze z předem nastavených hodnot. (Kalchev, 2019)

3.4 Vývojové desky Arduino

V současné době existuje na trhu velké množství různých vývojových desek Arduino. Z toho důvodu jsem se rozhodl udělat malý přehled alespoň několika z nich, abych se na základě tohoto přehledu mohl následně rozhodnout, která z nich bude nejvhodnější pro cílové řešení.

3.4.1.1 Arduino Mini

Arduino Mini (Obrázek 10) je deska o rozměrech 18x30 mm navržená pro projekty vyžadující úsporu místa. A je nejmenší oficiální verzí Arduino. Základem desky je procesor ATmega328 s taktem 16 MHz. Procesor pracuje s operační pamětí o velikosti 2 kB.

Navzdory svým malým rozměrům tak disponuje srovnatelným výkonem s většími deskami. Naopak vzhledem k svým malým rozměrům neobsahuje USB port a pro programování desky je tak nutné využít externí USB-Seriál převodník. (Voda, 2017)

Navzdory své malé velikosti disponuje Arduino Mini následujícím rozhraním:

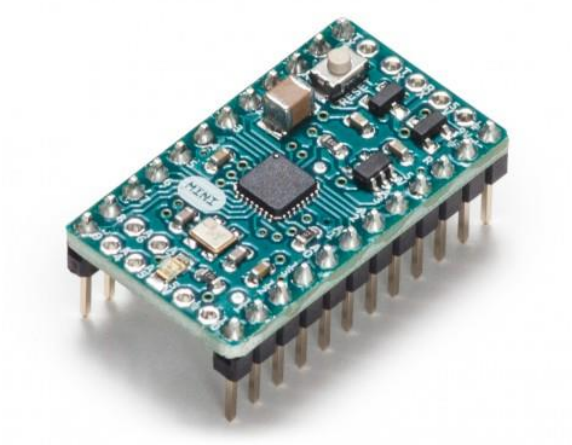
- 14 digitálních vstupně výstupních pinů (0-13)
- 8 Analogových vstupních pinů (A0-A7)
- Dvě dvojice pinů RX a TX pro nahrání programu či komunikaci pomocí sériové linky

(Getting Started with the Arduino Mini, 2018)

Arduino Mini dále umožnuje využití EEPROM paměti o velikosti 1 kB.

Cena oficiální distribuce Arduino Mini se pohybu kolem 370–430 Kč (údaje z března 2020 na e-shopech: www.hwkitchen.cz, www.alza.cz). Na trhu se však pohybují klony z neoficiální distribuce se stejnými vlastnostmi, které je možné pořídit řádově za nižší ceny okolo 100–160 Kč (údaje z března 2020 na e-shopech: www.gme.cz, www.arduino-shop.cz)

Obrázek 10: Arduino Mini



Zdroj: (Arduino Store)

3.4.1.2 Arduino Nano

Arduino Nano (Obrázek 11) je deska o rozměrech 18x45 mm. Od desky Arduino Mini, popsané v předchozí kapitole se liší především přítomností USB portu a USB-Seriál převodníku, díky čemuž je deska celkově o trochu větší.

Deska tedy obsahuje tedy shodně procesor ATmega328 s taktem 16 MHz a operační paměť o velikosti 2 kB. (Voda, 2017)

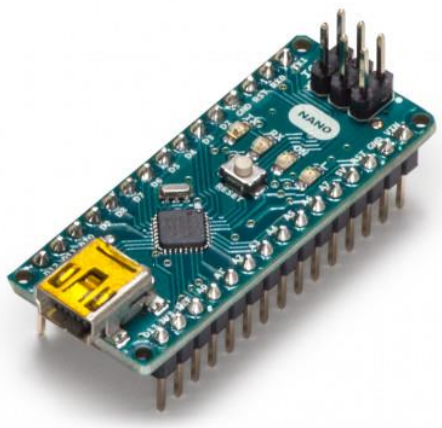
Rozhraní desky je opět velmi podobné:

- 14 digitálních vstupně výstupních pinů (0-13)
- 8 Analogových vstupních pinů (A0-A7)
- Mini USB pro nahrání programu nebo komunikaci po sériové lince
- Dvojice RX a TX pinů pro komunikaci prostřednictvím sériové linky

Arduino Nano také umožnuje využití EEPROM paměti o velikosti 1 kB. (Arduino Store)

Cena oficiální distribuce: 610-670 Kč (údaje z března 2020 na e-shopech: www.hwkitchen.cz, www.alza.cz)
Cena dostupných klonů: 140-160 Kč (údaje z března 2020 na e-shopech: www.gme.cz, www.arduino-shop.cz)

Obrázek 11: Arduino Nano



Zdroj: (Arduino Store)

3.4.1.3 Arduino Micro

Arduino Micro (Obrázek 12) je deska o 18x48 mm. Na rozdíl od desek Arduino Mini a Nano je základem desky Arduino Micro procesor ATmega32U4, která desce po připojení k PC umožňuje simulovat chování počítačové myši či klávesnice. To desce umožňuje posílat počítači příkazy typu stisk tlačítka nebo posun myši. Deska je tedy možné využít například k sestavení vlastí klávesnice nebo herního ovladače.

Procesor ATmega32U4 pracuje s taktem 16 Hz a pracuje s operační pamětí o velikosti 2.5 kB. (Voda, 2017)

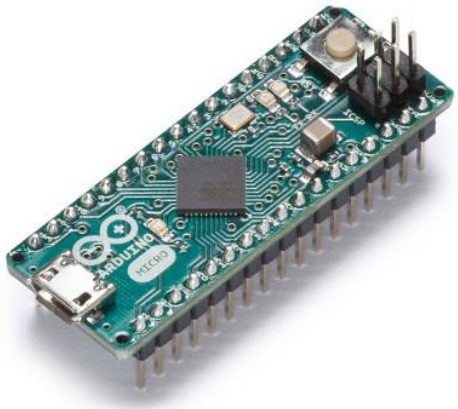
Rozhraní desky:

- 14 digitálních vstupně výstupních pinů (0-13)
- 6 Analogových vstupních pinů (A0-A7)
- 4 piny SPI rozhraní
- Mini USB pro nahrání programu nebo komunikaci po sériové lince
- Dvojice RX a TX pinů pro komunikaci prostřednictvím sériové linky

Stejně jako ostatní i Arduino Micro umožnuje využití EEPROM paměti o velikosti 1 kB. (Arduino Store)

Cena oficiální distribuce: 550-570 Kč (údaje z března 2020 na e-shopech: www.hwkitchen.cz, www.alza.cz)
Cena dostupných klonů: 300 Kč (údaje z března 2020 na e-shopech: www.gme.cz, www.arduino-shop.cz)

Obrázek 12: Arduino Micro



Zdroj: (Arduino Store)

3.4.1.4 Arduino Uno

Arduino UNO (Obrázek 13) je v současné době jedna z nejpoužívanějších desek. Její rozměry jsou 68.6x53.4 mm. Jádrem desky je procesor ATmega328 s frekvencí 16 MHz a operační pamětí 2 kB. (Voda, 2017)

Rozhraní desky:

- 14 digitálních vstupně výstupních pinů (0-13)

- 6 Analogových vstupních pinů (A0-A5)
- USB pro nahrání programu nebo komunikaci po sériové lince
- Souosý napájecí konektor

Arduino UNO umožnuje využití EEPROM paměti o velikosti 1 kB. (Arduino Store)

Cena oficiální distribuce: 610-660 Kč (údaje z března 2020 na e-shopech: www.hwkitchen.cz, www.alza.cz)
Cena dostupných klonů: 150-198 Kč (údaje z března 2020 na e-shopech: www.gme.cz, www.arduino-shop.cz)

Obrázek 13: Arduino UNO

Zdroj: (Arduino Store)

3.4.1.5 Arduino Leonardo

Deska Arduino Leonardo (Obrázek 14) je deska, která navazuje na desku UNO. Má shodné rozměry 68.6x53.4 mm. Jádrem desky je však, stejně jako u Arduino Micro, procesor ATmega32u4 umožňující simulovat chování klávesnice nebo myši. Rychlost procesoru je 16 MHz a pracuje s operační pamětí o velikosti 2.5 kB. (Voda, 2017)

Rozhraní desky:

- 14 digitálních vstupně výstupních pinů (0-13)
- 6 Analogových vstupních pinů (A0-A5)
- Micro USB pro nahrání programu nebo komunikaci po sériové lince
- Souosý napájecí konektor

Arduino UNO umožnuje využití EEPROM paměti o velikosti 1 kB. (Arduino Store)

Cena oficiální distribuce: 550-570 Kč (údaje z března 2020 na e-shopech: www.hwkitchen.cz, www.alza.cz)
Cena dostupných klonů: 270 Kč (údaje z března 2020 na e-shopech: www.arduino-shop.cz)

Obrázek 14: Arduino Leonardo

Zdroj: (Arduino Store)

3.4.1.6 Arduino Mega2560

Arduino Mega2560 (Obrázek 15) je jedna z desek, která vznikla prodloužením desky UNO. Její rozměry jsou 101.52x53.3 mm. Prodloužený design umožnuje osazení výkonnějšími čipy a také vzniká prostor pro více vstupních a výstupních pinů. Jádrem desky je procesor ATmega2560 s frekvencí 16 MHz pracující s operační pamětí 8 kB. (Voda, 2017)

Rozhraní desky:

- 54 digitálních vstupně výstupních pinů (0-53)
- 16 Analogových vstupních pinů (A0-A15)
- USB pro nahrání programu nebo komunikaci po sériové lince
- Souosý napájecí konektor

Arduino UNO umožnuje využití EEPROM paměti o velikosti 4 kB. (Arduino Store)

Cena oficiální distribuce: 1050 Kč (údaje z března 2020 na e-shopech: www.alza.cz)
Cena dostupných klonů: 460-480 Kč (údaje z března 2020 na e-shopech: www.arduino-shop.cz, www.gme.cz)

Obrázek 15: Arduino Mega2560

Zdroj: (Arduino Store)

4 Vlastní řešení

Kapitola Vlastní řešení je jádrem celé práce. V kapitole a relevantních podkapitolách popíšu jednotlivé kroky řešení. Celá kapitola je rozdělena na čtyři dílčí celky.

Nejprve bylo nutné provést analýzu řešení z jejichž základů celé řešení vychází. Popis analýzy řešení je obsahem kapitoly 4.1 Analýza řešení. Na základě poznatků z této kapitoly jsem vytvořil funkční řešení. Způsob implementace tohoto řešení jsem popsal v kapitole 4.2 Realizace. Implementované řešení jsem následně otestoval v rozsahu definovaném v kapitole 4.3 Testování řešení, kde také uvádím výsledky tohoto testování. Na závěr jsem vypočítal cenovou kalkulaci realizovaného řešení. Kalkulaci naleznete v kapitole 4.4 Cenová kalkulace navrhovaného řešení.

4.1 Analýza řešení

V rámci analýzy řešení nejdříve vydefinují požadavky na cílové řešení, které budou sloužit jako podklad pro další kapitoly a v dalších částech práce budou podkladem pro hodnocení naplnění cílů řešení. Definované požadavky navazují na předpokládané výhody popsané v kapitole 3.1.4. Do analýzy řešení také zařadím výběr vhodné komponenty a způsobu řízení krokového motoru, které byly popsány v kapitole 3.2 a výběr typu Arduino desky, na které bude postavené celé řešení. Při výběru desky budu odkazovat zejména na popis desek z kapitoly 3.4. Na závěr na základě těchto výběrů navrhnu vhodnou architekturu cílového řešení. Kapitola Analýza řešení bude hlavním podkladem pro následující kapitoly 4.2 Realizace řešení a 4.3 Testování řešení.

4.1.1 Analýza požadavků a návrh cílového řešení

V rámci této kapitoly vydefinuji funkční a nefunkční požadavky na řídící jednotku, popíšu jednotlivé případy užití a popíšu návrh cílového řešení.

4.1.1.1 Funkční požadavky řídící jednotky

V následující tabulce (Tabulka 5) jsou popsány všechny funkční požadavky, které by měla řídící jednotka splňovat. Požadavky jsou rozděleny do dvou kategorií:

- **Must-have** – požadavky, které jsou klíčové pro správně fungování celého řešení
- **Nice-to-have –** požadavky, které v případě nerealizace neovlivní funkčnost celého řešení, pouze relevantní části aplikace

Tabulka 5: Funkční požadavky

Číslo požadavku	Název požadavku	Popis požadavku	Kategorie
F1	Ovládání aplikace (Menu)	Řídící jednotku čerpadla bude možné obsluhovat pomocí připojených tlačítek a displeje. Na displeji bude po spuštění aplikace zobrazeno menu aplikace, kterým bude možné procházet pomocí ovládacích tlačítek.	Must-have

		Pomocí menu budou dostupné všechny funkcionality.	
F2	Kontinuální čerpání	Řídící jednotka umožní spuštění kontinuálního čerpání s možností nastavení průtoku čerpadla	Must-have
F3	Přesné čerpání	Řídící jednotka umožní spuštění přesného čerpání s možností nastavení průtoku čerpadla a požadovaného objemu.	Must-have
F4	Kalibrace	Bude dostupná funkce kalibrace čerpadla, pomocí, které bude možné určit množství načerpané kapaliny na jeden krok motoru. Výsledek kalibrace bude uložen do paměti zařízení a bude dostupný i po odpojení napájení.	Must-have
F5	Nastavení	Bude možné nastavit výchozí hodnoty pro: Režim čerpání Průtok Objem	Nice-to-have
F6	Ovládání pomocí připojeného PC	Zařízení bude možné ovládat přes příkazovou řádku, připojením zařízení k počítači. Bude vydefinována sada příkazů, pomocí kterých bude možné provést všechna nastavení, kalibraci a spustit čerpání.	Nice-to-have
F7	Okamžité zastavení čerpání	K řídící jednotce bude připojeno tlačítko, které umožní okamžité zastavení spuštěného čerpání.	Must-have

Zdroj: vlastní zpracování

4.1.1.2 Nefunkční požadavky řešení

V následují tabulce uvádím veškeré nefunkční požadavky, které musí navrhované řešení řídící jednotky splňovat:

Tabulka 6: Nefunkční požadavky

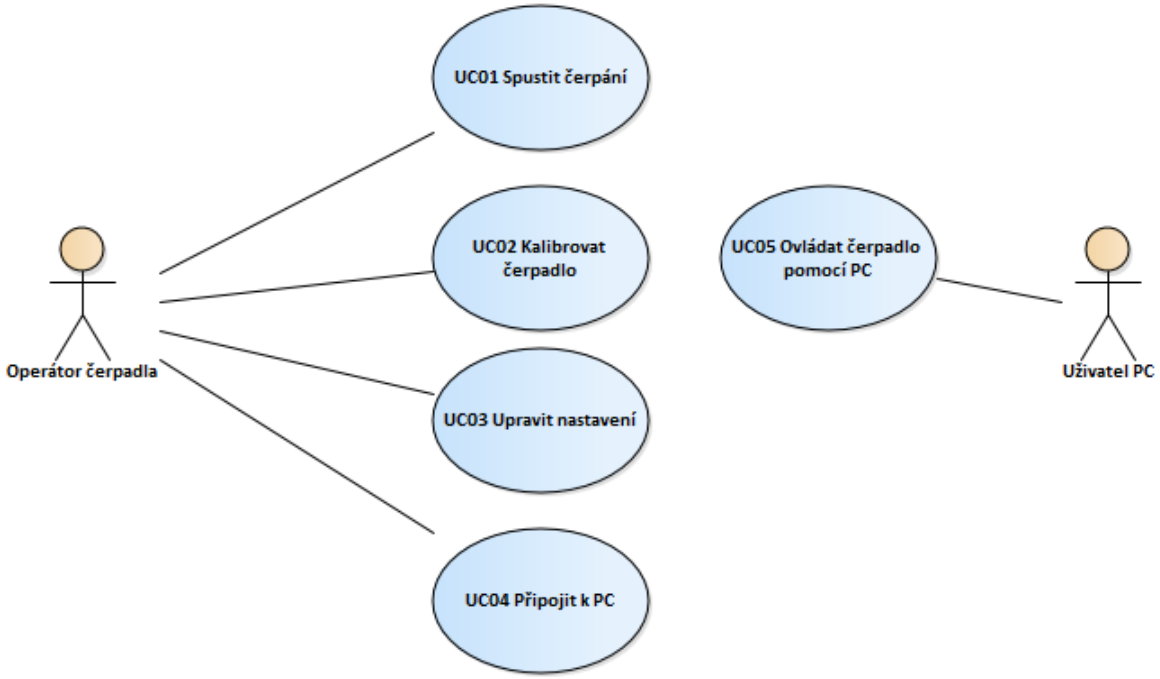
Číslo požadavku	Název požadavku	Popis
N1	Použitá platforma	Vývojová deska Arduino
N2	Model čerpadla	Řídící jednotka bude ovládat čerpadlo WP1000 od firmy Welco, jehož základem je bipolární krokový motor s označením FB (WP1000 Peristaltic pump)
N3	Displej	Řešení bude obsahovat min. dvouřádkový displej.
N4	Ovládací prvky	Jako vstupní ovládací prvek budou použita tlačítka.
N5	Prototyp – nepájivé pole	Prototyp řídící jednotky čerpadla bude sestaven pomocí nepájivého pole a propojovacích kabelů.

Zdroj: vlastní zpracování

4.1.1.3 Případy užití řešení (Use case)

Na základě analýzy funkčních požadavků cílového řešení pro řídící jednotku čerpadla jsem navrhl sadu případů užití, které bude nutné realizovat za účelem pokrytí všech požadavků. Návrh případů užití je znázorněn na obrázku - Obrázek 16. V následujících podkapitolách podrobněji popíšu scénáře pro znázorněné případy užití. Use casy budou hlavním podkladem pro tvorbu testovacích scénářů definovaných v kapitole 4.3.1.

Obrázek 16: Diagram případů užití



Zdroj: vlastní zpracování

4.1.1.3.1 UC01 Spustit čerpání

Číslo UC	*UC01*
Název UC	Spustit čerpání
Aktéři	Operátor čerpadla
Vstupní podmínky	• Čerpadlo je zapnuté • Neprobíhá žádné čerpání • Je zobrazeno menu
Hlavní scénář	1. Uživatel v menu vybere volbu START a potvrdí tlačítkem. 2. Systém načte defaultní hodnoty nastavení. 3. Systém zobrazí podmenu pro spuštění čerpání. Na prvním místě je zobrazena položka MODE s volbou „Continous“ 4. Uživatel pomocí tlačítek vybere jeden z režimů: a. „Continous“ – průběžné čerpání b. „Specific volume“ – Přesné čerpání 5. Pokud uživatel vybral hodnotu „Specific volume“ pokračuje Alt. scénářem. V opačném případě pokračuje krokem 5. 6. Uživatel vybere volbu SPEED. 7. Systém zobrazí výchozí nastavení hodnoty SPEED (průtok). 8. Uživatel pomocí tlačítek nastaví požadovanou hodnotu průtoku. 9. Uživatel zobrazí položku RUN a potvrdí tlačítkem. 10. Systém dle nastavené hodnoty průtoku vypočítá potřebnou hodnotu otáček motoru. 11. Systém spustí čerpání pomocí spuštění chodu motoru.

Alt. Scénář – Přesné čerpání	1. Uživatel vybere volbu VOLUME. 2. Systém zobrazí výchozí nastavení hodnoty VOLUME (objem). 3. Uživatel pomocí tlačítek nastaví požadovanou hodnotu objemu. 4. Uživatel zobrazí položku RUN a potvrdí tlačítkem. 5. Systém dle nastavené hodnoty průtoku vypočítá potřebnou hodnotu otáček motoru. 6. Systém spustí čerpání pomocí spuštění chodu motoru. 7. Systém průběžně kontroluje načerpané množství kapaliny a po dosažení požadované hodnoty zastaví otáčení motoru.
Výstupní podmínky	Čerpadlo běží, dle nastavených podmínek nebo bylo zastaveno po dosažení požadované hodnoty u přesného čerpání.
Poznámky	

4.1.1.3.2 UC02 Kalibrovat čerpadlo

Číslo UC	*UC02*
Název UC	Kalibrovat čerpadlo
Aktéři	Operátor čerpadla
Vstupní podmínky	• Čerpadlo je zapnuté • Neprobíhá žádné čerpání • Je zobrazeno menu
Hlavní scénář	1. Uživatel v menu vybere volbu CALIBRATION a potvrdí tlačítkem 2. Systém zobrazí submenu kalibrace, jako první je zobrazena položka VOLUME 3. Uživatel nastaví hodnotu VOLUME, podle objemu kalibrační nádoby. 4. Uživatel vybere volbu CALIBERATE a potvrdí tlačítkem 5. Systém spustí čerpání čerpadla s nízkými otáčkami a zobrazí na displeji volbu STOP, zároveň spustí časovač. 6. Uživatel čeká, než dojde k načerpání zvoleného množství a následně potvrdí pomocí STOP. 7. Systém vypočítá hodnotu průtoku (v ml/min) a uloží do paměti společně s počtem provedených kroků motoru. 8. Systém zobrazí volbu CONTINUE. 9. Uživatel vyprázdní nádobu a stiskne CONTINUE. 10. Systém spustí čerpání čerpadla s vysokými otáčkami a zobrazí na displeji volbu STOP, zároveň spustí časovač. 11. Uživatel čeká, než dojde k načerpání zvoleného množství a následně potvrdí pomocí STOP. 12. Systém vypočítá hodnotu průtoku (v ml/min) a uloží do paměti společně s počtem provedených kroků motoru.

	13. Systém ověří správnost údajů kalibrace – zda hodnota průtoků při nízkých otáčkách není vetší nebo rovna hodnotě průtoku při vysokých otáčkách. a. Pokud ano vypíše chybovou hlášku 14. Systém ze zjištěného počtu kroků vypočítá počet kroků potřebných k načerpání jedné desetiny ml a uloží jí do paměti. 15. Kalibrace je dokončena. 16. Systém zobrazí úvodní menu.
Výstupní podmínky	Nové nastavení kalibrace bylo uloženo do EEPROM paměti. Do paměti EEPROM byly uloženy: • Minimální hodnota průtoků o Dosahována při minimální rychlosti motoru • Maximální hodnota průtoku o Dosahována při maximální rychlosti motoru • Počet kroků motoru pro k načerpání desetiny ml

4.1.1.3.3 UC03 Upravit nastavení

Číslo UC	*UC03*
Název UC	Upravit nastavení
Aktéři	Operátor čerpadla
Vstupní podmínky	• Čerpadlo je zapnuté • Neprobíhá žádné čerpání • Je zobrazeno menu
Hlavní scénář	1. Uživatel v menu vybere volbu SETTINGS a potvrdí tlačítkem 2. Systém zobrazí submenu nastavení, jako první je zobrazena položka DEF.MODE 3. Uživatel pomocí tlačítek nastaví jeden z režimů: a. „Continous“ – průběžné čerpání b. „Specific volume“ – Přesné čerpání 4. Uživatel zobrazí volbu DEF.SPEED a nastaví jeho hodnotu. 5. Uživatel zobrazí volbu DEF.VOLUME a nastaví jeho hodnotu. 6. Uživatel zobrazí volbu SAVE a potvrdí. 7. Systém uloží hodnoty do paměti EEPROM. 8. Systém zobrazí potvrzení o uložení.
Výstupní podmínky	Nové nastavení defaultních hodnot bylo uloženo do EEPROM paměti.

4.1.1.3.4 UC04 Připojit k PC

Číslo UC	*UC04*
Název UC	Připojit k PC
Aktéři	Operátor čerpadla
Vstupní	• Čerpadlo je zapnuté

podmínky	• Neprobíhá žádné čerpání • Je zobrazeno menu • Zařízení je připojení k PC pomocí USB
Hlavní scénář	1. Uživatel v menu vybere volbu PC CONTROL a potvrdí tlačítkem 2. Systém zobrazí submenu s jedinou položkou CONNECT. 3. Uživatel potvrdí volbu CONNECT. 4. Systém vytvoří rozhraní pro připojení pomocí PC. 5. Zařízení je nyní možné ovládat pomocí PC sériové linky (viz UC05 Ovládat čerpadlo pomocí PC). 6. V případě probíhajícího čerpání není možné odpojit. 7. Uživatel zadá na příkazové řádce příkaz „exit“ 8. Systém ukončí propojení pomocí sériové linky.
Výstupní podmínky	
Poznámky	

4.1.1.3.5 UC05 Ovládat čerpadlo pomocí PC

Číslo UC	*UC05*
Název UC	Ovládat čerpadlo pomocí PC
Aktéři	Uživatel PC
Vstupní podmínky	• Čerpadlo je zapnuté • Neprobíhá žádné čerpání • Je vytvořeno propojení na PC
Hlavní scénář	1. Uživatel se pomocí PC připojí k sériové lince pomocí parametrů nastavených na čerpadle. 2. Systém čeká na zadání příkazu 3. Uživatel zadá libovolný příkaz (tabulka validních příkazů viz Tabulka 7: Seznam příkazů) 4. Systém provede validaci příkazu, pokud je příkaz validní pokračuje dalším krokem. V opačném případě vypíše chybovou hlášku (viz Poznámky) a UC pokračuje krokem 2. 5. Systém zpracuje příkaz dle tabulky - Tabulka 7: Seznam příkazů. V případě, že se jedná o příkaz Exit pokračuje se následujícím krokem v opačném případě se po zpracování příkazu pokračuje krokem 2. 6. Systém ukončí spojení s PC a na zařízení je zpět zobrazeno menu.
Poznámky	Seznam chybových hlášek: • Invalid command – nebyl rozpoznán příkaz • Invalid syntax – příkaz byl rozpoznán ale je zapsán chybně • Invalid parameters – příkaz má nevalidně zadané parametry

Tabulka 7: Seznam příkazů

Příkaz	Formát příkazu	Stručný popis scénáře příkazu	Poznámky
Kalibrace	calibrate VOL	1. Systém zobrazí název prvního kroku čerpání. 2. Uživatel potvrdí spuštění čerpání Enterem 3. Čerpadlo začne čerpat pomalou rychlostí 4. Uživatel po načerpání hodnoty stiskne Enter 5. Systém zobrazí název druhého kroku čerpání. 6. Uživatel potvrdí spuštění čerpání Enterem 7. Čerpadlo začne čerpat rychlou rychlostí 8. Uživatel po načerpání hodnoty stiskne Enter 9. Systém zobrazí hlášku o konci kalibrace 10. Systém provede validaci stejně jako v UC02 a uloží hodnoty.	VOL hodnota objemu v ml SPD průtok čerpadla v ml/min MOD c – průběžné čerpání s – přesné čerpání V případě zadání nevalidních parametrů nebo nevalidních příkazů dojde k vypsání chybové hlášky.
Průběžné čerpání	pump -c SPD	Systém spustí čerpání a postupně vypisuje načerpanou hodnotu, pokud uživatel stiskne Enter čerpání se ukončí.	
Přesné čerpání	pump -s SPD VOL	Systém spustí čerpání a postupně vypisuje načerpanou hodnotu, pokud dojde k načerpání hodnoty VOL dojde k ukončení čerpání. Pokud uživatel stiskne Enter čerpání se ukončí dříve.	
Nastavení hodnot	settings VOL SPD MOD	Systém uloží zadané parametry do paměti a vypíše potvrzující hlášku.	
Nápověda	info	Systém zobrazí seznam příkazů	

Ukončení	exit	Systém ukončí připojení s čerpadlem

Zdroj: vlastní zpracování

4.1.1.3.6 UC06 Přerušit čerpání

Číslo UC	*UC06*
Název UC	Přerušit čerpání
Aktéři	Operátor čerpadla
Vstupní podmínky	• Čerpadlo je zapnuté • Probíhá libovolné čerpání
Hlavní scénář	1. Systém pomocí ovládání krokového motoru provádí čerpání. 2. Uživatel stiskne tlačítko pro předčasné přerušení čerpání. 3. Systém zastaví krokový motor a čerpání je ukončeno. 4. Systém zobrazí úvodní menu.
Výstupní podmínky	Je zobrazeno menu, čerpání neprobíhá.
Poznámky	

Zdroj: vlastní zpracování

4.1.2 Výběr desky pro vývoj prototypu řídící jednotky

Jedním ze základních nefunkčních požadavků je implementace řadiče pomocí vývojové desky Arduino. Do výběru vhodné desky jsem tedy zařadil pouze desky splňující toto základní kritérium. U každé desky se zaměřím na parametry uvedené v tabulce - Tabulka 8. Pro každý parametr uvádím prioritu na stupnici 1-3, kde číslo 1 znamená nejvyšší prioritu.

Tabulka 8: Seznam parametrů pro výběr desky

Parametr	Popis	Priorita
Počet digitálních vstupně výstupních pinů	Vzhledem k předpokládanému většímu množství periferií je počet digitálních pinů velice důležitý.	1
Integrovaná EEPROM paměť	Pro uložení nastavení i po vypnutí řadiče je potřeba využít EEPROM paměť. Tu by bylo možné v případě potřeby nahradit externím modulem, což by však zvýšilo cenu a celkovou náročnost řešení.	1
Cena	Jedním z cílů projektu, je minimalizovat náklady na cílové řešení. Cena desky, bude jedním ze základních pilířů cenové kalkulace.	1
Konektivita s PC	Jedním z funkčních požadavků je možnost ovládání řadiče pomocí PC za pomocí USB.	2

USB-Seriál převodník	Parametr má nižší prioritu, jelikož je možné jej zajistit externím převodníkem.	2
Velikost EEPROM paměti	Velikost paměti je méně prioritní parametr, jelikož nepředpokládám ukládání velkého množství dat. Předpokládám, že velikost paměti nebude limitující.	3
Velikost desky	Rozměry desky jsou spíše pomocným parametrem pro rozhodování. Požadavky na velikost nejsou přesně specifikovány.	3

Zdroj: vlastní zpracování

Pro výběr vhodně desky jsem vybral sadu desek blíže popsaných v kapitole 3.4. Pro jednoduší porovnání jsem sestavil tabulku (Tabulka 9) kde uvádím klíčové rozhodovací parametry jednotlivých desek.

Tabulka 9: Přehled desek a výběrových parametrů

Typ Arduino desky	**Priorita**	**Mini**	**Nano**	**Micro**	**UNO**	**Leonardo**	**Mega2560**
Počet digitálních I/O pinů	1	14	14	14	14	14	54
EEPROM paměť	1	✓	✓	✓	✓	✓	✓
Cena v Kč[2]	1	370-430 (100-160)	610-670 (140-160)	550-570 (300)	610-660 (150-198)	550–570 (270)	1045 (460–480)
Konektivita s PC	2	Serial (RX, TX)	Mini USB	Micro USB	USB	Micro USB	USB
USB-Seriál převodník	2	✗	✓	✓	✓	✓	✓
Velikost EEPROM paměti	3	1 kB	1 kB	1 kB	1 kB	1 kB	4 kB

[2] V závorce je uvedena cena na trhu dostupného kompatibilního klonu.

Velikost desky v mm	3	18x45	18x48	18x48	68.6x53.4	68.6x53.4	101.52x 53.3
Další poznámky	-				Souosý napájecí konektor	Souosý napájecí konektor	Souosý napájecí konektor

Zdroj: vlastní zpracování dat z kapitoly 3.4

Na základě dat v Tabulka 9 jsem usoudil, že nejvhodnější desky pro implementaci řadiče jsou desky Arduino Nano a Arduino UNO. Vzhledem ke srovnatelným parametrům obou desek jsem se rozhodl pro výběr desky Arduino UNO, která navíc obsahuje souosý napájecí konektor.

4.1.3 Výběr komponenty pro ovládání krokového motoru

V kapitole 3.2 jsem popisoval možnosti řízení krokových a možnosti implementace. Z nabízených variant jsem se rozhodl využít pro cílové řešení H-můstek a implementaci řízení pomocí knihovny Stepper. Hlavními důvody pro tento výběr byly jednoduchost ovládání pomoc knihovny Stepper a možnost jednoduchého nastavení rychlosti.

4.1.4 Návrh architektury řešení

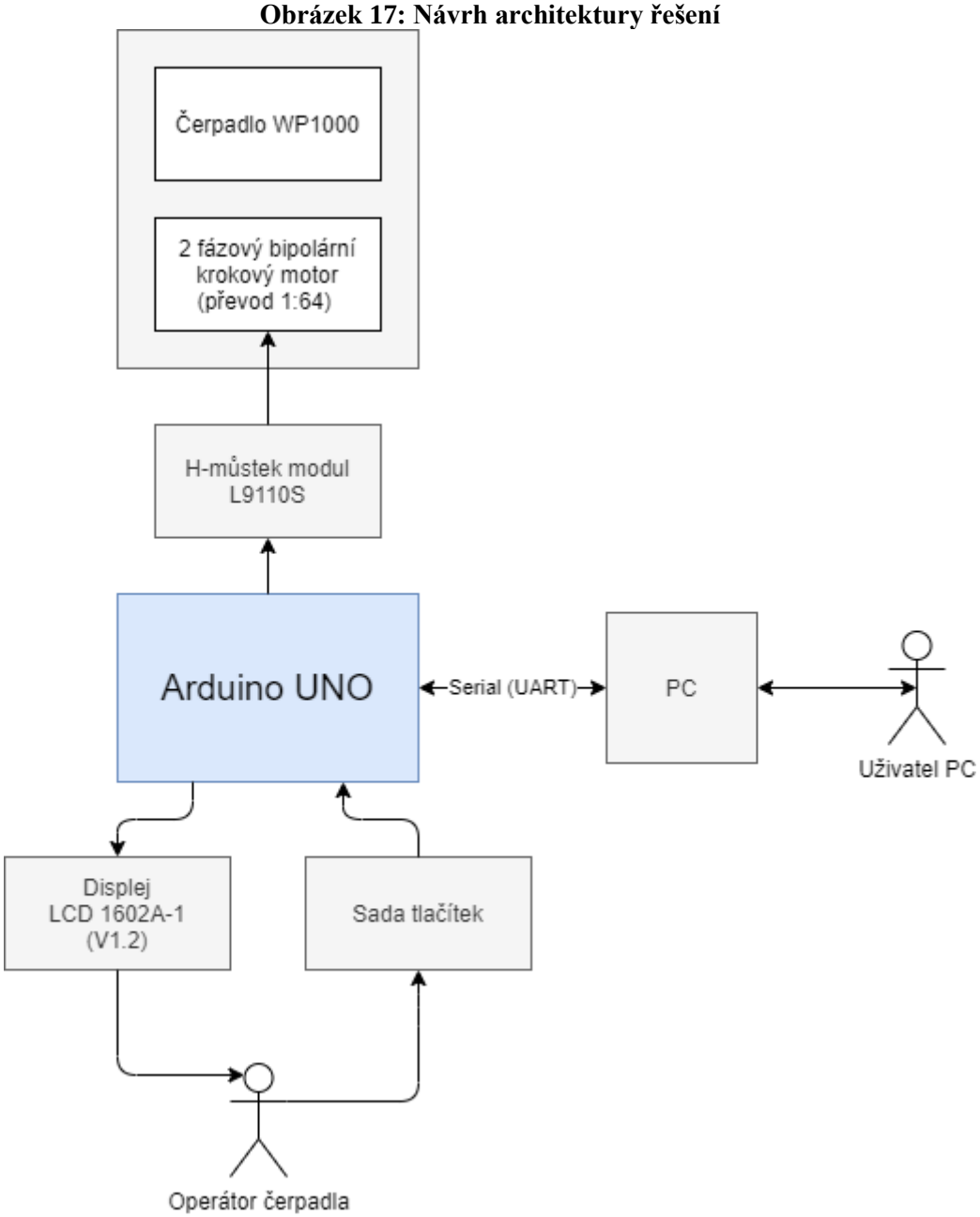
Pro cílové řešení jsem se rozhodl využít architekturu znázorněnou na obrázku - Obrázek 17.

Základem celého řešení je deska Arduino UNO, která zajišťuje logiku řešení a komunikaci mezi ostatními součástmi řešení. Komunikaci s operátorem zajišťuje dvouřádkový displej typu LCD 1602A-1 (Specification of LCD module, 2008), připojen pomocí šesti datových pinů přímo k desce a sada ovládacích tlačítek. Komunikaci mezi řídící jednotkou a krokovým motorem čerpadla je zajištěna pomocí modulu L9110S (H-můstek modul L9110S, 2016), který je složen z dvojice h-můstků.

V řešení jsem se rozhodl využít jednoduchý alfanumerický displej, jehož vlastnosti jsou pro navrhované řešení dostatečné a umožní snížení nákladů na cílové řešení.

Podrobné schéma zapojení jednotlivých součástí je součástí kapitoly 4.2 Realizace řešení.

Obrázek 17: Návrh architektury řešení



Zdroj: vlastní zpracování

4.2 Realizace řešení

V této kapitole popíšu celkové řešení. Kapitola má dvě základní podkapitoly. První podkapitola 4.2.1 Popis komponent řídící jednotky se zabývá HW částí řešení. SW část řešení je následně popsána v kapitole 4.2.2 Popis SW řešení řídící jednotky.

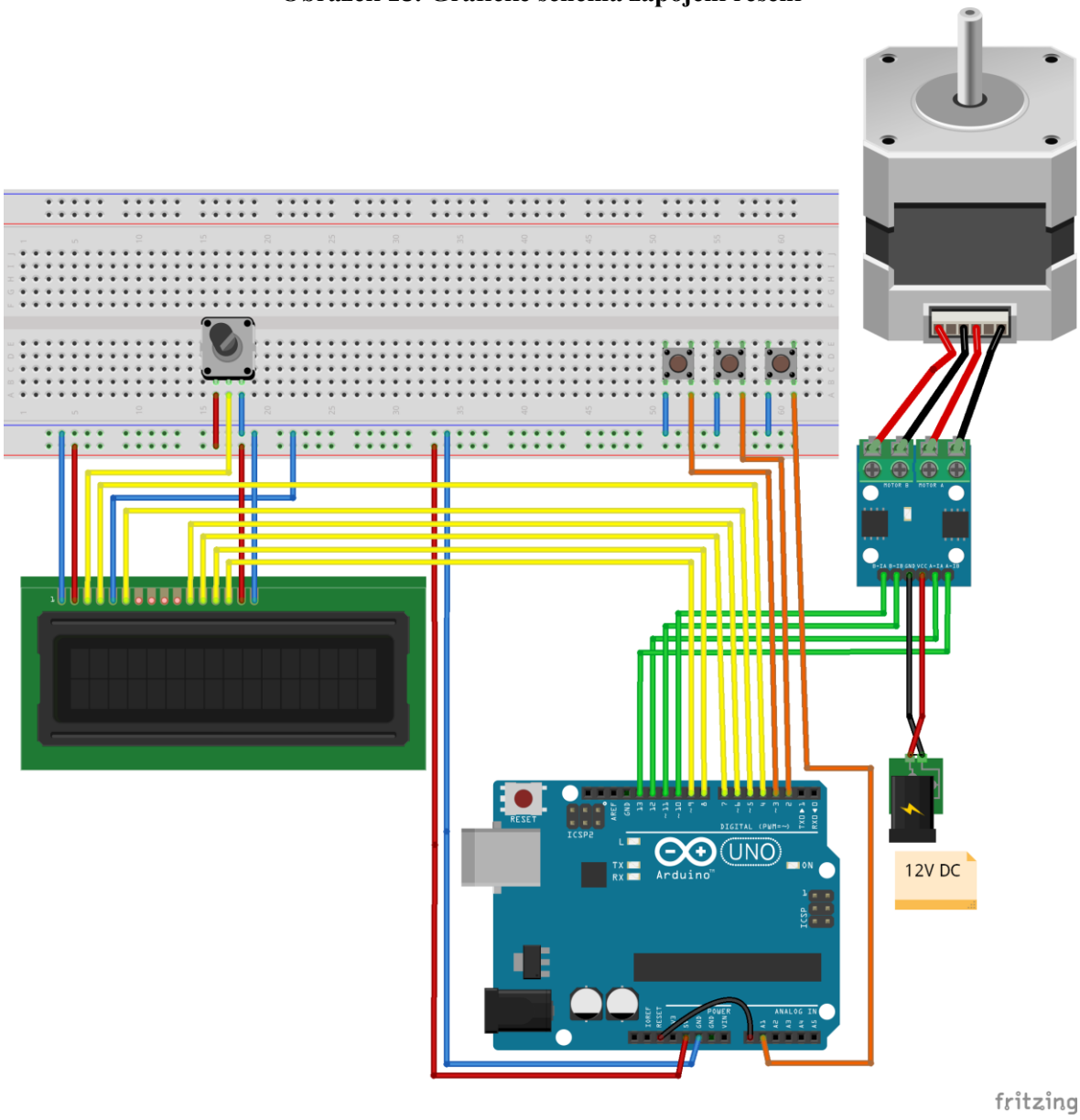
4.2.1 Popis komponent řídící jednotky

Tato kapitola se zabývá popisem jednotlivých komponent a jejich zapojení. Nejdříve představím celkové schéma zapojení a následně popíšu jednotlivé komponenty celkového řešení a jejich účel.

4.2.1.1 Schéma zapojení

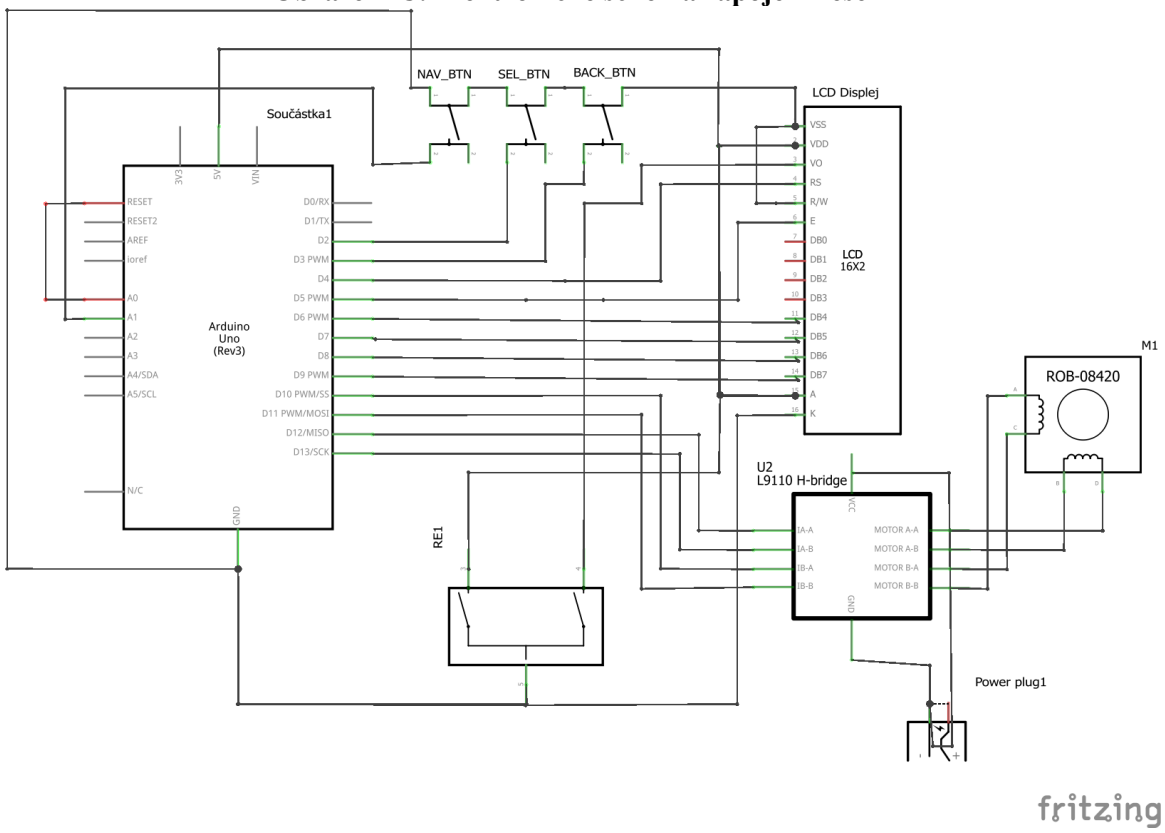
Schéma zapojení řešení s použitím nepájivého pole uvádím na obrázku - Obrázek 18. Elektronické schéma zapojení je následně uveden na obrázku - Obrázek 19. Popis jednotlivých komponent uvádím v jednotlivých podkapitolách dále.

Obrázek 18: Grafické schéma zapojení řešení



Zdroj: vlastní zpracování pomocí nástroe Fritzing

Obrázek 19: Elektronické schéma zapojení řešení



Zdroj: vlastní zpracování pomocí nástroje Fritzing

4.2.1.2 Arduino UNO

Vývojová deska Arduino UNO slouží jako jádro celého řešení a společně s nahraným programem, popsaným v kapitole 4.2.2 Popis SW řešení řídící jednotky, tvoří jádro celého řešení.

Úkoly a zodpovědnost komponenty:

- Zajištění aplikační logiky
- Zajištění komunikace mezi jednotlivými komponentami
- Trvalé uložiště nastavení v paměti EEPROM

Deska je s ostatními komponentami propojena pomocí dvanácti digitálních pinů (3-13) a pomocí dvou analogových pinů A0 a A1. Piny A0 a A1 jsou však využity jako digitální vstupy a výstupy, což deska Arduino UNO umožnuje. Piny A0 a A1 jsem využil jako náhradu za digitální piny 0 a 1, které jsou vyhrazeny pro TX a RX signály sériové komunikace. Při jejich využití nastávali během využívání sériové linky ke komunikaci s PC neošetřené stavy a celé řešení bylo díky tomu značně nestabilní.

Konkrétní využití jednotlivých pinů je vidět na nákresech výše a zároveň jsou popsány v následujících kapitolách týkajících se jednotlivých komponent.

4.2.1.3 LCD displej

Realizované řešení využívá alfanumerický LCD displej o rozměrech 16x2 znaků. LCD displej slouží jako výstup pro zobrazení menu aplikace a aktuálním stavu probíhajícího čerpání.

LCD displej je k vývojové desce připojen paralelně pomocí šesti vodičů (Arduino Piny 04–09). Alternativní možnost připojení pomocí I2C sběrnice jsem se rozhodl nevyužít.

Přínosem propojení displeje pomocí I2C sběrnice řešení by byla úspora vstupně/výstupních pinů vývojové desky. Nevýhodou by pak byla nutnost zařadit I2C převodník. Zařazením tohoto převodníku by došlo k mírnému navýšení nákladů.

Vzhledem k dostatečnému množství pinů vývojové desky pro mé řešení nebyla popisovaná výhoda pro řešení relevantní.

4.2.1.4 Sada tlačítek

Řešení obsahuje sadu tří tlačítek pro ovládání řešení. Tlačítka slouží pro navigaci v menu, ke spuštění a zastavení jednotlivých funkcí řídící jednotky.

Tlačítka jsou připojena pomocí digitálních pinů 3 a 2 a pomocí analogového pinu A1, který je v programu přenastaven jako digitální vstup.

Tlačítka jsou s deskou propojena pomocí integrovaného Pull-up rezistoru[3]. Díky tomu je možné tlačítka zapojit přímo na digitální vstupy bez připojení externího rezistoru.

4.2.1.5 H-můstek – modul L9110S

Jak bylo zmíněno již v kapitole 3.2.2 h-můstek slouží k řízení bipolárního krokového motoru. Pro mé řešení jsem použil konkrétně modul L9110S, který umožňuje pomocí 4 vstupních pinů ovládat jednotlivé fáze motoru. K samotnému krokovému motoru je připojen pomocí čtyř vodičů, dva pro každou fázi. (H-můstek modul L9110S, 2016)

H-můstek zároveň také umožňuje napájet krokový motor odděleně od ostatní logiky řídící jednotky v rozmezí vstupního napětí 2.5–12 V a maximálního proudu 0.8A na každou fázi.

H-můstek je s deskou propojen pomoc digitálních pinů 10–13.

4.2.1.6 Krokový motor

Krokový motor v nákresu zastupuje celé čerpadlo, které je řídící jednotkou obsluhováno. Jedná se o čerpadlo od firmy Welco typu WP1000.

Čerpadlo je poháněno bipolárním krokovým motorem se dvěma fázemi (označení FB ve WP1000 Peristaltic pump). Jmenovité napětí motoru je 1.92 V a jmenovitý proud je 0.8 A na fázi.

[3] Jedná se rezistor integrovaný uvnitř čipu, který je zapojen mezi digitální vstup a +5V. Digitální pin je tedy přest tlačítko propojen s GND Výchozí hodnota na vstupu je HIGH po stisku tlačítka je hodnota LOW. (Voda, 2017)

Maximální rychlost otáček čerpadla je 20 otáček za minutu. Úhel kroku čerpadla je 0,028125° což odpovídá 12 800 krokům na jednu otáčku. Mezi samotným motorem a čerpadlem je umístěna převodová soustava 1:64. Výsledné hodnoty motoru, které budou důležité pro ovládání čerpadla tedy jsou následující:

- 12 800 / 64 => 200 (úhel 1,8°)
- 20*64 => 1280 otáček za minutu

(WP1000 Peristaltic pump)

4.2.2 Popis SW řešení řídící jednotky

Tato kapitola popisuje programovou část řešení. Popíšu jednotlivé funkční části celého programu a jejich vzájemnou kooperaci.

4.2.2.1 Úvodní inicializace

Při každém startu aplikace je nutné provést úvodní inicializaci a nastavení prostředí. Nejdříve popíšu definované konstanty prostředí, definované globální proměnné a veškeré operace, které se provádí při startu aplikace (definované v rámci funkce **setup**).

4.2.2.1.1 Použité konstanty

Pro definování konstant používám standardní řešení pomocí #define:

```
#define NAZEV_KONSTANTY HODNOTA
```

Definované konstanty včetně popisu uvádím v následující tabulce:

Tabulka 10: Seznam definovaných konstant

Název konstanty	Hodnota	Popis použití
RESET	A0	Číslo pinu připojeného na signál RESET (analogový pin A0)
BACK_BTN	A1	Číslo pinu připojeného na tlačítko zpět (analogový pin A1)
SEL_BTN	2	Číslo pinu připojeného na tlačítko Enter / OK
NAV_BTN	3	Číslo pinu připojeného na navigační tlačítko
MOT1	10	Jednotlivé piny připojené k H-můstku ovládající krokový motor čerpadla
MOT2	11	
MOT3	12	
MOT4	13	
MAX_DEPTH	2	Maximální zanoření menu (počet úrovní menu)

SOFT_BOUNCE_MS	100	Interval mezi stisky tlačítek
STP_ADR 1	1	Adresa v paměti pro uložení hodnoty počtu kroků pro jednu desetinu ml
SPD_ADR	3	Adresa v paměti pro uložení defaultní hodnoty průtoku
MOD_ADR	7	Adresa v paměti pro uložení defaultní hodnoty režimu čerpání
TOPSPD_ADR	9	Adresa v paměti pro uložení maximální hodnoty průtoku
MINSPD_ADR	13	Adresa v paměti pro uložení minimální hodnoty průtoku
INIT_ADR	17	Adresa v paměti pro uložení příznaku prvního spuštění
VOL_ADR	21	
MOTSPDL	1	Hodnota minimální rychlosti krokového motoru
MOTSPDH	1280	Hodnota maximální rychlosti krokového motoru
MOTSTEPS	200	Počet kroků krokového motoru v jedné otáčce

Zdroj: vlastní zpracování

4.2.2.1.2 Globální proměnné a objekty

Vyvinuté řešení používá sadu globálních proměnných a objektů uvedených v následující tabulce. Kromě zde zmíněných objektů a proměnných je celé použité menu definované jako sada globálních objektů (viz kapitola 4.2.2.2 Menu aplikace).

Tabulka 11: Seznam používaných globálních proměnných a objektů

Název proměnné / objektu	Datový typ	Popis použití
INIT	Char	Příznak, zda byla aplikace spuštěna poprvé, při deklaraci je načtena hodnota z paměti EEPROM
topSpeed	Float	Maximální hodnota průtoku, při deklaraci (spuštění aplikace) je načtena hodnota z paměti EEPROM.
minSpeed	Float	Minimální hodnota průtoku, při deklaraci (spuštění aplikace) je načtena hodnota z paměti EEPROM.
stepsPerDeciMl	Unsigned integer (uint16_t)	Počet kroků potřebných k načerpání desetiny ml, při deklaraci je načtena hodnota z paměti EEPROM.
endTime	Unsigned long	Čas ukončení čerpání, využívaný pro kalibraci.

mode	Char	Příznak nastaveného režimu čerpání.
defMode	Char	Výchozí příznak nastaveného režimu čerpání.
spd	Float	Průtok čerpadla v ml/min.
spdDef	Float	Výchozí hodnota průtoku čerpadla v ml/min.
Volf	Float	Cílový objem čerpání v ml.
volCalf	Float	Cílový objem kalibrace v ml.
volDeff	Float	Výchozí hodnota cílového objemu v ml.
stopPump	Boolean (bool)	Příznak zastavení čerpání.
lcd	LiquidCrystal	Objekt pro řízení LCD displeje. Součástí definice je seznam čísel výstupních pinů připojených na displej.
motor	Stepper	Objekt pro řízení krokového motoru. Deklaruje se pomocí konstruktoru Stepper. Parametry konstruktoru: Počet kroků na jednu otáčku Seznam čísel výstupních pinů pro řízení motoru

Zdroj: vlastní zpracování

4.2.2.1.3 Úvodní operace

Veškeré úvodní operace se provádí v defaultní funkci **setup**, která je spuštěna vždy při spuštění aplikace. V rámci funkce **setup** provádím následující operace:

- Zapnutí podpory přerušení pomocí funkce **interrupts.**
- Inicializace a nastavení vstupních a výstupních pinů.
- V případě, že jde o první spuštění aplikace, zavolá se funkce **initSets.**
    - V rámci funkce dochází k nastavení všech hodnot v paměti EEPROM na výchozí hodnoty a nastavení hodnoty INIT v paměti na 1
    - Zdrojový kód funkce **initSets** je zobrazen níže pod výpisem operací
- Spuštění LCD displeje pomocí metody **begin** na objektu *lcd*
- Načtení defaultních hodnot

- Podrobný popis je součástí kapitoly 4.2.2.5 Možnost uložení a načtení výchozích hodnot

Zdrojový kód funkce **initSets**:

```
void initSets() { //Nastavení hodnot při prvním spuštění na konkrétní desce nebo po smazání EEPROM paměti

  EEPROM.put (VOL_ADR,10.0);
  EEPROM.update (STP_ADR, 1);
  EEPROM.update (STP_ADR + 1, 0);
  EEPROM.put (SPD_ADR,1.0);
  EEPROM.update (MOD_ADR, 1);
  EEPROM.put (TOPSPD_ADR, 10.0);
  EEPROM.put (MINSPD_ADR, 1.0);
  EEPROM.update (INIT_ADR, 1);
  digitalWrite (RESET, LOW);
}
```

Zdrojový kód funkce **setup:**

```
void setup() {
  interrupts(); // zapnutí podpory přerušení
  digitalWrite(RESET, HIGH);
  delay(200);
  pinMode (RESET, OUTPUT);
  if (INIT != 1) {
    initSets(); // inicializace při prvním spuštění aplikace
  }
  //konfigurace vstupních a výstupních pinů
  pinMode (NAV_BTN, INPUT_PULLUP);
  pinMode (SEL_BTN, INPUT_PULLUP);
  pinMode (BACK_BTN, INPUT_PULLUP);

  pinMode (MOT1, OUTPUT);
  pinMode (MOT2, OUTPUT);
  pinMode (MOT3, OUTPUT);
  pinMode (MOT4, OUTPUT);

  lcd.begin (16, 2);

  readSet();
  volf = volDeff;
  spd = spdDef;
  mode = defMode;

}
```

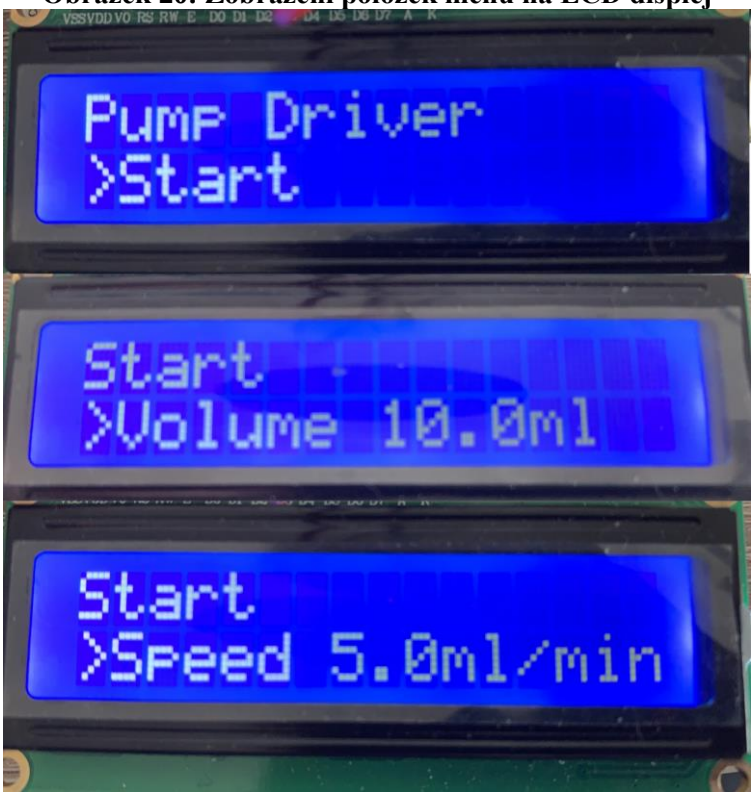
4.2.2.2 Menu aplikace

Základem celého řešení je menu aplikace, které má následující strukturu:
Pump Driver:

- Start
  - Mode – přepínač sloužící pro nastavení módu čerpání
  - Volume – nastavení hodnoty cílového objemu v ml (proměnná *volf*)
  - Speed – nastavení rychlosti čerpání (průtoku) v ml/min (proměnná *spd)*
  - Run – spouští čerpání
- Calibration
  - Volume – nastavení kalibračního objemu (proměnná *volCalf*)
  - Calibrate – spouští proces kalibrace
- PC Control
  - Connect – slouží k zahájení spojení s PC přes sériovou linku
- Settings – submenu slouží k nastavení výchozích hodnot
  - Def. mode – přepínač sloužící pro nastavení výchozího módu čerpání (proměnná modeDef)
  - Def. vol. – nastavení výchozí hodnoty objemu v ml (proměnná volDeff)
  - Def. spd. – nastavení výchozí hodnoty průtoku v ml/min (proměnná spdDef)
  - Save – uloží nastavené hodnoty do trvalé paměti

Příklad zobrazení některých položek menu na LCD displeji si můžete prohlédnout na obrázku níže - Obrázek 20.

Obrázek 20: Zobrazení položek menu na LCD displej



Zdroj: vlastní zpracování

Pro implementaci menu jsem použil knihovnu ArduinoMenu 4 (Azevedo, 2019). Knihovna umožňuje sestavit libovolné vlastní menu využitím několika základních strukturálních prvků. Pro mé potřeby jsem využil následující prvky:

Menu Node

Jedná se o zastřešující prvek celého menu, který je možné využít i pro definování submenu. V mém vlastním řešení je definováno celkem pět menu node:

- topMenu – hlavní zastřešující prvek celého menu
- startMenuData – submenu Start
- calibMenu – submenu Calibration
- setMenu – submenu Settings
- pcCtrlMenu – submenu PC Control

Pro definici menu nebo submenu jsem vždy nejdříve vytvořil pole obsahující veškeré prvky daného menu, o kterých se budu zmiňovat dále v kapitole. Následně jsem vytvořil objekt menu pomocí konstruktoru níže:

```
menuNode(constText* text, idx_t sz, prompt* constMEM data[], action
a=noAction, eventMask e=noEvent, styles style=wrapStyle, systemStyles
ss=(systemStyles)
```

- text – zobrazovaný název menu nebo submenu
- sz – počet prvků menu
- data – pole definovaných prvků menu
- a – definuje akci, kterou je možné spustit při výběru menu, osobně jsem v řešení nevyužíval a ponechal jsem výchozí chování – vstup do submenu
- e – eventMask – slouží k upřesnění chování prvku, u menu node jsem nevyužíval
- s – style – umožňuje definovat, zda se má menu po dosažení konce seznamu vrátit zpět na začátek – wrapStyle, či nikoliv – noStyle

Vlastní příklad definice menu uvádím níže. Jedná se konkrétně o submenu Start.

```
prompt* startMenuData[] = {
  &modeMenu,

  new decimalslField<typeof(volf)>(volf, "Volume", "ml", 1.0, 5000.0,
0.1, 1, doNothing, noEvent, wrapStyle),

  new menuField<typeof(spd)>(spd, "Speed", "ml/s" ,minSpeed, topSpeed,
1, 5, doNothing, noEvent, wrapStyle),

  new prompt ("Run", (Menu::callback) runPump, enterEvent),
};

menuNode& startMenu = *new menuNode(
                        "Start",
                        sizeof(startMenuData) / sizeof(prompt*),
                        startMenuData,
                        doNothing,
                        noEvent,
                        wrapStyle
                      );
```

Prompt (OP)

Základní prvek struktury menu, umožňující vyvolat libovolnou funkci nebo akci. V mém řešení je využívána pro spuštění čerpání (Start -> Run), kalibrace (Calibration – Calibrate), pro uložení nastavení (Settings -> Save) nebo pro zahájení UART spojení s PC (PC Connect -> Connect).

Definici Promptu jsem prováděl vždy při definování pole prvků pro konkrétní submenu nebo menu. K vytvoření jsem využil následující konstruktor

```
prompt(constText* t, action a=doNothing, eventMask e=noEvent, styles
s=noStyle, systemStyles ss=_noStyle)
```

- t – zobrazovaný název prvku
- a – action – slouží k definici akce, která se má provést po výběrů volby v menu nebo pro odkaz na funkci, která má být zavolána
- e – eventMask – slouží k upřesnění chování prvku, případně pro předání vstupních parametrů funkce, tuto možnost jsem ve svém řešení nevyužil.
- s – style – v řešení jsem nevyužíval a nechával jsem nastavenou výchozí hodnotu

Příklad definice prvku prompt jsme mohli vidět na příkladu definice menu výše, jako součást definice prvků menu.

Menu Field

Speciální prvek sloužící k editaci proměnné. Tento prvek mi v řešení sloužil k veškerému nastavování hodnot. Prvek Menu Field má hned několik variant využití. V rámci mého řešení jsem využil konkrétně dvě.

- **Toggle** – sloužící k výběru z předdefinovaných hodnot
- **Field** – sloužící k nastavení hodnoty proměnné v definovaném rozsahu.

Každá z variant má mírně odlišnou definici. Toggle se definuje podobně jako menu. Nejdříve je nutné vydefinovat seznam hodnot, které se pak použijí. Pro definici seznamu jsem využil připravené makro:

```
TOGGLE(var.name, id, title, action, event mask, styles, value, value
[,value ...])
```

- var.name – proměnná která bude prvkem měněna
- id – identifikátor prvku
- title – zobrazovaný název
- action, event mask, styles – mají stejný význam jako u menu
- value – jednotlivé hodnoty menu, definované následujícím makrem

```
VALUE(title, value, action, event)
```

- title – zobrazovaný text hodnoty
- value – hodnota proměnné, která bude nastavena v případě výběru této volby
- action – nevyužíval jsem vždy jsem nechal výchozí nastavení doNothing
- event – nevyužíval jsem vždy jsem nechal výchozí nastavení noEvent

Takto vydefinovaný **Toggle** se následně při definici menu zavolá pomocí reference na id. Příklad definice a použití **Toggle** uvádím níže:

```
TOGGLE (mode, modeMenu, "Mode: ", doNothing, noEvent, wrapStyle
       , VALUE ("Continuous", 1, doNothing, noEvent)
       , VALUE ("Specific volume", 2, doNothing, noEvent)
      );

prompt* startMenuData[] = {
  &modeMenu,
  new        decimalslField        <typeof        (volf)>        (volf,
"Volume","ml",1.0,5000.0,0.1,1, doNothing, noEvent, wrapStyle),
,
  new  menuField  <typeof  (spd)>  (spd,  "Speed",  "ml/s",  minSpeed,
topSpeed, 1, 5, doNothing, noEvent, wrapStyle),

  new prompt("Run", (Menu::callback) runPump, enterEvent ),
};
```

Field se definuje využitím template menuField:

```
menuField (T &value, constText* text, constText*units, T low, T high, T
step, T tune, action a=doNothing, eventMask e=noEvent, styles s=noStyle)
```

- T – zástupný název za datový typ
- &value – reference na proměnou, kterou bude prvek menu editovat
- text – zobrazovaný název položky
- units – jednotky měněné veličiny
- low – nejnižší možná hodnota rozsahu
- high – nejvyšší možná hodnota rozsahu
- step – hodnota kroku, při změně hodnoty
- tune – druhá hodnota kroku při změně hodnoty
- action, event – tyto hodnoty jsem pro definici nevyužíval
- style – definuje, zda se má po dosažení konce rozsahu pokračovat do začátku (wrapStyle) nebo ne (noStyle)

Příklad definice prvku Field je vidět na příkladu definice menu výše, jako součást definice prvků menu. Zároveň je zde uvedena i definice typu decimalsField, který má stejné vlastnosti jako menuFileld, ale slouží pro editaci proměnné typu float. Toto řešení jsem převzal z příkladů uvedených u dokumentace knihovny. (Azevedo, 2019) Všechny prvky menu i menu samotné jsou definovány jako globální proměnné.

Kromě definice samotné struktury menu, je nutné také zajistit zobrazení menu na displeji a zajistit možnost jeho ovládání. Jak je zmíněno v předešlých kapitolách k zobrazení výstupu jsem využil LCD znakový displej 16x2. Pro definici zobrazení výstupu menu na displeji stačí využít předdefinované makro. Definice je znázorněna níže.

```
MENU_OUTPUTS (out, MAX_DEPTH
              , NONE //makro slouží k definici dvou výstupů
              , LIQUIDCRYSTAL_OUT (lcd, {0, 0, 16, 2})
             );
```

Pro ovládání menu pak slouží objekt (v mém případě konkrétně objekt *nav)* typu navRoot, který se opět definuje pomocí připraveného makra:

```
NAVROOT (nav, topMenu, MAX_DEPTH, in, out);
```

- nav – identifikátor navigačního objektu
- topMenu – reference na ovládané menu
- MAX_DEPTH – maximální hloubka zanoření menu
- in – reference na vstupní stream (např. pro vstupy ze sériové linky)
    - vstupní stream jsem v řešení nevyužil, pro využití makra jsem tedy pouze referencoval prázdný stream
- out – reference na objekt zajišťující výstup menu na displej
    - vytvořený pomocí makra MENU_OUTPUTS (viz výše)

Samotné ovládání je v mém řešení realizování voláním metody **doNav** nad objektem *nav* typu navRoot, jehož jediným parametrem je kód příkazu. Tyto metody v řešení volám v hlavní části programu **loop** v případě stisknutí odpovídajícího tlačítka. Aby se změna projevila na výstupu, je vždy po každém zavolání metody **doNav**, nutné také zavolat metodu **doOutput**, které je bez parametrů a zajistí aktualizaci menu na displeji. Seznam mnou použitých kódů:

- escCmd – opuštění menu, vstup do nadřazené úrovně
- enterCmd – výběr možnosti, vstup do submenu, potvrzení
- upCmd – posun v menu, změna hodnoty

(Azevedo, 2019)

Zdrojový kód realizující ovládání menu uvádím zde – poupravená verze převzatá z příkladů:

```
void loop () {

  if (!digitalRead (SEL_BTN)) {
    delay (SOFT_DEBOUNCE_MS);
    while (!digitalRead (SEL_BTN)); //čekání na uvolnění tlačítka
    nav.doNav (enterCmd);
    delay (SOFT_DEBOUNCE_MS);
  }
  if (!digitalRead (NAV_BTN)) {
    delay (SOFT_DEBOUNCE_MS);
    while (!digitalRead (NAV_BTN));
    nav.doNav(upCmd);
    delay(SOFT_DEBOUNCE_MS);
  }
  if (!digitalRead(BACK_BTN) && &(nav.active()) != &topMenu){
    delay (SOFT_DEBOUNCE_MS);
    while (!digitalRead (BACK_BTN));
    nav.doNav (escCmd);
    delay(SOFT_DEBOUNCE_MS);
  }
  nav.doOutput ();
}
```

(Azevedo, 2019)

4.2.2.3 Funkce čerpání

Pro zajištění funkcionality čerpání ve dvou popsaných režimech (kontinuální a přesné čerpání – viz 4.1.1 Analýza požadavků a návrh cílového řešení) jsem pro každý režim čerpání vytvořil jednu funkci. Pro průběžné čerpání využívám funkci cPump a pro přesné čerpání sPump. Definice těchto funkcí je následující:

- void **cPump** (int pumpSpeed, bool serial = false)
- void **sPump** (int pumpSpeed, float pumpVolume, bool serial = false)

Význam jednotlivých argumentů funkcí:

- **pumpSpeed** – požadovaný průtok v ml/min
- **pumpVolume** – cílový požadovaný objem v ml
- **serial** – příznak, zda má být výstup o průběhu čerpání zobrazen na displeji nebo odeslán pomocí sériové linky

Funkce **cPump** a **sPump**, které podrobně popisuji v následujících podkapitolách jsou volány z dvou možných míst programu. První možností volání funkcí je z funkce **runPump**, která je spuštěna při výběru volby (Start – Run) z menu aplikace a slouží k načtení nastavených hodnot a rozhodnutí jaký režim čerpání má být spuštěn. Druhou možností je zavolání jedné z funkcí prostřednictvím funkce **doSerialCommand**, která slouží ke zpracování příkazu ze sériové linky a podrobně ji budu popisovat v kapitole 4.2.2.6 Funkce ovládání pomocí počítače.

Funkce pracují s globálním objektem *motor*, který jsem zmínil v kapitole 4.2.2.1 Úvodní inicializace.

4.2.2.3.1 runPump

Jak jsem zmínil výše funkce **runPump** je spuštěna po potvrzení volby Start – Run z menu aplikace. Funkce pomocí globálních parametrů nastavovaných (*mode*, *spd*, *volf*) v menu zavolá funkci:

- **cPump**, pokud *mode* = 1
- **sPump**, pokud *mode* se nerovná 1

Funkce **sPump**, příp **cPump** je volána s parametry:

- *pumpSpeed* = *spd*
- *pumpVolume* = *volf*

Zdrojový kód funkce **runPump** uvádím níže:

```
result runPump () {
  attachInterrupt(0, stopPumping, RISING);
  lcd.setCursor(0, 1);
  lcd.print("Stop      ");
  stopPump = false;
  if (mode == 1)
    cPump(spd);
  else
    sPump(spd, volf);
  lcd.setCursor(0, 0);
  lcd.print("Done!  ");
  lcd.setCursor(0, 1);
  lcd.print("Run again");
  detachInterrupt(0);
  return proceed;
}
```

4.2.2.3.2 cPump

Na začátku funkce nejdříve dochází pomocí funkce **map** k převodu požadovaného průtoku (proměnná *spd*) na požadovanou rychlost motoru. Funkce map slouží k přepočítání hodnoty z rozsahu A-B na odpovídající hodnotu z rozsahu X-Y. Využívají se hodnoty získané při kalibraci čerpadla o maximálním a minimálním možném průtoku a maximální a minimální otáčky motoru. Výsledná hodnota se následně použije pro nastavení rychlosti motoru pomocí metody **setSpeed** objektu *motor*.

Po nastavení rychlosti je spuštěn **do-while** cyklus, který je základem celé funkce. V každém opakování vypíše aktuálně načerpanou hodnotu na displej nebo sériovou linku, dle hodnoty parametru *seriál*, a zavoláním metody **step** s parametrem *stepsPerDeciMl* nad objektem *motor* otočí motor o takový počet kroků, aby došlo k přečerpání jedné desetiny ml. Hodnota *stepsPerDeciMl* je jedním z výstupů kalibrace popisované v kapitole 4.2.2.4 Funkce kalibrace. V každém opakování cyklu je navýšena hodnota načerpané kapaliny *currentVolume* o jednu desetinu ml.

Výše zmíněný cyklus probíhá, dokud není parametr *stopPump* nastaven na hodnotu true. Parametr je nastaven po stisknutí konkrétního tlačítka. Funkce zastavení čerpání je implementována pomocí HW přerušení. Podrobně je funkcionalita popsána v kapitole 4.2.2.7 Okamžité ukončení čerpání. Zdrojový kód funkce cPump uvádím níže:

```
void cPump (int pumpSpeed, bool serial = false) {
      float currentVolume = 0.0;
      uint16_t motorSpeed = map (pumpSpeed, minSpeed, topSpeed, MOTSPDL,
MOTSPDH);
      motor.setSpeed(motorSpeed);
currentVolume += (0.1);
   do {
       motor.step (stepsPerDeciMl);
      if (serial){
            Serial.println ("Pumped: ");
            Serial.print (currentVolume, 2);
Serial.print("ml");
      }
     else{
            lcd.clear();
            lcd.print ("Pumped: ");
            lcd.print (currentVolume,2);
lcd.print("ml      ");
      }
   }while (!stopPump);
}
```

4.2.2.3.3 sPump

Funkce **sPump** se od **cPump** popsané v předchozí kapitole liší zejména podmínkou na konci cyklu, kdy se kontroluje, zda již bylo dosaženo cílového objemu (porovnání proměnných *pumpVolume* a *currentVolume*). V případě, že ano, dojde k nastavení hodnoty stopPump na true. Čerpání již tedy dále nepokračuje. Čerpání je možné také zastavit před dosažením této hodnoty stisknutím tlačítka pro zastavení čerpání. Zdrojový kód funkce sPump uvádím níže:

```
void sPump(int pumpSpeed, float pumpVolume, bool serial = false){
  float currentVolume = 0.0;
  int motorSpeed = map (pumpSpeed, minSpeed, topSpeed, MOTSPDL,
MOTSPDH);
  motor.setSpeed (motorSpeed);
  while (!stopPump) {
     lcd.setCursor(0, 0);
      motor.step(stepsPerDeciMl);
currentVolume += 0.1;

      if (serial){
            Serial.println ("Pumped: ");
            Serial.print (currentVolume,2);
Serial.print("/");
Serial.print(pumpVolume,1);
Serial.print("ml");
Serial.println("");
         }
     else{
            lcd.print("Pumped: ");
            lcd.print(currentVolume,2);
            lcd.print("/");
lcd.print(pumpVolume,1);
lcd.print("ml");
     }
      if (currentVolume >= pumpVolume) {
            stopPump = true;
      }
  }
}
```

4.2.2.4 Funkce kalibrace

Funkce kalibrace slouží k získání a uložení nastavení hodnot minimálního a maximálního průtoku vzhledem k nastavené rychlosti motoru a k získání počtu kroků motoru potřebného k načerpání jedné desetiny ml. Průběh kalibrace je podrobně popsán v kapitole 4.1.1.3.2 UC02 Kalibrovat čerpadlo.

Kalibrace se skládá ze dvou kroků na jejichž základě dojde k výpočtu parametrů. Jedná se o načerpání předem definovaného objemu kapaliny pomocí:

- Nejnižší možné rychlosti motoru (konstanta *MOTSPDL*)
- Nejvyšší možné rychlosti motoru (konstanta *MOTSPDH*)

Po ukončení dvojice čerpání dojde k výpočtu průtoku pro obě rychlosti motoru a k výpočtu počtu kroků potřebných pro načerpání jedné desetiny ml.

Hlavní logiku kalibraci zajišťuje dvojice funkcí **runCalibration** a **calibPump.** Definice obou funkcí je zde:

- void **runCalibration** (bool serial)
- unsigned long **calibPump** (int motorSpeed, unsigned long * steps, bool serial=false)

Funkce **runCalibration** zajišťuje celý proces kalibrace. Funkce **calibPump** provádí kalibrační čerpání s definovanou rychlostí motoru. Podrobnější popis jednotlivých funkcí uvádím v podkapitolách níže. Význam jednotlivých argumentů funkcí uvádím níže:

- serial – příznak, zda se má výstup funkce odesílat na sériovou linku nebo vypisovat na LCD displej
- motorSpeed – rychlost motoru s kterou se má spustit kalibrační čerpání
- steps – ukazatel na proměnou, do které bude uložen počet kroků, provedených během kalibračního čerpání

4.2.2.4.1 runCalibration

Jak jsem již zmínil funkce **runCalibration** zajišťuje celý proces kalibrace. Funkce může být spuštěna ze dvou míst programu. Prvním místem je funkce **doSerialCommand** (parametr *serial* je nastaven na hodnotu *true*), zajišťující zpracování příkazů zadaných pomocí sériové linky (viz kapitola 4.2.2.6). Druhou možností je zavolání funkce při potvrzení volby Calibration -> Run z menu. Funkce je volána zprostředkovaně pomocí funkce **runCalib**, která doplní požadovaný parametr *serial = false.*

Funkce na začátku vypíše název prvního kroku kalibrace – "Step 1 - low speed“. Výpis je dle parametru *serial* proved buď na LCD displej nebo odeslán pomocí sériové linky. Program následně čeká vstup uživatele potvrzující start prvního kroku kalibrace:

- Stisknutí potvrzovacího tlačítka (*SEL_BTN*), pokud *serial = false*
- Stisknutí klávesy Enter na PC, pokud *serial = true*

Po potvrzení uživatele je zavolána funkce **calibPump** s následujícími parametry (viz následující kapitola) v rámci niž probíhá první kalibrační čerpání. Parametr *motorSpeed* je nastaven na hodnotu *MOTSPDL* (nejnižší možná rychlost motoru). Parametru *steps* je předána lokálně deklarovaná proměnná *stepsL*. Poslednímu parametru serial je předána stejná hodnota, s jakou byla zavolána funkce **runCalibration**.

Návratová hodnota funkce **calibPump** (celkový čas kalibračního čerpání) je uložen do lokálně deklarované proměnné *calibTimeL*.

Stejný proces je proveden i pro čerpání s rychlostí motoru *MOTSPDH* (nejvyšší možná rychlost motoru. Název kroku kalibrace je "Step 2 - high speed", počet kroků je uložen do lokální proměnné *stepsH* a doba čerpání je uložena do proměnné *calibTimeH*.

Zdrojový kód popsané části funkce uvádím zde:

```
if (serial){
    Serial.println("Step 1 - low speed ");
    Serial.println("Press Enter ...");
    char c;
    while(c!=10 && c!=13 && c!=64){
      if(Serial.available()){
      c = Serial.read();
      }
    }
  }
  else{
    lcd.clear();
    lcd.print("Step 1 - low speed ");
    lcd.setCursor(0, 1);
    lcd.print(">Start calib. step.");
    delay(SOFT_DEBOUNCE_MS);
    while (digitalRead(SEL_BTN)) {
    delay(SOFT_DEBOUNCE_MS);
    }
  }
  unsigned long stepsL = 0;
  unsigned long calibTimeL = calibPump(MOTSPDL, &stepsL,serial);

  if(serial){
    Serial.println("Step 2 - high speed");
    Serial.println("Press Enter ...");
    char c;
    while(c!=10 && c!=13 && c!=64){
      if(Serial.available()){
      c = Serial.read();
      }
    }
   }
  else{
    lcd.clear();
    lcd.print("Step 2 - high speed");  //tady bude krok motoru
    lcd.setCursor(0, 1);
    lcd.print(">Start calib. step.");
    delay(SOFT_DEBOUNCE_MS);
    while (digitalRead(SEL_BTN)) {
      delay(SOFT_DEBOUNCE_MS);
    }
  }
```

Po ukončení i druhého kroku kalibrace jsou vypočítány hodnoty průtoku v ml/min jako poměr kalibračního objemu *volCalf* (globální proměnná nastavena v menu nebo pomocí příkazové řádky) a hodnoty calibTimeH (původní hodnota v ms, je převedená na minuty vydělením hodnotou 60000). Tyto hodnoty jsou uloženy do globálních proměnných *minSpeed* a *topSpeed.*

Po výpočtu hodnot provede program validaci výsledků kalibrace pomocí porovnání, zda hodnota *minSpeed* není větší než *topSpeed,* zda si nejsou hodnoty rovny nebo zda není hodnota *minSpeed* rovna nule. Pokud je jedna z těchto podmínek splněna dojde k vypsání hlášky „Calibration failed.“ a nastavení výchozích hodnot *minSpeed* = 1 a *topSpeed* = 10.

Po validaci dojde k výpočtu kroků pro načerpání jedné desetiny ml. Výpočet je proveden jako poměr průměrného počtu provedených kroků z obou kalibračních čerpání vůči kalibračnímu objemu (proměnná *volCalf*) vynásobenému deseti (převod ml na desetiny ml).

Posledním krokem funkce je uložení kalibračních hodnot do paměti pomocí funkcí metod **put** a **update** objektu *EEPROM a* restart programu po skončení 5s odpočtu nastavením výstupního pinu *RESET* na hodnotu *Low*.

Zdrojový kód druhé části funkce uvádím zde:

```
unsigned long stepsH = 0;
unsigned long calibTimeH = calibPump(MOTSPDH, &stepsH,serial);

  topSpeed = volCalf / ((float)calibTimeH / 60000.0);
  minSpeed = volCalf / ((float)calibTimeL / 60000.0);
  bool failed = false;
  if(minSpeed>=topSpeed){
    if(serial){
      Serial.println("Calibration failed.");
    }
    else{
      lcd.clear();
      lcd.print("Calibration failed.");
      lcd.setCursor(0, 1);
    }
      minSpeed = 1.0;
      topSpeed = 10.0;
      failed = true;
  }
  else if(minSpeed == 0.0){
      minSpeed = 1.0;
  }
uint16_t volDeciML = (uint16_t)(volCalf * 10);
unsigned long stepsAvr = (stepsL+stepsH)/2;
stepsPerDeciMl = stepsAvr / (unsigned long)(volCalf * 10);

if(serial){
    Serial.println(calEnd);
    Serial.println("Restart in: ");
    for (int i = 5; i > 0; i--) {
      Serial.print(i);
      Serial.print(" s");
      Serial.println();
      delay(1000);
    }
}
else{
    if(!failed){
      lcd.clear();
      lcd.print(calEnd);
    }
    lcd.setCursor(0, 1);
    lcd.print("Restart in: ");
    for (int i = 5; i > 0; i--) {
      lcd.print(i);
      lcd.print(" s");
      lcd.setCursor(12, 1);
      delay(1000);
    }
}
  EEPROM.put(TOPSPD_ADR, topSpeed);
  EEPROM.put(MINSPD_ADR, minSpeed);
  EEPROM.update(STP_ADR, stepsPerDeciMl % 256);
  EEPROM.update(STP_ADR + 1, stepsPerDeciMl / 256);
  digitalWrite(RESET, LOW);
```

4.2.2.4.2 calibPump

Jak jsem popsal v předchozí kapitole funkce **calibPump** slouží ke spuštění kalibračního čerpání.

Na začátku funkce je registrována funkce přerušení (viz 4.2.2.7 Okamžité ukončení čerpání) aby bylo možné zastavit čerpání pomocí tlačítka. V dalším kroku je pomocí metody **setSpeed** nad objektem *motor* nastavena rychlost otáčení motoru na hodnotu vstupního parametru *motorSpeed*.

Před zahájením čerpání je pomocí funkce **millis** načten údaj o době běhu programu do proměnné *startTime*. Tento údaj bude využit k vypočítání doby kalibračního čerpání. Samotné čerpání je prováděno v rámci cyklu **do-while.** V každém průběhu cyklu jsou vypsány aktuální informace na LCD displej nebo sériovou linku (dle parametru *serial*):

- Čas čerpání
- Počet provedených kroků

Zároveň je v každém opakování cyklu otočeno motorem o jeden krok pomocí metody **step** nad objektem *motor* a je zvýšena hodnota vstupního parametru *steps* o jedna.

Cyklus je ukončen, pokud je globální proměnná *stopPump* nastavena na *true*. Hodnota je nastavena v rámci obsluhy přerušení (funkce **stopPumping**) v případě stisknutí potvrzovacího tlačítka (SEL_BTN). Nebo v rámci prováděného cyklu v případě stisknutí klávesy Enter, pokud byla kalibrace spuštěna z příkazové řádky. Po ukončení čerpání je do globální proměnné endTime uložena aktuální hodnota doby programu pomocí funkce **millis.** Po ukončení cyklu dojde k odpojení obsluhy přerušení a je funkce vrátí rozdíl hodnot *endTime* a *startTime*.

Ukázku zdrojového kódu funkce calibPump uvádím zde:

```
unsigned long calibPump(int motorSpeed, unsigned long * steps, bool
serial=false) {
  while (!digitalRead(SEL_BTN)) {}
  attachInterrupt(0, stopPumping, RISING);
  motor.setSpeed(motorSpeed);
  stopPump = false;
  unsigned long currentTime, startTime = 0;
  startTime = millis();
  * steps = 0;
  uint16_t stepsPre = 0;
  if(serial){
    Serial.println("Press enter at end");
    }
  do {
   currentTime = (millis() - startTime) / 1000;
   if(serial){
    char c;
      Serial.print("Time: ");
      Serial.print(currentTime); Serial.print(" s");
      Serial.print(" | ");Serial.print(* steps); Serial.println();
      if(Serial.available()){
      c = Serial.read();
      if(c==10 || c==13 || c==64){
        stopPump = true;
        endTime=(currentTime*1000)+startTime;
        }
      }
   }
   else{
    lcd.clear();
    lcd.print("Time: "); lcd.print(currentTime); lcd.print(" s");
    lcd.print(" | "); lcd.print(* steps); lcd.setCursor(0, 1);
    lcd.print(">Stop(Vol. reached)");
   }
    motor.step(1); * steps += 1;
  } while (!stopPump);
  detachInterrupt(0);
  return endTime - startTime;
}
```

4.2.2.5 Možnost uložení a načtení výchozích hodnot

Jelikož hodnoty veškerých proměnných zůstávají uloženy v napěťově závisle operační paměti, při každém vypnutí či restartu řídící jednotky dojde ke smazání těchto hodnot. Z tohoto důvodu je potřeba ukládat některé hodnoty do trvalé paměti a při spuštění řídící jednotky tyto hodnoty opět načíst. V opačném případě by nastavené defaultní hodnoty zůstali nastavené pouze po dobu běhu programu. Zároveň by bylo nutné po

každém spuštění opakovaně provádět kalibraci jednotky. Pro trvalé uložení paměti tedy používám paměť EEPROM a pro práci s touto pamětí knihovnu EEPROM.h.

Pro uložení defaultních hodnot nám slouží funkce **saveSet**, která je zavolána při výběru volby Settings – Save z menu aplikace. Funkce využívá metodu **put** a **update** objektu *EEPROM*. Metodu **put** jsem použil pro uložení defaultních hodnot objemu (*volDeff*) a průotku (*spdDef*), které jsou datového typu float a metodu **update** pro defaultní hodnotu módu čerpání (*defMode*), která je datového typu char.

Pro uložení hodnoty char, jsem upřednostnil metodu **update** před metodou **write.** Metoda **update**, před zápisem navíc kontroluje, zda se zapisovaná hodnota shoduje s již zapsanou hodnotou. Zápis následně provede jen v případě, že se hodnoty liší. Tímto způsobem omezuje počet operací zápisu do paměti, který je u každé paměti limitován cca 100 000 zápisy. (Andrews, 2017)

Po uložení hodnot do paměti zároveň uložím defaultní hodnoty do proměnných, reprezentující nastavované hodnoty prostřednictvím menu. Po zapsání hodnot vypíšu potvrzující hlášku na displej a sériovou linku.

Zdrojový kód funkce **saveSet** uvádím níže**.** Parametry metod **put** a **update** jsou vždy adresy v paměti (definovaná makrem v rámci inicializace) a samotná hodnota.

```
void saveSet() {

  EEPROM.put(VOL_ADR,volDeff);
  EEPROM.put(SPD_ADR, spdDef);
  EEPROM.update(MOD_ADR, defMode);

  volf = volDeff;
  spd = spdDef;
  mode = defMode;
  lcd.setCursor(0,1);
  lcd.print("Saved");
  Serial.println("Saved");
}
```

Načtení uložených hodnot probíhá v rámci úvodní inicializace (viz 4.2.2.1 Úvodní inicializace) zavoláním funkce **readSet**. Funkce využívá metodu **get** a **read** objektu *EEPROM*. Metodu **get** jsem použil pro načtení defaultní hodnoty objemu (*volDeff*) a průtoku (spdDef), které jsou datového typu float a metodu **read** pro defaultní hodnotu módu čerpání (defMode), která je datového typu char.

Po načtení hodnot následují kontrolní podmínky, které zajišťují, aby v případě, že do EEPROM paměti dosud nebyla zapsána žádná data hodnoty nepřekročily povolené meze.

Zdrojový kód funkce **readSet** uvádím níže**.** Parametry metod **get** a **read** jsou vždy adresa v paměti (definovaná makrem v rámci inicializace). U metody **get** je navíc druhým parametrem proměnná, do které má být načtená hodnota uložena. Návratovou hodnotou metody **read** je načtená hodnota z paměti.

```
void readSet() {

  EEPROM.get(VOL_ADR, volDeff);
  EEPROM.get(SPD_ADR,spdDef);
  defMode = EEPROM.read(MOD_ADR);
  if (defMode != 2 && defMode != 1) {
    defMode = 1;
    EEPROM.update(MOD_ADR, defMode);
  }
  if (spdDef > topSpeed) {
    spdDef = topSpeed;
  }
  if (spdDef < minSpeed){
    spdDef = minSpeed;
    }
  if (volDeff > 500.0)
  {
    volDeff = 500.0;
  }
}
```

4.2.2.6 Funkce ovládání pomocí počítač

Mezi definovanými funkčními požadavky je možnost ovládat funkce čerpadla pomocí příkazů zasílaných pomocí sériové linky z připojeného PC.

Realizaci tohoto požadavku provádějí dvě funkce. První je funkce **serialRemote**, která je volána při výběru volby PC Control – Connect z menu aplikace. Jejím hlavním úkolem je příjem řetězce znaků zakončených oddělovačem řádků (Enterem). Tento řetězech pak následně předá funkci **doSerialCommand,** která vyhodnotí, zda se jedná o validní příkaz a zajistí jeho zpracování.

Při vytváření funkcí jsem se inspiroval řešením popsaným na Root.cz. (Malý, 2011)

4.2.2.6.1 Funkce serialRemote

Řídící jednotka komunikuje s PC pomocí rychlosti 9600 bit/s. Při zavolání funkce tedy nejprve dojde k otevření spojení na této rychlosti pomocí metody **begin** objektu *Serial*. Na začátku funkce dojde také k deklaraci potřebných proměnných:

- char lineCmd [60] – pro průběžné ukládání načteného řetězce o max. délce 60 znaků
- int i – pro uložení počtu načtených znaků
- char c – pro uložení načteného znaku
- int res – pro výsledek funkce **doSerialCommand**

Jádrem funkce je while cyklus, který má v rozhodující podmínce hodnotu true. Tato podmínka je tedy vždy splněna. Jediná možnost ukončení while cyklu je příkaz **break**.

V cyklu opakovaně dochází pomocí metody **read** objektu *Serial* k čtení znaku na sériové lince. Ihned po přečtení znaku je znak opět poslán zpět na sériovou linku. Díky tomu uživatel uvidí průběžně všechny jím zapsané znaky.

Načtený znak je následně dále zpracován pomocí podmínek:

- Pokud se jedná o znak konce řádku (Enter):

    - Zavolá se funkce doSerialCommand jako jediný parametr se použije načtený řetězec. Návratová hodnota funkce se uloží do proměnné *res.*
    - Hodnota *res* se vyhodnotí a v případě vrácení chybových kódů se vypíše příslušná hláška:
        - „Invalid syntax“ pro -2
        - „Invalid command“ pro -1
        - „Good bye“ pro 1
    - V případě že hodnota *res* se rovná 1, dojde k ukončení cyklu – jedná se o příkaz exit, který ukončí spojení.
- Pokud se jedná o znak Backspace:
    - Dojde ke snížení hodnoty *i* a smazání posledního znaku z řetězce *lineCmd*
- V ostatních případech:
    - Dojde k zapsání znaku *c* na konec řetězce a navýšení hodnoty *i.*

Ukázka zdrojového kódu funkce **serialRemote**:

```
void serialRemote() {
  Serial.begin(9600);
  Serial.println ("Pump driver 3.0");
  char lineCmd [60];
  int i=0;
  char c;
  int res;
  while(true){
   if(Serial.available()){
    c = Serial.read();
    Serial.print(c);
    if(c==10 || c==13 || c==64){//newline
      Serial.print("\n");
        if(i>60){res = -2;}
        else{
          res = doSerialCommand(lineCmd);
        }
        switch(res){
          case -2:
             Serial.println("Invalid syntax");
             break;
          case -1:
             Serial.println("Invalid command");
              break;
          case 1:
             Serial.println("Good bye!");
             break;
        }
      lineCmd[0]='\0'; i=0;

      if(res==1){
        break;
      }
    }
    else if(c==8 || c==127){
      i--;
      if(i<0)
        i=0;
      lineCmd[i]='\0';

      }
    else{
      lineCmd[i]=c;
      lineCmd[i+1]='\0';
      i++;
    }
   }
  }
  Serial.end();
  }
```

4.2.2.6.2 Funkce doSerialCommand

Jak jsem zmínil výše funkce **doSerialCommand** slouží ke zpracování načteného řetězce z příkazové řádky a v případě potřeby zavolání odpovídajících funkcí.

Funkce má definovaný návratový datový typ integer a definuje celkem čtyři typy návratových hodnot:

- -2: Invalid syntax – pokud byl detekován příkaz se špatnou syntaxí
- -1: Invalid command – nebyl detekován žádný z příkazů
- 0: Command execude – bezchybné zpracování příkazu
- 1: Exit – bezchybné zpracování příkazu Exit pro ukončení spojení

Funkce provádí zpracování příkazů postupně dle následujících bodů:

- Detekce příkazu
- Získání parametrů příkazu
- Provedení příkazu

Detekce příkazu

Pomocí funkce **strchr** je dohledán výskyt první mezery v řetězci.

V případě, že řetězec neobsahuje mezeru, porovná se pomocí funkce **strncmp,** zda se obsah řetězce rovná jednomu z následujících příkazů. V případě shody se provede zpracování odpovídajícího příkazu. Pokud se řetězec nerovná jedné z následujících možností, vrátí funkce hodnotu -1.

- „exit“ – funkce vrátí hodnotu 1: Exit
- „info“ – funkce vypíše na příkazovou řádku seznam dostupných příkazů a vrátí hodnotu 0

Pokud řetězec obsahuje mezeru, porovná se pomocí **strncmp** část řetězce před mezerou. V případě shody z jedním z následujících příkazů se pokračuje v dalším zpracování dle příslušného příkazu. V opačném případě, vrátí funkce hodnotu -1.

- „pump“
- „calibrate“
- „settings“

Získání parametrů příkazu

Získání parametrů příkazů probíhá pouze u trojice příkazů: *pump, calibrate* a *settings*. Syntaxe jednotlivých příkazů byla zmíněna již v kapitole 4.1.1.3.5. Pro připomenutí ji uvedu i v této kapitole:

- pump -c SPD
- pump -s SPD VOL
- calibrate VOL
- settings SPD VOL MOD

Získání parametrů příkazu Pump začíná vyhledáním znaku „-“ pomocí **strchr** v druhé části řetězce (za první nalezenou mezerou). V případě neúspěchu funkce vrátí -2. V opačném případě zkontroluje, zda se za pomlčkou nachází znaky „s“ případně „S“ nebo „c“ případně „C“. V případě že ne opět vrátí funkce hodnotu -2. V případě shody následuje další zpracování dle nalezeného znaku.

Zpracování je pro obě varianty velmi podobné. Pomocí funkce **strchr** nalezne v části řetězce za pomlčkou postupně všechny mezery. V případě neúspěchu funkce vždy vrátí hodnotu -2. Pomocí mezer rozdělí řetězec na jednotlivé části odpovídající parametrům.

K převodu parametrů z datového typu string na desetinné číslo použije program funkci **atof**. V případě neúspěšné konverze opět vrací funkce hodnotu -2.

Zisk parametrů pro funkce *calibrate* a *settings* je analogický se získáním parametrů pro příkaz pump. Jelikož však funkce *calibrate* a *settings* neobsahují přepínač, je tato část vynechána a dochází rovnou k získání parametrů

Provedení příkazu

Provedení příkazů *info* a *exit* jsem již popsal v části detekce příkazu. Vykonání ostatních příkazů *pump, calibrate* a *settings* probíhá zavoláním odpovídající funkce se získanými parametry.

- Pump – zavolání funkce **cPump** nebo **sPump** dle přepínače
- Calibrate – zavolání funkce **runCalibration**
- Settings – dojde k úpravě hodnot *volDeff, spdDef* a *defMode* a zavolání funkce **saveSet.**

4.2.2.7 Okamžité ukončení čerpání

Vzhledem k tomu, že čerpání (otáčení motoru) je prováděno v cyklu, může k ověření, zda uživatel stiskl tlačítko dojít vždy na konci každé smyčky. V každé smyčce je však prováděné několik kroků motoru s nastavenou rychlostí. Program by tudíž v případě delšího trvání smyčky nemusel zaznamenat stisknutí tlačítka. Z tohoto důvodu pro zachycení stisknutí tlačítka pro zastavení čerpání používám HW přerušení. HW přerušení funguje tak, že v případě stisknutí tlačítka procesor přeruší zpracování instrukcí hlavního programu a načte instrukce pro přerušení. To se nazývá obsluha přerušení. (Slinták, 2011)

Pro implementaci přerušení je nutné provést povolení přerušení pomocí funkce **interrupts** a registrovat funkci pro obsluhu přerušení pomocí funkce **attachInterrupt.**

```
attachInterrupt(PIN, INT_FUNC, MODE);
```

- PIN – číslo vstupního pinu na které bude přerušení reagovat
- INT_FUNC – funkce, která bude zavolána při detekci přerušení
- MODE – dle zvolené hodnoty definuje kdy bude přerušení detekováno:
    - LOW – pokud hodnota vstupního pinu bude logická 0
    - CHANGE – pokud se hodnota změní
    - RISING – pokud se hodnota mění z logické 0 na logickou 1
    - FALLING – pokud se hodnota mění logické 1 na logickou 0

Pomocí funkce **detachInterrupt** je možné naopak odpojit funkci od přerušení.

```
detachInterrupt(PIN);
```

Obsluha přerušení je v mém případě realizována jednoduchou funkcí **stopPumping,** která nastaví hodnotu proměnné *stopPump* na true a do proměnné *endTime* uloží pomocí

funkce **millis** aktuální hodnotu doby běhu řídící jednotky, která se využívá při kalibraci. Díky nastavení hodnoty *stopPump* na true dojde k vyhodnocení podmínky cyklu při čerpání nebo kalibraci a cyklus se přeruší a čerpadlo ukončí čerpání. Zdrojový kód obsluhy přerušení je uveden níže:

```
void stopPumping () {
  stopPump = true;
  endTime = millis ();
}
```

4.3 Testování řešení

V této kapitole vydefinuji testovací scénáře sloužící k ověření funkčnosti řešení a popíšu průběh testů a výsledek testování. Na závěr vyhodnotím funkčnost řešení na základě výsledku testů.

4.3.1 Rozsah testování, definice testovacích případů

Rozsah testovacích případů do značné míry vychází zejména z definovaných Use casů, které doplním o některé negativní scénáře. Pro každý testovací případ uvádím podmínky nutné ke spuštění případu a scénář testovacího případu – seznam kroků a očekávaných výsledků daného testovacího případu.

4.3.1.1 TC01 Spuštění přesného čerpání pomocí menu

ID Název	**TC01 Spuštění přesného čerpání pomocí menu**	
Podmínky	Řídící jednotka je spuštěna, je zobrazena úvodní položka menu. Proběhla úspěšná kalibrace. Byly nastaveny výchozí hodnoty parametrů: Mode Objem Průtok	
Scénář TC:	Krok scénáře:	Očekávaný výsledek
1.	Uživatel v menu vybere volbu START a potvrdí tlačítkem	Systém zobrazí podmenu pro spuštění čerpání. Na prvním místě je zobrazena položka MODE, zobrazena je nastavená výchozí hodnota.
2.	Uživatel pomocí tlačítek nastaví režim na „Specific“ a přepne menu na položku Volume.	Systém zobrazí hodnotu Volume s přednastavenou výchozí hodnotou.
3.	Uživatel změní nastavení Volume	Hodnotu Volume je možné upravit

	na požadovanou hodnotu a přejde na položku Speed	v rozmezí 1–500 ml. Systém zobrazí hodnotu Speed s přednastavenou výchozí hodnotou.
4.	Uživatel změní hodnotu Speed na požadovanou hodnotu a přejde na položku Run, kterou potvrdí.	Hodnotu Speed je možné upravit v rozmezí definovaném kalibrací. Systém spustí čerpání. Systém průběžně vypisuje načerpanou hodnotu na displej. Rychlost čerpání odpovídá zvolené rychlosti (závisí na přesnosti kalibrace).
5.	Uživatel čeká do načerpání hodnoty	Po načerpání cílové hodnoty systém zastaví čerpání. Systém vypíše hlášky „Done!“ a „Run again“ na displej.

4.3.1.2 TC02 Spuštění průběžného čerpání pomocí menu

ID Název	**TC02 Spuštění průběžného čerpání pomocí menu**	
Podmínky	Řídící jednotka je spuštěna, je zobrazena úvodní položka menu. Proběhla úspěšná kalibrace. Byly nastaveny výchozí hodnoty parametrů: Mode Průtok	
Scénář TC:	Krok scénáře:	Očekávaný výsledek
1.	Uživatel v menu vybere volbu START a potvrdí tlačítkem	Systém zobrazí podmenu pro spuštění čerpání. Na prvním místě je zobrazena položka MODE, zobrazena je nastavená výchozí hodnota.
2.	Uživatel pomocí tlačítek nastaví režim na „Continous“ a přepne menu na položku Speed.	Systém zobrazí hodnotu Speed s přednastavenou výchozí hodnotou.
3.	Uživatel změní hodnotu Speed na požadovanou hodnotu a přejde na položku Run, kterou potvrdí.	Hodnotu Speed je možné upravit v rozmezí definovaném kalibrací. Systém spustí čerpání. Systém průběžně vypisuje načerpanou hodnotu na displej. Rychlost čerpání odpovídá zvolené rychlosti

		(závisí na přesnosti kalibrace).
5.	Uživatel stiskne potvrzovací tlačítko.	Systém zastaví čerpání. Systém vypíše hlášky „Done!“ a „Run again“ na displej.

4.3.1.3 TC03 Předčasné zastavení přesného čerpání

ID Název	**TC03 Předčasné zastavení přesného čerpání**	
Podmínky	Probíhá přesné čerpání a nebylo dosaženo nastavené hodnoty (TC02) Čerpání bylo spuštěno z menu aplikace.	
Scénář TC:	Krok scénáře:	Očekávaný výsledek
1.	Uživatel před načerpáním cílové hodnoty stiskne potvrzovací tlačítko.	Systém okamžitě zastaví čerpání ještě před dosažením cílové hodnoty. Systém vypíše hlášky „Done!“ a „Run again“ na displej.

4.3.1.4 TC04 Nastavení výchozích hodnot

ID Název	**TC04 Nastavení výchozích hodnot**	
Podmínky	Řídící jednotka je spuštěna, je zobrazena úvodní položka menu.	
Scénář TC:	Krok scénáře:	Očekávaný výsledek
1.	Uživatel v menu vybere volbu Settings a potvrdí tlačítkem	Systém zobrazí podmenu pro nastavení hodnot. Na prvním místě je zobrazena položka Def.mode, zobrazena je v minulosti nastavená výchozí hodnota.
2.	Uživatel pomocí tlačítek nastaví jeden režimů a přepne na položku Def.vol.	Je možné nastavit režim „Continous“ nebo „Specific“. Systém zobrazí hodnotu Def.vol

		s dříve nastavenou výchozí hodnotou.
3.	Uživatel změní nastavení Def.vol na požadovanou hodnotu a přejde na položku Def.spd	Hodnotu Def.vol je možné upravit v rozmezí 1–500 ml. Systém zobrazí hodnotu Def.spd s dříve nastavenou výchozí hodnotou.
4.	Uživatel změní hodnotu Def.spd na požadovanou hodnotu a přejde na položku Save, kterou potvrdí.	Hodnotu Def.spd je možné upravit v rozmezí definovaném kalibrací, nebo v rozmezí 1–10 pokud kalibrace úspěšně neproběhla. Systém uloží hodnoty a vypíše hlášku Saved.
5.	Uživatel stiskne tlačítko zpět a přejde do měnu Start, kde zkontroluje, že se zobrazují nově nastavené výchozí hodnoty.	Nastavené hodnoty se zobrazují.
6.	Uživatel vypne napájení jednotky.	Jednotka se vypne.
7.	Uživatel zapne napájení jednotky. Po zapnutí jednotky zkontroluje, že nastavené hodnoty se nachází v menu Start i v menu Settings.	Hodnoty jsou nastaveny i po restartu jednotky.

4.3.1.5 TC05 Spuštění kalibrace z menu

ID Název	TC01 Spuštění přesného čerpání pomocí menu	
Podmínky	Řídící jednotka je spuštěna, je zobrazena úvodní položka menu.	
Scénář TC:	Krok scénáře:	Očekávaný výsledek
1.	Uživatel v menu vybere volbu Calibrate a potvrdí tlačítkem	Systém zobrazí podmenu pro spuštění Kalibrace. Na prvním místě je zobrazena položka Volume, zobrazena je nastavená výchozí hodnota.
2.	Uživatel změní nastavení Volume	Hodnotu Volume je možné upravit

	na požadovanou kalibrační hodnotu a přejde na položku Run.	v rozmezí 1–500 ml. Systém zobrazí hlášku „Step 1 - low speed“ a volbu ">Start calib. step."
3.	Uživatel stiskne potvrzovací tlačítko.	Čerpadlo začne čerpat pomalou rychlostí. Na displeji průběžně vypisuje čas čerpání a počet kroků.
4.	Uživatel po načerpání cílové hodnoty stiskne potvrzovací tlačítko.	Čerpadlo ukončí čerpání a zobrazí hlášku „"Step 2 - high speed" a volbu ">Start calib. step."
5.	Uživatel stiskne potvrzovací tlačítko.	Čerpadlo začne čerpat rychlou rychlostí. Na displeji průběžně vypisuje čas čerpání a počet kroků.
6.	Uživatel po načerpání cílové hodnoty stiskne potvrzovací tlačítko. (Čas druhého čerpání je menší než prvního.)	Systém zobrazí hlášku „Calibration end“ na displeji a vypíše odpočet 5 s a poté restartuje jednotku.
7.	Uživatel ověří, zda se nastavilo nové rozmezí hodnoty Speed v submenu Start.	Nové rozmezí dle kalibrace bylo nastaveno.
8.	Uživatel opakuje kalibraci, ale kalibraci pokazí tak, že čas trvání prvního kroku je menší než druhého kroku.	Systém vypíše hlášku „Calibration failed“ na displej a vypíše odpočet 5 s a poté restartuje jednotku.
9.	Uživatel ověří, zda nastavení rozmezí hodnoty Speed v submenu Start.	Rozmezí hodnoty v nastaveni je ve výchozím rozsahu 1–10.

4.3.1.6 TC06 Spuštění čerpání z příkazové řádky

ID Název	TC06 Spuštění čerpání z příkazové řádky	
Podmínky	Řídící jednotka je spuštěna, je zobrazena úvodní položka menu. Proběhla úspěšná kalibrace.	
Scénář TC:	Krok scénáře:	Očekávaný výsledek

1.	Uživatel na PC otevře terminál (např.: Putty) a otevře spojení rychlostí 9600 baud/s a vybere port na kterém je připojeno zařízení.	Připojení proběhlo úspěšně je zobrazen prázdná terminál.
2.	Uživatel v menu aplikace vybere volbu PC Control a potvrdí tlačítkem.	Systém vypíše do terminálu „Pump driver“.
3.	Uživatel zadá příkaz „pump -c SPD a stiskne Enter. SPD nahradí libovolným číslem z rozsahu rychlostí (lze zjistit například v rámci TC02).	Systém spustí čerpání. Systém průběžně vypisuje načerpanou hodnotu do terminálu. Rychlost čerpání odpovídá zvolené rychlosti (závisí na přesnosti kalibrace).
4.	Uživatel stiskne Enter na klávesnici nebo potvrzovací tlačítko jednotky.	Systém zastaví čerpání. Vypíše hlášku „Done!“.
5.	Uživatel zadá příkaz „pump -s VOL SPD a stiskne Enter. SPD nahradí libovolným číslem z rozsahu rychlostí (lze zjistit například v rámci TC02). VOL nahradí libovolným číslem v rozsahu 1–500. Může použít i desetiny.	Systém spustí čerpání. Systém průběžně vypisuje načerpanou hodnotu do terminálu. Rychlost čerpání odpovídá zvolené rychlosti (závisí na přesnosti kalibrace).
6.	Uživatel počká do načerpání cílové hodnoty.	Systém po načerpání daného objemu ukončí čerpání a zobrazí hlášku „Done!“.
7.	Uživatel opakuje krok 5.	Systém spustí čerpání. Systém průběžně vypisuje načerpanou hodnotu do terminálu. Rychlost čerpání odpovídá zvolené rychlosti (závisí na přesnosti kalibrace).
8.	Uživatel stiskne Enter na klávesnici nebo potvrzovací tlačítko jednotky.	Systém zastaví čerpání před načerpáním cílové hodnoty. Vypíše hlášku „Done!“.
9.	Uživatel změní hodnotu Speed na požadovanou hodnotu a přejde na položku Run, kterou potvrdí.	Hodnotu Speed je možné upravit v rozmezí definovaném kalibrací. Systém spustí čerpání. Systém průběžně vypisuje načerpanou hodnotu na displej. Rychlost čerpání odpovídá zvolené rychlosti

		(závisí na přesnosti kalibrace).
10.	Uživatel stiskne potvrzovací tlačítko.	Systém zastaví čerpání. Systém vypíše hlášky „Done!“ a „Run again“ na displej.

4.3.1.7 TC07 Spuštění kalibrace z příkazové řádky

ID Název	**TC07 Spuštění kalibrace z příkazové řádky**	
Podmínky	Řídící jednotka je propojená s PC. Na PC je spuštěn terminál (např. Putty). Je možné zadávat příkazy	
Scénář TC:	Krok scénáře:	Očekávaný výsledek
1.	Uživatel zadá příkaz „calibrate VOL“ a stiskne Enter. Místo VOL zadá požadovaný kalibrační objem z rozsahu 1–500. Může použít i desetiny	Systém zobrazí na terminálu hlášku „Step 1 - low speed“ a volbu "Press Enter …"
2.	Uživatel stiskne Enter.	Čerpadlo začne čerpat pomalou rychlostí. V terminálu se průběžně vypisuje čas čerpání a počet kroků.
3.	Uživatel po načerpání cílové hodnoty stiskne Enter	Čerpadlo ukončí čerpání a zobrazí v terminálu hlášku „"Step 2 - high speed" a volbu "Press Enter …"
4.	Uživatel stiskne potvrzovací tlačítko.	Čerpadlo začne čerpat rychlou rychlostí. V terminálu průběžně vypisuje čas čerpání a počet kroků.
5.	Uživatel po načerpání cílové hodnoty stiskne Enter. (Čas druhého čerpání je menší než prvního.)	Systém zobrazí hlášku „Calibration end“ v terminálu a vypíše odpočet 5 s a poté restartuje jednotku.
6.	Uživatel ověří, zda se nastavilo nové rozmezí hodnoty Speed v submenu Start.	Nové rozmezí dle kalibrace bylo nastaveno.
7.	Uživatel vybere v menu na	Systém do terminálu vypíše

	jednotce volbu PC Connect – Connect a potvrdí.	„Pump driver“
8.	Uživatel opakuje kalibraci, ale kalibraci pokazí tak, že čas trvání prvního kroku je menší než druhého kroku.	Systém vypíše hlášku „Calibration failed“ v terminálu a vypíše odpočet 5 s a poté restartuje jednotku.
9.	Uživatel ověří nastavení rozmezí hodnoty Speed v submenu Start.	Rozmezí hodnoty v nastaveni je ve výchozím rozsahu 1–10.

4.3.1.8 TC08 Nastavení hodnot pomocí příkazové řádky

ID Název	**TC09 Ostatní funkce příkazové řádky**	
Podmínky	Řídící jednotka je propojená s PC. Na PC je spuštěn terminál (např. Putty). Je možné zadávat příkazy	
Scénář TC:	Krok scénáře:	Očekávaný výsledek
1.	Uživatel zadá příkaz „settings VOL SPD MODE“ a stiskne Enter. VOL nahradí požadovanou hodnotou v rozsahu 1-500. SPD nahradí hodnotou v rozsahu dle kalibrace. MODE nahradí hodnotou 1 nebo 2.	Systém vypíše hlášku Saved do terminálu.
2.	Uživatel zadá v terminálu příkaz „exit“ a stiskne Enter.	Systém zobrazí v terminálu hlášku „Good bye!“ a ukončí spojení
2.	Uživatel stiskne tlačítko zpět a přejde do měnu Start, kde zkontroluje, že se zobrazují nově nastavené výchozí hodnoty.	Nastavené hodnoty se zobrazují.
3.	Uživatel vypne napájení jednotky.	Jednotka se vypne.
4.	Uživatel zapne napájení jednotky. Po zapnutí jednotky zkontroluje, že nastavené hodnoty se nachází v menu Start i v menu Settings.	Hodnoty jsou nastaveny i po restartu jednotky.

4.3.1.9 TC09 Ostatní funkce příkazové řádky

ID Název	**TC09 Ostatní funkce příkazové řádky**	
Podmínky	Řídící jednotka je propojená s PC. Na PC je spuštěn terminál (např. Putty). Je možné zadávat příkazy	
Scénář TC:	Krok scénáře:	Očekávaný výsledek
1.	Uživatel zadá v terminálu příkaz „info“ a stiskne Enter.	Systém zobrazí v terminálu seznam dostupných příkazů.
2.	Uživatel zadá v terminálu příkaz „exit“ a stiskne Enter.	Systém zobrazí v terminálu hlášku „Good bye!“ a ukončí spojení
3.	Uživatel zkusí zapsat příkaz.	Napsané znaky se v terminálu nezobrazují. Terminál nekomunikuje.

4.3.1.10 TC10 Neexistující příkazy – negativní scénář příkazové řádky

ID Název	**TC09 Ostatní funkce příkazové řádky**	
Podmínky	Řídící jednotka je propojená s PC. Na PC je spuštěn terminál (např. Putty). Je možné zadávat příkazy	
Scénář TC:	Krok scénáře:	Očekávaný výsledek
1.	Uživatel zadá v terminálu libovolné slovo, které není příkaz a stiskne Enter	Systém zobrazí v terminálu hlášku „Invalid command“
2.	Uživatel zadá v terminálu příkaz například „pump -x “	Systém zobrazí v terminálu hlášku „Invalid syntax“
3.	Uživatel zadá v terminálu příkaz například „pump -c -1 “	Systém zobrazí v terminálu hlášku „Invalid parameters“
4.	Uživatel zadá příkaz „settings“ pouze s jedním nebo dvěma parametry například „settings 2.0 5.0“	Systém zobrazí v terminálu hlášku „Invalid parameters“

5.	Uživatel zadá příkaz „calibrate -20.0“	Systém zobrazí v terminálu hlášku „Invalid parameters“

4.3.2 Průběh a výsledky testování

Průběh testování rozdělím do dvou částí. První otestování, jsem prováděl vždy po dokončení určité funkcionality a sloužilo hlavně k základnímu ověření funkčnosti. V první fázi testování jsem ještě nevyužíval testovací případy, ale podařilo se mi odhalit většinu základních chyb, které by spuštění testovacích scénářů ve většině případů neumožňovali.

K druhé fázi testování podle testovacích případů jsem přistoupil až po dokončení všech funkcí daného řešení. Pomocí testovacích případů se mi podařilo odhalit chyby uvedené v následující tabulce.

Tabulka 12: Seznam odhalených chyb

Popis chyby	Způsob vyřešení
Nestabilita řešení při komunikaci se sériovou linkou.	Nahradil jsem využití digitálních pinů (používaných pro signály RX a TX) analogovými piny nastavenými pro digitální vstup a výstup.
Výsledky kalibrace nebyly validovány.	Doplnění validace výsledků kalibrace, prostřednictvím porovnání hodnot maximálního a minimálního průtoku
Nemožnost ukončit čerpání spuštěné ze sériové linky tlačítkem.	Doplnění definice přerušení při spuštění čerpání z příkazové řádky.

4.3.3 Vyhodnocení funkčnosti řešení

Výsledné řešení je funkční, splňuje funkční a nefunkční požadavky popsané v kapitole 4.1.1 Analýza požadavků a návrh cílového řešení.

V rámci testování byly spuštěny všechny scénáře testovacích případů. Veškeré odhalené chyby popsané v přechozí kapitole byly odstraněny a popis funkcionalit popsaných v předešlých kapitolách reflektuje změny provedené v důsledku opravy nalezených chyb.

4.4 Cenová kalkulace navrhovaného řešení

V rámci následujících tabulek uvádím cenovou kalkulaci implementovaného řešení. Uvádím dvě varianty. Jedná počítá s deskou Arduino UNO z originální distribuce. Druhá počítá s využitím kompatibilního klonu.

Tabulka 13: Cenová kalkulace s využitím originálního Arduino UNO

Název komponenty	**Cena v Kč (včetně DPH)**	**Zdroj ceny**
Arduino UNO	610–660	www.alza.cz, www.hwkitchen.cz
H-Můstek	29–42	www.gme.cz
LCD displej 16x2	133	www.gme.cz
Tlačítka a kabeláž	100	odhadované náklady
DC Adaptér 12V, 5A	221	https://www.mojerc.cz/
Suma	1093–1156	

Zdroj: vlastní zpracování

Tabulka 14: Cenová kalkulace s využitím kompatibilního klonu

Název komponenty	**Cena v Kč (včetně DPH)**	**Zdroj ceny (březen 2020)**
Klon Arduino UNO	150-198	www.alza.cz, www.hwkitchen.cz
H-Můstek	29–42	www.gme.cz
LCD displej 16x2	133	www.gme.cz
Tlačítka a kabeláž	100	odhadované náklady
DC Adaptér 12V, 5A	221	https://www.mojerc.cz/
Suma	633–694	

Zdroj: vlastní zpracování

Do kalkulace jsem nezahrnul náklady na opláštění řešení a náklady na samotné čerpadlo. Zle předpokládat, že tyto náklady budou shodné pro současné řešení popsané v kapitole 3.1 i pro nově implementované řešení.

Cílové náklady jsou výrazně nižší, než náklady na současné řešení uvedené v kapitole 3.1.3. a to i v případě použití dražší originální varianty desky Arduino UNO. Náklady na současné řešení bez boxu jsou 4750, - Kč.

5 Zhodnocení výsledků a doporučení

Cílem práce bylo vytvořit program pro vývojovou desku Arduino sloužící k ovládání dvou pístového čerpadla řízeného krokovým motorem, sestavení prototypu řídící jednotky a sestavení cenové kalkulace implementovaného řešení.

Cílové řešení mělo implementovat níže vypsané funkcionality, které měli být podrobněji popsány v analytické části práce.

Požadované funkcionality:

- Základní uživatelské rozhraní
  - Zobrazení na LCD displeji
  - Ovládání pomocí tlačítek nebo klávesnice
- Ovládání pomocí připojení k PC
- Umožnit spustit čerpání ve dvou režimech s nastavením parametrů průtoku a objemu
  - Průběžné čerpání
  - Přesné čerpání
- Možnost uložení výchozích hodnot parametrů do paměti zařízení
- Možnost kalibrace čerpadla

V následujících podkapitolách nejprve popíšu výsledný stav řešení a zhodnotím naplnění cílů následně navrhnu další možnosti rozvoje řešení.

5.1 Popis výsledného stavu

Výsledky mé práce bych rozdělil do několika částí. Ještě před zahájením zpracování vlastního řešení jsem popsal současný stav problematiky. Také jsem provedl literární rešerši. V rešerši jsem stručně popsal způsoby řízení krokových motorů, různé druhy zobrazovacích zařízení a různé typy Arduino desek.

První výstupem vlastního řešení práce je Analýza funkčních požadavků. V rámci analýzy jsem podrobněji rozpracoval popis funkcionalit popisovaných v kapitole 2.1 Cíl práce. Zároveň jsem vypracoval seznam případů užití a popsal jednotlivé scénáře.

Druhým navazujícím výstupem je program pro vývojovou desku Arduino UNO. Výsledný program s využitím definované sady funkcí, popsané v kapitole 4.2.2, implementuje všechny požadavky a případy užití popsané v analytické části práce. Zdrojový kód programu je přílohou této diplomové práce - Příloha 2.

Průběžně s vytvářením zmíněného programu jsem také pomocí nepájivého pole a jednotlivých komponent vytvořil funkční prototyp řídící jednotky, jehož schéma a popis je uveden v kapitole 4.2.1. Výsledný prototyp je nedílným výsledkem práce. Fotografie sestaveného prototypu je uvedena v přílohách práce - Příloha 1. Po dokončení implementace řešení jsem vytvořil sadu testovacích případů a podle nich provedl otestování celého řešení. Průběh a výsledky testování včetně seznamu testovacích případů jsou uvedeny v kapitole 4.3.

Posledním výstupem práce je cenová kalkulace implementovaného řešení. Cílové náklady se pohybují mezi **1093,-** a **1156,- Kč** případě použití originální verze Arduino

UNO nebo mezi **633,-** a **694,- Kč** v případě použití kompatibilního klonu. V obou případech je hodnota výrazně nižší než náklady současného řešení, které jsou **4750**, - Kč

Všechny definované díle práce byly splněny.

5.2 Další možnosti rozvoje aplikace

V následujících podkapitolách stručně popíšu další návrhy na rozvoj implementovaného řešení, které by bylo možné realizovat.

5.2.1 Využití odlišného zobrazovacího zařízení

Současné řešení využívá Alfanumerický LCD displej 16x2. V případě potřeby je možné jej s minimálními úpravami programu nahradit LCD displejem s odlišnými rozměry (například 20x4 apod.) Ve zdrojovém kódu je pouze nutné upravit definici výstupního elementu menu. V částech kódu, kde dochází k výpisu na displej je nutné dle potřeby upravit souřadnice na které jsou vypisovány hlášky.

Nahrazení alfanumerického displeje za jiný typ zařízení je také možné, ale úpravy budou pravděpodobně v závislosti na zvoleném typu zařízení o něco rozsáhlejší. Zvolená knihovna Arduino Menu však poskytuje podporu pro různé varianty zobrazovacích zařízení.

5.2.2 Doplnění senzoru hladiny pro zpřesněni kalibrace

Zajímavým, užitečným rozšířením by mohlo být doplnění řešení o senzor hladiny. Tento senzor hladiny by nahrazoval stisknutí tlačítka uživatelem při dosažení kalibračního objemu pro zastavení čerpání. Čerpání by bylo zastaveno okamžitě po dosažení požadované hladiny, čímž by došlo k zpřesnění kalibrace a ke snížení pravděpodobnosti chybné kalibrace.

Alternativní využití tohoto senzoru by mohlo být bezpečnostní vypnutí čerpání po naplnění nádoby.

5.2.3 Doplnění režimu čerpání

Současné řešení podporuje režimy průběžného čerpání a přesně čerpání předem známého objemu kapaliny.

Tyto režimy by bylo možné doplnit o režim čerpání neznámého objemu. Návrh předpokládá přítomnost hladinového senzoru popsaného v předchozí kapitole. Čerpání by probíhalo stejně jako u kalibrace až do dosažení hladinového senzoru. Režim by mohl sloužit k zjištění objemu neznámé nádoby. Načerpaný objem by byl po skončení čerpání zobrazen uživateli.

5.2.4 Rozšíření pro unipolární motory

Současné řešení je navrženo pro ovládání čerpadla pomocí bipolárního krokového motoru. Jednoduchou výměnou H-Můstku za obvod ULN2003 je však možné řešení upravit pro řízení čerpadla poháněného unipolárním krokovým motorem. (ULN200x, 1976) Samozřejmě je nutné také upravit parametry motoru ve zdrojovém kódu, pokud jsou odlišné od stávajících.

6 Závěr

V rámci práce jsem se zabýval tvorbou programu pro vývojovou desku Arduino sloužící k ovládání dvou pístového čerpadla řízeného bipolárním krokovým motorem, sestavením prototypu řídící jednotky a sestavením cenové kalkulace implementovaného řešení.

Před samotnou implementací řešení jsem se v rámci literární rešerše seznámil s možnostmi řízení krokových motorů, možnosti využití zobrazovacích zařízení včetně možností implementace menu a s jednotlivými typy Arduino desek.

Druhým krokem, který jsem provedl před zahájením implementace byla analýza funkčních a nefunkčních požadavků a definice případů užití. Na základě výsledků analýzy a literární rešerše jsem navrhl architekturu řešení. Základem této architektury je vývojová deska Arduino UNO. Krokový motor čerpadla je k desce připojen pomocí H-můstku. Uživatelské rozhraní zajišťuje alfanumerický LCD displej v kombinaci se třemi vstupními tlačítky.

Po dokončení analýzy jsem přistoupil k samotné implementaci řešení, která má dvě části. První části bylo sestavení prototypu dle navržené architektury pomocí nepájivého pole, a propojovacích vodičů. Druhou částí implementace bylo vytvoření programu pro vývojovou desku Arduino UNO. Program pro Arduino UNO se skládá ze sady funkcí, které implementují veškeré požadavky popsané v analytické části práce.

Po dokončení implementace jsem na základě definovaných případů užití vytvořil sadu testovacích případů, které jsem využil pro ověření funkčnosti navrženého systému.

Po úspěšném otestování řešení a odstranění nalezených chyb jsem sestavil cenovou kalkulaci. Výsledné náklady na implementované řešení jsou výrazně nižší než náklady na současně využívané řešení.

Na závěr práce jsem provedl shrnutí současného stavu, popsal jsem výstupy práce a zhodnotil jsem naplnění cílů. Výstupem práce jsou: analýza funkčních požadavků, program pro desku Arduino UNO, prototyp řídící jednotky a cenová kalkulace řešení. Výsledky práce splňují všechny definované cíle.

Na závěr práce jsem navrhl další možnosti rozvoje řešení: využití jiného typu displeje, doplněné senzoru detekce hladiny pro zpřesnění kalibrace, doplnění režimu čerpání ukončeného sepnutím senzoru hladiny nebo rozšíření řešení pro unipolární krokové motory.

7 Seznam použitých zdrojů

VODA, Zbyšek, 2017. *Průvodce světem Arduina*. Vydání druhé. Bučovice: Martin Stříž. ISBN 978-80-87106-93-8.

SYROVÝ, Jaroslav, 2014. Arduino – řízení krokového motoru. In: *Nul.cz* [online]. Praha, 2014-11-11 [cit. 2020-04-06]. Dostupné z: http://nul.cz/arduino/arduino-rizeni-krokoveho-motoru/

SLINTÁK, Vlastimil, 2011. Arduino a přerušení. *UART.cz* [online]. UART.cz, 2011-10-11 [cit. 2020-04-06]. Dostupné z: https://uart.cz/271/arduino-a-preruseni/

SCHMALZ, Brian, 2015. Easy Driver Stepper Motor Driver: An Open Source Hardware Stepper Motor Drive Project. In: *Schmalz Haus LLC* [online]. Schmalz Haus, 2015-08-18 [cit. 2020-02-01]. Dostupné z: http://www.schmalzhaus.com/EasyDriver/

SANTY.CZ, 2020. LCD displej 16x2 pro Arduino – modrý. SANTY.CZ. *Santy.cz* [online]. Braníškov: Santy.cz [cit. 2020-02-08]. Dostupné z: https://www.santy.cz/zobrazovace-c7/lcd-1602-blue-i8/

MAŇÁK, Petr, 2014. *Řízení krokového motoru pomocí platformy Arduino*. Praha: Fakulta elektrotechnická, ČVUT. Bakalářská práce. Fakulta elektrotechnická, ČVUT. Vedoucí práce Ing. Hlinovský Vít CSc.

MALÝ, Martin, 2011. Ovládání Arduina v reálném čase z počítače. *Root.cz* [online]. Root.cz, 2011-01-25 [cit. 2020-04-06]. Dostupné z: https://www.root.cz/clanky/ovladani-arduina-v-realnem-case-z-pocitace/

LEUCHTER, Jan a Huy Dong QUANG, 2018. Design of interfering mobile device in the band Wi-Fi with magnetron. *Advances in electrical and electronic engineering*. **16**(4), 489-500. DOI: 10.15598/aeee.v16i4.2438. ISSN 1804-3119.

LÁTAL, Martin, 2018. Krátce k problematice vendor lock-in. *Epravo.cz* [online]. Praha: epravo.cz, 2018-10-19 [cit. 2019-11-23]. Dostupné z: https://www.epravo.cz/top/clanky/kratce-k-problematice-vendor-lock-in-108256.html

KALCHEV, Vasil, 2019. *LiquidMenu* [online]. Varna, Bulgaria, 2019-12-07 [cit. 2020-04-06]. Dostupné z: https://github.com/VaSe7u/LiquidMenu

GASSNER, Sebastian a Noah SHIBLEY, 2019. Stepper Library. *Arduino* [online]. Arduino, 2019-12-24 [cit. 2020-02-01]. Dostupné z: https://www.arduino.cc/en/reference/stepper

COLLI, Marco, 2017. MD_Menu Library. *MajicDesigns* [online]. Sydney: MajicDesigns, 2017-01 [cit. 2020-02-02]. Dostupné z: https://majicdesigns.github.io/MD_Menu/

AZEVEDO, Rui, 2019. ArduinoMenu Library 3.x/4.x Wiki. *ArduinoMenu 4* [online]. Azores, Portugal, 2019-02-16 [cit. 2020-04-06]. Dostupné z: https://github.com/neu-rah/ArduinoMenu/wiki

ANDREWS, Christopher, 2017. EEPROM Library. *Arduino* [online]. 2017-09-05 [cit. 2020-04-06]. Dostupné z: https://www.arduino.cc/en/Reference/EEPROM

LCD displej, 2016. *Arduino návody* [online]. Praha: Arduino návody [cit. 2020-04-01]. Dostupné z: https://navody.arduino-shop.cz/zaciname-s-arduinem/lcd-displej.html

***ULN200x, ULQ200x High-Voltage, High-Current Darlington Tranzistor Arrays,* 1976.** Rev. 2019. Texas Instrument. SLRS027P. Dostupné také z: http://www.ti.com/lit/ds/symlink/uln2003a.pdf

WATREX, DeltaChrom™ Membrane Suppression Unit MSU 010. *Watrex* [online]. [cit. 2019-11-16]. Dostupné z: https://watrex.com/index.php/cs/produkty/hplc-pristroje/jin%C3%A9-lc-komponenty/dr99101502-36-detail?fbclid=IwAR2l-wYG45F40tyobiYV6xGLUxgAVszas_ZvnTGvTAO2eMniv__KkpmdQv4

WP1000 Peristaltic pump: Selection guide, Tokyo, Japan: Welco Co. Dostupné také z: https://www.welco.net/product/wp1000_1100/wp1000_1100_guide/wp1000_1100_guide03.html

H-můstek modul L9110S, 2016. ECLIPSERA s.r.o. Výrobní číslo: 1458419853. Dostupné také z: https://arduino-shop.cz/docs/produkty/0/68/1458419853.pdf

Arduino Store [online], Arduino [cit. 2020-04-06]. Dostupné z: https://store.arduino.cc/arduino-mini-05

Getting Started with the Arduino Mini, 2018. In: *Arduino* [online]. Arduino, 2018-02-10 [cit. 2020-04-06]. Dostupné z: https://www.arduino.cc/en/Guide/ArduinoMini

2.8'' TFT TOUCH SHIELD V2.0, 2020. In: *Hwkitchen.cz* [online]. [cit. 2020-04-06]. Dostupné z: https://www.hwkitchen.cz/2-8-tft-touch-shield-v2-0/?gclid=CjwKCAjwpqv0BRABEiwA-TySwbRhEKTuDuojKZ-yzs0oi-hQDkOYht38Lpy3lZwksOn9SU9J08WwJhoCqYIQAvD_BwE

U8g2, 2019. *GitHub* [online]. San Francisco, 2019-06-30 [cit. 2020-04-06]. Dostupné z: https://github.com/olikraus/u8g2

Grafický LCD display 128x64 ST7920, 2017. *Arduino návody* [online]. Praha: Arduino návody, 2017-04-25 [cit. 2020-02]. Dostupné z: https://navody.arduino-shop.cz/navody-k-produktum/graficky-lcd-display-128x64-st7920.html

16x2 LCD Module, 2017. *Components 101* [online]. Components 101, 2017-08-30 [cit. 2020-04-06]. Dostupné z: https://components101.com/16x2-lcd-pinout-datasheet

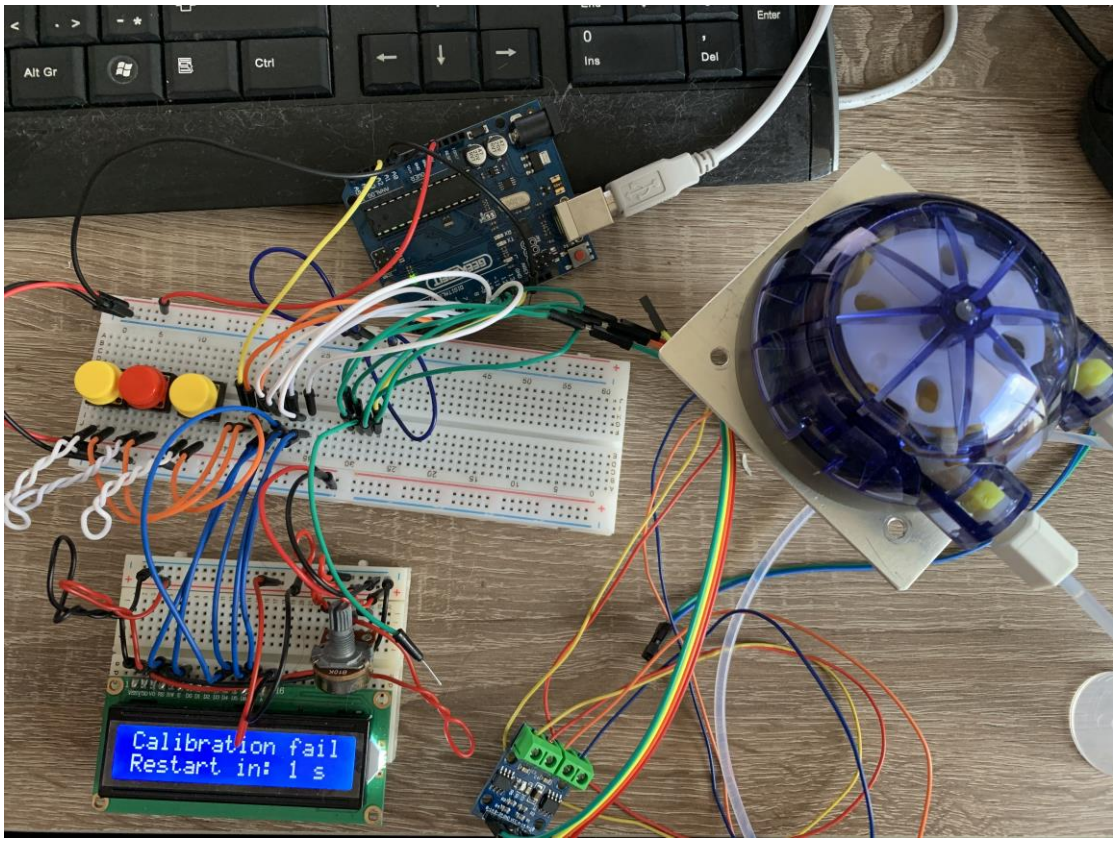
***Specification of LCD module,* 2008**. Xiamen: XIAMEN AMOTEC DISPLAY CO. ADM1602K-NSW-FBS.

Arduino Bipolar Stepper Motor Control, 2018. In: *Simple Projects* [online]. Simple Projects, 2018-10-16 [cit. 2020-01-02]. Dostupné z: https://simple-circuit.com/arduino-bipolar-stepper-motor-control/

Driver krokových motorů A3967 [online], 2019. In: 2019-05-28 [cit. 2020-02-01]. Dostupné z: https://navody.arduino-shop.cz/navody-k-produktum/driver-krokovych-motoru-a3967.html

8 Přílohy

Příloha 1: Fotografie sestaveného prototypu řešení



Příloha 2: Zdrojový kód aplikace

```
#include <Stepper.h>

#include <menu.h>
#include<menuIO/serialIO.h>
#include<menuIO/stringIn.h>
#include<menuIO/chainStream.h>
#include<menuIO/liquidCrystalOut.h>
#include <EEPROM.h>
#include<LiquidCrystal.h>
#include"decimalField.h"

using namespace Menu;

//Definice pinu desky
#define RESET A0
#define BACK_BTN A1
#define SEL_BTN 2
#define NAV_BTN 3

#define MAX_DEPTH 2
#define SOFT_DEBOUNCE_MS 100

#define INT_MAX 32767
//Adresy v pameti
#define STP_ADR 1
#define SPD_ADR 3
#define MOD_ADR 7
#define TOPSPD_ADR 9
#define MINSPD_ADR 13
#define INIT_ADR 17
#define VOL_ADR 21

const char * calEnd = "Calibration ended.";
const char * pumped = "Pumped: ";

//pripojeni cerpadla
#define MOT1 10
#define MOT2 11
#define MOT3 12
#define MOT4 13
//konfigurace motoru cerpadla
#define MOTSPDL 1
#define MOTSPDH 1280 // 20*64  20 = rpm čerpadla po zpřevodování motoru 1:64
#define MOTSTEPS 200 // uhel kroku čerpadla po zprevodování je 0,0282, 0,0282*64
= 1.8 uhel kroku motoru => 360/1,8 = 200 kroků
```

```
char INIT = EEPROM.read(INIT_ADR);
//Nacteni kalibracnich hodnot
float topSpeed = EEPROM.get(TOPSPD_ADR,topSpeed);
float minSpeed = EEPROM.get(MINSPD_ADR,minSpeed);
uint16_t stepsPerDeciMl = EEPROM.read(STP_ADR) + EEPROM.read(STP_ADR+1) * 256;

unsigned long endTime = 0;
//definice promenych pro menu
char mode, defMode = 1;
float spd = minSpeed, spdDef = minSpeed;
float volf = 1.0,volCalf = 1.0, volDeff = 1.0;

bool stopPump = false;

void stopPumping() { //funkce pro obsluhu preruseni
  stopPump = true;
  endTime = millis();
}

void initSets() { //Nastavení hodnot při prvním spuštění na konkrétní desce nebo
po smazání EEPROM paměti
 EEPROM.put(VOL_ADR,10.0);
 EEPROM.update(STP_ADR,1);
 EEPROM.update(STP_ADR+1,0);
 EEPROM.put(SPD_ADR, 1.0);
 EEPROM.update(MOD_ADR, 1.0);
 EEPROM.put(TOPSPD_ADR, 10.0);
 EEPROM.put(MINSPD_ADR, 1.0);
 EEPROM.update(INIT_ADR, 1);
 digitalWrite(RESET, LOW);
}



//Definice objektů pro LCD displej a motor
LiquidCrystal lcd(4, 5, 6, 7, 8, 9);
Stepper motor = Stepper(MOTSTEPS, MOT4, MOT3, MOT2, MOT1); //predefinovat s
opravym motorem

void cPump(float pumpSpeed, bool serial = false){
      float currentVolume = 0.0;
     uint16_t motorSpeed = map(pumpSpeed, minSpeed, topSpeed, MOTSPDL, MOTSPDH);
     motor.setSpeed(motorSpeed);
   do {
     motor.step(stepsPerDeciMl);
      currentVolume += (0.1);
      if(serial){
        Serial.print(pumped);
```

```
       Serial.print(currentVolume,2);Serial.print("ml");
        Serial.println();
         }
       else{
         lcd.clear();
         lcd.print(pumped);
         lcd.print(currentVolume,2);lcd.print("ml      ");
         }

      if(Serial.available()){
         char c;
         c = Serial.read();
         if(c==10 || c==13 || c==64){
           stopPump = true;
         }
       }
   }while (!stopPump);
}

void sPump(float pumpSpeed, float pumpVolume, bool serial = false){
  float currentVolume = 0.0;
 uint16_t motorSpeed = map((uint16_t)(pumpSpeed*10), (uint16_t)(minSpeed*10),
(uint16_t)(topSpeed*10), MOTSPDL, MOTSPDH);
 motor.setSpeed(motorSpeed);
  while (!stopPump) {
      lcd.setCursor(0, 0);
     motor.step(stepsPerDeciMl);
      currentVolume += 0.1;
      if(serial){
        Serial.println(pumped);
       Serial.print(currentVolume,2); Serial.print("/");Serial.print(pumpVolume,
1);Serial.print("ml");Serial.println("");
         }
       else{
         lcd.print(pumped);
        lcd.print(currentVolume,2); lcd.print("/"); lcd.print(pumpVolume,1); lcd.
print("ml");
         }
      if (currentVolume >= pumpVolume) {
        stopPump = true;
       }
   }
}

result runPump() {
  attachInterrupt(0, stopPumping, RISING);
  lcd.setCursor(0, 1);
  lcd.print("Stop      ");
  stopPump = false;
```

```
  if (mode == 1)
    cPump(spd);
  else
    sPump(spd, volf);
  lcd.setCursor(0, 0);
  lcd.print("Done!  ");
  lcd.setCursor(0, 1);
  lcd.print("Run again");
  detachInterrupt(0);
  return proceed;

}



unsigned long calibPump(int motorSpeed, unsigned long * steps, bool serial=false)
{
  while (!digitalRead(SEL_BTN)) {}
  attachInterrupt(0, stopPumping, RISING);
  motor.setSpeed(motorSpeed);
  stopPump = false;
  unsigned long currentTime, startTime = 0;
  startTime = millis();
  * steps = 0;
  uint16_t stepsPre = 0;
  if(serial){
    Serial.println("Press enter at end");
     }
  do {
   currentTime = (millis() - startTime) / 1000;
   if(serial){
    char c;
      Serial.print("Time: ");
      Serial.print(currentTime);
      Serial.print(" s");
      Serial.print(" | ");
      Serial.print(* steps);
      Serial.println();
      if(Serial.available()){
      c = Serial.read();
      if(c==10 || c==13 || c==64){
        stopPump = true;
       endTime=(currentTime*1000)+startTime;
        }
       }
   }
   else{
    lcd.clear();
    lcd.print("Time: ");
```

```
    lcd.print(currentTime);
    lcd.print(" s");
    lcd.print(" | ");
    lcd.print(* steps);
    lcd.setCursor(0, 1);
    lcd.print(">Stop(Vol. reached)");
    }
    motor.step(1); * steps += 1;

  } while (!stopPump);
  detachInterrupt(0);
  return endTime - startTime;
}

void runCalib(){
  runCalibration(false);
  }

void runCalibration(bool serial) {
  if(serial){
    Serial.println("Step 1 - low speed ");
    Serial.println("Press Enter ...");
    char c;
    while(c!=10 && c!=13 && c!=64){
      if(Serial.available()){
      c = Serial.read();
      }
    }
  }
  else{
    lcd.clear();
    lcd.print("Step 1 - low speed ");
    lcd.setCursor(0, 1);
    lcd.print(">Start calib. step.");
    delay(SOFT_DEBOUNCE_MS);
    while (digitalRead(SEL_BTN)) {
    delay(SOFT_DEBOUNCE_MS);
    }
  }
  unsigned long stepsL = 0;
  unsigned long calibTimeL = calibPump(MOTSPDL, &stepsL,serial);

  if(serial){
    Serial.println("Step 2 - high speed");
    Serial.println("Press Enter ...");
    char c;
    while(c!=10 && c!=13 && c!=64){
      if(Serial.available()){
      c = Serial.read();
```

```
      }
     }
    }
  else{
    lcd.clear();
    lcd.print("Step 2 - high speed");  //tady bude krok motoru
    lcd.setCursor(0, 1);
    lcd.print(">Start calib. step.");
    delay(SOFT_DEBOUNCE_MS);
    while (digitalRead(SEL_BTN)) {
      delay(SOFT_DEBOUNCE_MS);
     }
  }

  unsigned long stepsH = 0;
  unsigned long calibTimeH = calibPump(MOTSPDH, &stepsH,serial);

  topSpeed = volCalf / ((float)calibTimeH / 60000.0);
  minSpeed = volCalf / ((float)calibTimeL / 60000.0);
  bool failed = false;
  if(minSpeed>=topSpeed ){
    if(serial){
     Serial.println("Calibration failed.");
    }
    else{
      lcd.clear();
     lcd.print("Calibration failed.");
     lcd.setCursor(0, 1);
      }
     minSpeed = 1.0;
     topSpeed = 10.0;
     failed = true;
    }
  else if(minSpeed == 0.0){
      minSpeed = 1.0;

  }

  uint16_t volDeciML = (uint16_t)(volCalf * 10);
  unsigned long stepsAvr = (stepsL+stepsH)/2;
  stepsPerDeciMl = stepsAvr / (unsigned long)(volCalf * 10);

if(serial){
   Serial.println(calEnd);
   Serial.println("Restart in: ");
    for (int i = 5; i > 0; i--) {
   Serial.print(i);
   Serial.print(" s");
   Serial.println();
```

```
    delay(1000);
  }
  }
  else{

    if(!failed){
    lcd.clear();
    lcd.print(calEnd);
       }

    lcd.setCursor(0, 1);
    lcd.print("Restart in: ");
    for (int i = 5; i > 0; i--) {
     lcd.print(i);
     lcd.print(" s");
     lcd.setCursor(12, 1);
     delay(1000);

    }
  }


 EEPROM.put(TOPSPD_ADR, topSpeed);
 EEPROM.put(MINSPD_ADR, minSpeed);
 EEPROM.update(STP_ADR, stepsPerDeciMl % 256);
 EEPROM.update(STP_ADR + 1, stepsPerDeciMl / 256);

 digitalWrite(RESET, LOW);

}

int doSerialCommand(char * line){
   //return value:
   // -3  Invalid parameters
   // -2: Invalid syntax
   // -1: se comand
   //  1: Exit
   //  0: Command execude
  char * space = strchr(line, ' ');
  if(strncmp(line,"info",strlen(line))==0 && space == NULL){
    Serial.println("calibrate VOL");
    Serial.println("pump -c SPD");
    Serial.println("pump -s VOL SPD");
    Serial.println("settings VOL SPD MOD");
    Serial.println("exit");
    return 0;
  }
  else if(strncmp(line,"exit",strlen(line))==0 && space==NULL){
    //konec
```

```
    return 1;
    }
   else if (space==NULL){
    return -1;
    }

 char * command = line;
 * space = '\0';
 char * param;
 int len = 0;
 float spd;
 float volf;
 int spc;
if(strncmp(command,"pump",strlen(command))==0){
   char * dash = strchr(space+1, '-');
   if(dash==NULL){
     return -2;
     //invalid syntax
     }
   else{
     param = dash+1;
     }
   if(* param=='c' || * param=='C'){

         spd = atof(dash+3);
        if(spd == 0.0 || spd>topSpeed || spd<minSpeed){

           return -3;
           }
          attachInterrupt(0, stopPumping, RISING);
           stopPump = false;
           cPump(spd,true);
           Serial.println("Done!  ");

    }else if (* param=='s' || * param=='S'){
        char * space2 = strchr(param, ' ');

        if(space2 == NULL){
           return -2;
           }
         * space2 = '\0';
        spd = atof(dash+3);
        volf = atof(space2+1);
        if(spd == 0.0 || volf==0.0 || spd<minSpeed || spd>topSpeed || volf<0){
           //invalid syntax
           return -3;
           }
          attachInterrupt(0, stopPumping, RISING);
           stopPump = false;
```

```
            sPump(spd,volf,true);
            Serial.println("Done!  ");
       }else{
         return -2;
         }

 }else if (strncmp(command,"calibrate",strlen(command))==0){
      volf = atof(space+1);
      if(volf==0.0 || volf<0){ return -3;}
      runCalibration(true);

 }else if(strncmp(command,"settings",strlen(command))==0){
    char * space2 = strchr(space+1, ' ');
    if (space2 == NULL)
      return -2;
    char * space3 = strchr(space2+1, ' ');
    if (space3 == NULL)
      return -2;
    * space2 = '\0'; * space3 = '\0';

    float vols = atof(space+1);
    float spds = atof(space2+1);
    int mods = atoi(space3+1);
    if(vols==0.0 || spds==0.0 || mods == 0 || spds<minSpeed || spds>topSpeed ||
volf<0){
      Serial.println(vols);
      Serial.println(spds);
      Serial.println(mods);
       return -3;
       }
    else{
      volDeff = vols;
      spdDef = spds;
      defMode = mods;
      saveSet();
       }
   }else{
     //invalid command
     return -1;

  }
  return 0;
  }

void serialRemote() { //https://www.root.
cz/clanky/ovladani-arduina-v-realnem-case-z-pocitace/

  Serial.begin(9600);
  Serial.println ("Pump Driver");
```

```
Serial.print(">");
char lineCmd [60];
int i=0;
char c;
int res;
while(true){
 if(Serial.available()){
  c = Serial.read();

  Serial.print(c);

  if(c==10 || c==13 || c==64){//newline
    Serial.print("\n");
      if(i>60){res = -2;}
      else{
        res = doSerialCommand(lineCmd);
       }
      switch(res){
        case -2:{
           Serial.println("Invalid syntax");
           Serial.print(">");
           break;
         }
        case -1:{
           Serial.println("Invalid command");
           Serial.print(">");
            break;
         }
        case -3:{
            Serial.println("Invalid parameters");
            Serial.print(">");
            break;
         }
        case 1:{
           Serial.println("Good bye!");
           break;
         }
        case 0:{
          Serial.print(">");
         }
       }
    lineCmd[0]='\0'; i=0;

    if(res==1){
      break;
     }
  }
  else if(c==8 || c==127){
    i--;
```

```
      if(i<0)
        i=0;
      lineCmd[i]='\0';

      }
    else{
      lineCmd[i]=c;
      lineCmd[i+1]='\0';
      i++;
    }
   }
  }
  Serial.end();

  }

void saveSet() {
//https://arduino8.webnode.cz/news/lekce-17-arduino-a-eeprom/
;
 EEPROM.put(VOL_ADR,volDeff);
 EEPROM.put(SPD_ADR, spdDef);
 EEPROM.update(MOD_ADR, defMode);

  volf = volDeff;
  spd = spdDef;
  mode = defMode;
  lcd.setCursor(0,1);
  lcd.print("Saved");
  Serial.println("Saved");
}

void readSet() {

  EEPROM.get(VOL_ADR, volDeff);
  EEPROM.get(SPD_ADR,spdDef);
  defMode = EEPROM.read(MOD_ADR);
  if (defMode != 2 && defMode != 1) {
    defMode = 1;
   EEPROM.update(MOD_ADR, defMode);
  }
  if (spdDef > topSpeed) {
    spdDef = topSpeed;
  }
  if(spdDef < minSpeed){
    spdDef = minSpeed;
    }
  if (volDeff > 500.0)
  {
    volDeff = 500.0;
```

```
  }
}

//
https://github.com/neu-rah/ArduinoMenu/blob/a0440c39a9de5453c1b2141035e6b6eb11e1fd
1e/examples/dynamic/dynamic/dynamic.ino

TOGGLE(mode, modeMenu, "Mode: ", doNothing, noEvent, wrapStyle
       , VALUE("Continuous", 1, doNothing, noEvent)
       , VALUE("Specific volume", 2, doNothing, noEvent)
      );
TOGGLE(defMode, modeMenuDef, "Def. mode: ", doNothing, noEvent, wrapStyle
       , VALUE("Continuous", 1, doNothing, noEvent)
       , VALUE("Specific volume", 2, doNothing, noEvent)
      );

prompt* startMenuData[] = {
  &modeMenu,
 new decimalslField<typeof(volf)>(volf, "Volume","ml",1.0,500.0,0.
1,1,doNothing,noEvent,wrapStyle),
 new decimalslField<typeof(spd)>(spd, "Speed","ml/min",minSpeed,topSpeed,0.
1,1,doNothing,noEvent,wrapStyle),
  new prompt("Run", (Menu::callback)runPump, enterEvent ),
};

menuNode& startMenu = *new menuNode(
                        "Start",
                       sizeof(startMenuData) / sizeof(prompt*),
                        startMenuData,
                        doNothing,
                        noEvent,
                        wrapStyle
                      );

prompt* calibMenuData[] = {
 new decimalslField<typeof(volCalf)>(volCalf, "Volume","ml",1.0,500.0,0.
1,1,doNothing,noEvent,wrapStyle),
  new prompt("Run", (Menu::callback)runCalib, enterEvent),
};
menuNode& calibMenu = *new menuNode(
                        "Calibration",
                       sizeof(calibMenuData) / sizeof(prompt*),
                        calibMenuData,
                        doNothing,
                        noEvent,
                        wrapStyle);

prompt* setMenuData[] = {
  &modeMenuDef,
```

```
 new decimalslField<typeof(volDeff)>(volDeff, "Def.vol","ml",1.0,500.0,0.
1,1,doNothing,noEvent,wrapStyle),
 new decimalslField<typeof(spdDef)>(spdDef, "Def.spd","ml/min",minSpeed,topSpeed,
0.1,1,doNothing,noEvent,wrapStyle),
  new prompt("Save", (Menu::callback)saveSet, enterEvent),

};
menuNode& setMenu = *new menuNode(
                      "Settings",
                     sizeof(setMenuData) / sizeof(prompt*),
                      setMenuData,
                      doNothing,
                      noEvent,
                      wrapStyle
                    );

prompt* pcCtrlMenuData[] = {
  new prompt("Connect", (Menu::callback)serialRemote, enterEvent),
};
menuNode& pcCtrlMenu = *new menuNode(
                         "PC Control",
                        sizeof(pcCtrlMenuData) / sizeof(prompt*),
                         pcCtrlMenuData,
                         doNothing,
                         noEvent,
                         wrapStyle
                       );

prompt* topMenuData[] = {
  &startMenu,
  &calibMenu,
  &setMenu,
  &pcCtrlMenu,

};
menuNode& topMenu = *new menuNode(
                      "Pump Driver",
                     sizeof(topMenuData) / sizeof(prompt*),
                      topMenuData,
                      doNothing,
                      noEvent,
                      wrapStyle
                    );



MENU_OUTPUTS(out, MAX_DEPTH
             , NONE
            , LIQUIDCRYSTAL_OUT(lcd, {0, 0, 16, 2}) //must have 2 items at least
```

```
            );

//MENU_INPUTS(in);
chainStream<0> in(NULL);
NAVROOT(nav, topMenu, MAX_DEPTH, in, out);




void setup() {
  interrupts(); // zapnutí podpory přerušení
  digitalWrite(RESET, HIGH);
  delay(200);
  pinMode(RESET, OUTPUT);
  if (INIT != 1) {
    initSets(); // inicializace při prvním spuštění aplikace
  }
  //konfigurace vstupních avýstupních pinů
  pinMode(NAV_BTN, INPUT_PULLUP);
  pinMode(SEL_BTN, INPUT_PULLUP);
  pinMode(BACK_BTN, INPUT_PULLUP);

  pinMode(MOT1, OUTPUT);
  pinMode(MOT2, OUTPUT);
  pinMode(MOT3, OUTPUT);
  pinMode(MOT4, OUTPUT);

  lcd.begin(16, 2);

  readSet();
  volf = volDeff;
  spd = spdDef;
  mode = defMode;

}

void loop() {

  if (!digitalRead(SEL_BTN)) {
    delay(SOFT_DEBOUNCE_MS);
    while (!digitalRead(SEL_BTN));
    nav.doNav(enterCmd);
    delay(SOFT_DEBOUNCE_MS);
  }
  if (!digitalRead(NAV_BTN)) {
    delay(SOFT_DEBOUNCE_MS);
    while (!digitalRead(NAV_BTN)); //wait for button release
    nav.doNav(upCmd);
    delay(SOFT_DEBOUNCE_MS);
```

```
  }
  if (!digitalRead(BACK_BTN) && &(nav.active()) != &topMenu) {
    delay(SOFT_DEBOUNCE_MS);
    while (!digitalRead(BACK_BTN)); //wait for button release
    nav.doNav(escCmd);
    delay(SOFT_DEBOUNCE_MS);
  }
  nav.doOutput();
}
```

```
//-----decimalField.h-----
#include <menu.h>
#include<menuIO/serialOut.h>
#include<menuIO/chainStream.h>
#include<menuIO/serialIn.h>

using namespace Menu;
#define DECIMALSFLIED_DEFAULT 1


template<typename T>
class decimalslField :public menuField<T> {
private:
  idx_t decimals;
public:
  decimalslField(constMEM menuFieldShadow<T>& shadow) :menuField<T>(shadow) {
decimals = DECIMALSFLIED_DEFAULT; }
  decimalslField(
    T &value,
    constText* text,
    constText*units,
    T low,
    T high,
    T step,
    T tune,
    action a = doNothing,
    eventMask e = noEvent,
    styles s = noStyle
  ) :decimalslField(*new menuFieldShadow<T>(value, text, units, low, high, step,
tune, a, e, s)) {}

  Used printTo(navRoot &root, bool sel, menuOut& out, idx_t idx, idx_t len, idx_t
panelNr = 0) override {
   menuField<T>::reflex = menuField<T>::target();
    idx_t l = prompt::printTo(root, sel, out, idx, len);
    bool ed = this == root.navFocus;
    //bool sel=nav.sel==i;
    if (l < len) {
      out.print((root.navFocus == this&&sel) ? (menuField<T>::tunning ? '>' :
':') : ' ');
      l++;
      if (l < len) {
        l += out.print(menuField<T>::reflex, decimals);//NOTE: this can exceed
the limits!
        if (l < len) {
          l += print_P(out, fieldBase::units(), len);
        }
      }
    }
```

```
    return l;
  }

  void setDecimals(idx_t d) { decimals = d; }
  idx_t getDecimals(void) { return(decimals); }
};
//-----CustomfloatField----------------END
```