Jihočeská univerzita v Českých Budějovicích

Ekonomická fakulta

Katedra aplikované ekonomie a ekonomiky

Bakalářská práce

Vytvoření zařízení pro dálkové ostření objektivu

Autor práce: Novotný Jiří

Vedoucí práce: Mgr. Radim Remeš, Ph.D.

České Budějovice

2023

JIHOČESKÁ UNIVERZITA V ČESKÝCH BUDĚJOVICÍCH

Ekonomická fakulta

Akademický rok: 2021/2022

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Jiří NOVOTNÝ**
Osobní číslo:	**E20063**
Studijní program:	**B6209 Systémové inženýrství a informatika**
Studijní obor:	**Ekonomická informatika**
Téma práce:	**Vytvoření zařízení pro dálkové ostření objektivu**
Zadávající katedra:	**Katedra aplikované matematiky a informatiky**

Zásady pro vypracování

Cílem práce je vytvořit a sestavit plně funkční zařízení pro dálkové ostření objektivu (wireless lens control system – WLCS), které bude ovládáno pomocí dvou jednodeskových počítačů Arduino, jeden bude použit pro přijímač a druhý pro vysílač. Pro vývoj tohoto zařízení bude využito elektronických součástek (např. krokový motor s řadičem, potenciometr, tlačítko, LED, modul pro bezdrátovou komunikaci, aj.) které budou vyhodnocovány a použity pro softwarové ovládání WLCS. Pro vývoj bude využito Arduino IDE. Zařízení WLCS bude disponovat mnohými funkcemi, např. pro nastavení relativní maximální a minimální polohy, přenos a ovládání ostření bezdrátově, automatické zjištění maximální a minimální polohy.
Metodický postup:

1. Studium odborné literatury.
2. Popis použitých technologií pro vývoj aplikace.
3. Návrh, popis vývoje a implementace aplikace.
4. Zhodnocení použitelnosti aplikace pro nasazení v reálném prostředí.
5. Závěr a doporučení.

Rozsah pracovní zprávy:	**40 – 50 stran**
Rozsah grafických prací:	**dle potřeby**
Forma zpracování bakalářské práce:	**tištěná**

Seznam doporučené literatury:

1. Boxall, J. (2021). *Arduino Workshop*. No Starch.
2. Dunbar, N. (2020). *Arduino Software Internals: A Complete Guide to How Your Arduino Language and Hardware Work Together*. Apress.
3. Margolis, M., Jepson, B., Weldin, N. R. (2020). *Arduino Cookbook*. O'Reilly.
4. Monk, S. (2018). *Programming Arduino next steps: Going further with sketches*. McGraw Hill.
5. Nussey, J. (2018). *Arduino for dummies*. Wiley.

Vedoucí bakalářské práce: **Mgr. Radim Remeš, Ph.D.**
Katedra aplikované matematiky a informatiky

Datum zadání bakalářské práce: **11. ledna 2022**
Termín odevzdání bakalářské práce: **14. dubna 2023**

JIHOČESKÁ UNIVERZITA
V ČESKÝCH BUDĚJOVICÍCH
EKONOMICKÁ FAKULTA
Studentská 13 (26)
370 05 České Budějovice

L.S.

doc. Dr. Ing. Dagmar Škodová Parmová
děkanka

doc. RNDr. Tomáš Mrkvička, Ph.D.
vedoucí katedry

V Českých Budějovicích dne 16. března 2022

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně pouze s použitím pramenů a literatury uvedených v seznamu citované literatury. Prohlašuji, že v souladu s § 47b zákona č. 111/1998 Sb. v platném znění souhlasím se zveřejněním své bakalářské práce, a to v nezkrácené podobě elektronickou cestou ve veřejně přístupné části databáze STAG provozované Jihočeskou univerzitou v Českých Budějovicích na jejích internetových stránkách, a to se zachováním mého autorského práva k odevzdanému textu této kvalifikační práce. Souhlasím dále s tím, aby toutéž elektronickou cestou byly v souladu s uvedeným ustanovením zákona č. 111/1998 Sb. zveřejněny posudky školitele a oponentů práce i záznam o průběhu a výsledku obhajoby kvalifikační práce. Rovněž souhlasím s porovnáním textu mé kvalifikační práce s databází kvalifikačních prací Theses.cz provozovanou Národním registrem vysokoškolských kvalifikačních prací a systémem na odhalování plagiátů. V Českých Budějovicích dne …… Jiří Novotný …….

1 Úvod

Jako téma své bakalářské práce jsem si vybral programování a vytvoření systému pro dálkové ostření objektivu na platformě Arduino nano. Jedná se o velmi rozšířenou a oblíbenou platformu pro vývoj zařízení, které nepotřebují velký výpočetní výkon a obrovské paměťové kapacity, to je ideální volba právě pro zařízení tohoto typu.

Hlavní motivace pro výběr tohoto tématu byla původní myšlenka k zakoupení tohoto zařízení, ale při pohledu na cenu, která se pohybuje okolo 7 500 Kč, jsem začal přemýšlet, že takové zařízení by se dalo vyrobit.

2 Cíl práce

Cílem této bakalářské práce je navrhnout finančně dostupný systém pro dálkové ovládání clony nebo ostření u objektivů, to znamená potřeba vytvoření dvou separátních zařízení, které spolu komunikují pomocí bezdrátové komunikace.

Vhodný výběr komponent a jejich zprovoznění s platformou Arduino Nano. Vytvoření vlastního kódu pro požadovanou funkci obou zařízení v prostředí Arduino IDE 2.0. Modelování 3D součástek pomocí modelovacího programu Blender. Navržení základní desky pro umístění a komunikaci komponentů pomocí online aplikace EASYeda.

3 Základní informace

3.1.1 Použití a popis funkčnosti bezdrátového systému pro ostření objektivů.

V roce 2023, kdy se mnoho fotoaparátů a vizuálních tvůrců spoléhá na chytré telefony a fotoaparáty s automatickým ostřením, může systém bezdrátový ostřící systém vypadat jako rarita nebo něco z minulosti. Pro určité druhy obsahu však dobrý systém bezdrátový ostřící systém a zkušený ostřič jsou potřeba. Záběry, které s automatickými systémy prostě nelze vytvořit. Lidský dotek, i se svými nedokonalostmi, je něco, co „dýchá“ a co lze na obrazovce vnímat, a to i podvědomě, říká se, že obraz má „duši“. (Prada, 2022)

Používáme-li objektiv se širokou clonou (například f/1,2), je hloubka ostrosti tak malá, že i nepatrný pohyb obsluhy fotoaparátu nebo objektu bude vyžadovat úpravu zaostření, aby vše zůstalo ostré. Velké kinofilmové objektivy mají velké zaostřovací kroužky s dostatečnou rotací, aby byly přesné, ale zároveň vyžadují špičkový systém následného zaostřování, protože ručně je to téměř nemožné. Ve scénářích, jako jsou svatby, události nebo nízkorozpočtové filmy, kde chceme mít přesnou kontrolu nad zaostřováním, bude dokonalým řešením systém follow focus. (Prada, 2022)

3.2 Vysvětlivky

Decoupling – Stabilizace výkyvu proudu pomocí kondenzátoru.

Inputu – Vodič, který je definován jako vstup, například vstup pro spínač.

Mikrokontroler – Srdce Arduina, kde se nachází a spouští veškerý kód.

Konektivita - Konektivita se týká míry, jakou jsou prvky, systémy nebo osoby propojeny nebo schopny komunikovat a interagovat mezi sebou.

Modifikátor – Slovo nacházející se před definicí proměnné nebo funkce, které upravuje její chování.

Unsigned – Modifikátor, který upravuje proměnnou pouze pro kladná čísla

Iterace – Iterace je proces opakovaného provádění stejného úkonu nebo postupu, často za účelem dosažení určitého cíle nebo vyřešení problému.

Algoritmus – Systematické řešení určitého problému. Například algoritmus pro součet dvou čísel.

Analyzace - je systematický proces rozkladu, zkoumání a vyhodnocování prvků nebo dat s cílem porozumět jejich struktuře, vzorcům nebo vztahům a odhalit hlubší významy a souvislosti.

False – Hodnota reflektující Nepravdu. Například 1 + 1 = 3 není pravda, tudíž False.

Debugging - Je proces nalezení a odstraňování chyb (bugů) v programu nebo systému, aby se zajišťovala jeho správná funkčnost a chování.

Buffer - Je oblast paměti sloužící k dočasnému ukládání dat mezi dvěma zařízeními nebo částmi programu, umožňuje vyrovnání rychlostních rozdílů a zajišťuje plynulý tok informací.

Blender - Blender je otevřený 3D grafický software, který umožňuje tvorbu a animaci 3D modelů, vizuálních efektů, herního obsahu a dalších digitálních projektů.

Plugin - Plugin je doplňkový software, který rozšiřuje funkčnost existujícího programu nebo webového prohlížeče tím, že poskytuje nové nástroje, funkce nebo integrace s dalšími službami.

Pressure angle - Úhel náklonu, také nazýván úhel tlaku, je úhel mezi směrem síly (nebo tlaku) působící na zuby ozubeného kola a kolmou linií na povrch zubů, a je klíčovým parametrem při návrhu ozubených kol ovlivňujícím jejich výkon, účinnost a životnost.

Filamenty - Filamenty 3D tiskárny jsou materiály ve formě tenkých prutů, které se používají k vytváření třírozměrných objektů tím, že jsou postupně rozpouštěny a tisknuty ve vrstvách.

LED – Dioda emitující světlo

4 Ovládání a použití

Prvně je zapotřebí si vymezit, co vše zařízení musí umět a jak se zařízení ovládá. Zařízení, které má uživatel v ruce, budu v této práci nazývat slovem vysílač. Druhé zařízení, které je připevněno k objektivu, budu nazývat přijímač.

Vysílač se ovládá pomocí tlačítka a potenciometru, Tlačítko slouží pro vstup do kalibračního módu a potenciometr pro detekci rotace. Po zapnutí obou zařízení jsou automaticky spárována a připravena k použití, otáčení potenciometrem zapříčiní rotaci krokového motoru na přijímači, který pohybuje ozubeným kolem v kontaktu s ozubením na objektivu.

4.1 Vstupy

4.1.1 Spínač

Jeden ze základních vstupů, který bude sloužit k interakci se zařízením je spínač, zde konkrétně použité tlačítko.

Když je tlačítko stisknuto, je stisknut mikrospínač nebo přepnut přepínač, dvě kovové části se k sobě přiblíží. Uživateli se může zdát, že ke kontaktu dojde okamžitě. To však není správná úvaha. Uvnitř spínače jsou pohyblivé části, obvykle pružinka nebo pružný kov. Když je spínač stisknut, dojde nejprve ke kontaktu s druhou kovovou částí, ale jen v krátkém zlomku mikrosekundy. Pak se kontakt vytvoří o něco déle a pak opět o něco déle. Nakonec je spínač zcela uzavřen. Spínač se pohybuje mezi kontaktem a ne kontaktem. Když je spínač zavřený, oba kontakty se skutečně rozpojí a znovu spojí, obvykle 10 až 100krát během periody přibližně 1 ms." ("Umění elektroniky", Horowitz & Hill, druhé vydání, str. 506.)

Tento fakt vytváří komplikaci při čtení vstupu, protože při jednom stisku tlačítka integrovaný obvod zaznamená veškeré změny kontaktu spínače, což vytváří značnou komplikaci, která nám zabraňuje jednoduché čtení takového vstupu. Tento fakt bude potřeba vyřešit pomocí tzv. “debouncingu”.

4.1.2 Potenciometr

Potenciometr je elektrická součástka používaná k nastavování odporu v elektrickém obvodu. Také je známý pod zkratkou "pot" nebo jako proměnný rezistor. Jeho hlavní funkcí je umožnit uživateli nebo elektronickému zařízení upravit průchod elektrického proudu, což může ovlivnit různé parametry obvodu, jako je hlasitost zvuku, intenzita světla, rychlost motoru nebo jakoukoliv jinou veličinu, která závisí na elektrickém odporu. Potenciometry mají tři hlavní piny, z nichž dva jsou obvykle koncové body, na které je přivedeno referenční napětí a třetí je pohyblivý bod, tento výstup se většinou používá pro snímání vstupu. Když je potenciometr otáčen nebo posouván, pohyblivý bod mění své umístění mezi koncovými body, což mění délku odporu, který je překonáván elektrickým proudem. To umožňuje regulaci výstupní hodnoty v obvodu. Mechanické potenciometry mají nevýhodu v opotřebení jejich fyzických částí, například odporová dráha a jezdec. Existují také elektrické potenciometry přezdívané digipot. (Michalec, 2021)

4.2 Výstupy

4.2.1 světelná dioda

LED (Light Emitting Diode) je polovodičová elektronická součástka, která emituje světlo. Když elektrony proudí přes tento polovodič, setkávají se s dírami, což vyvolává uvolňování fotonů a následně záření světla. LEDky jsou charakteristické svou nízkou spotřebou energie, dlouhou životností a rychlým reakčním časem. Oproti tradičním žárovkám nebo zářivkám vydávají LEDky mnohem méně tepla, což je dělá energeticky efektivnějšími a zabraňuje zbytečnému vyzařování tepla. LED technologie se v posledních letech výrazně rozvinula, což umožnilo vytvářet efektivnější a jasnější LEDky s širší škálou aplikací, a staly se tak dominantním prvkem v moderním osvětlení a elektronice. (LEDme s. r. o., 2021)

4.3 Bezdrátová komunikace

4.3.1 Baterie

Baterie s NiCD články: Jedná se o starší typ článků, který býval často používán v mobilních zařízeních. Tyto baterie nabízejí schopnost dodávat vysoký proud bez výrazného poklesu napětí a dobrou funkčnost i při nižších teplotách. Avšak mají několik nevýhod, včetně větší hmotnosti, krátké životnosti (cca 200 cyklů) a výrazného "paměťového efektu", který snižuje kapacitu baterie. Dále obsahují škodlivé látky jako kadmium. Dnes se používají hlavně pro aku nářadí, kde je potřeba vyšších proudů, ale v jiných přístrojích už nemají tolik uplatnění. (Reichl, 2015)

Baterie s NiMH články: Tyto články jsou vylepšenou verzí NiCD baterií. Nahradily škodlivé kadmium hybridní slitinou, což snižuje znečištění životního prostředí. Mají menší samovybíjení, nižší hmotnost a vyšší životnost (cca 300-400 cyklů). Nicméně stále trpí "paměťovým efektem", což vyžaduje správné nabíjení. Používají se pro aku nářadí a dobíjecí baterie (AA, AAA). (Reichl, 2015)

Baterie s Li-Ion články: Tento typ článků je nejrozšířenější v elektronických zařízeních. Nabízejí lepší vlastnosti než NiCD a NiMH články, jako je snížení hmotnosti až o 50 %, menší samovybíjení a delší životnost (až kolem 1 000 cyklů). Navíc téměř nevykazují "paměťový efekt". Nicméně jejich vybíjecí proudy nejsou tak velké a jsou citlivé na teplotní změny. Používají se v moderních elektronických zařízeních. (Reichl, 2015)

Baterie s Li-Pol články: Li-Pol články jsou nejmodernější a očekává se, že budou nahrazovat Li-Ion články. Odlišují se pevným elektrolytem, což zvyšuje bezpečnost a snižuje hmotnost. Mají stejné výhody jako Li-Ion články a také si zachovávají vlastnosti při nižších teplotách. Nevýhodou je rychlejší stárnutí, bez ohledu na použití. Jsou známé pro svou tvarovatelnost pro zařízení. (Reichl, 2015)

5 Arduino

Termín Arduino se používá jak pro označení fyzické desky Arduino (jejíž nejoblíbenější typ je Arduino Uno), tak systém Arduino jako celek. Systém zahrnuje také software, který potřebujete spustit na počítači (k programování desky), a periferní štíty, které můžete připojit k počítači. Abyste mohli Arduino používat, potřebujete také "reálný" počítač. Může to být Mac, PC se systémem Windows nebo Linux. PC, nebo dokonce něco tak skromného, jako je Raspberry Pi. Hlavním důvodem, proč je potřeba

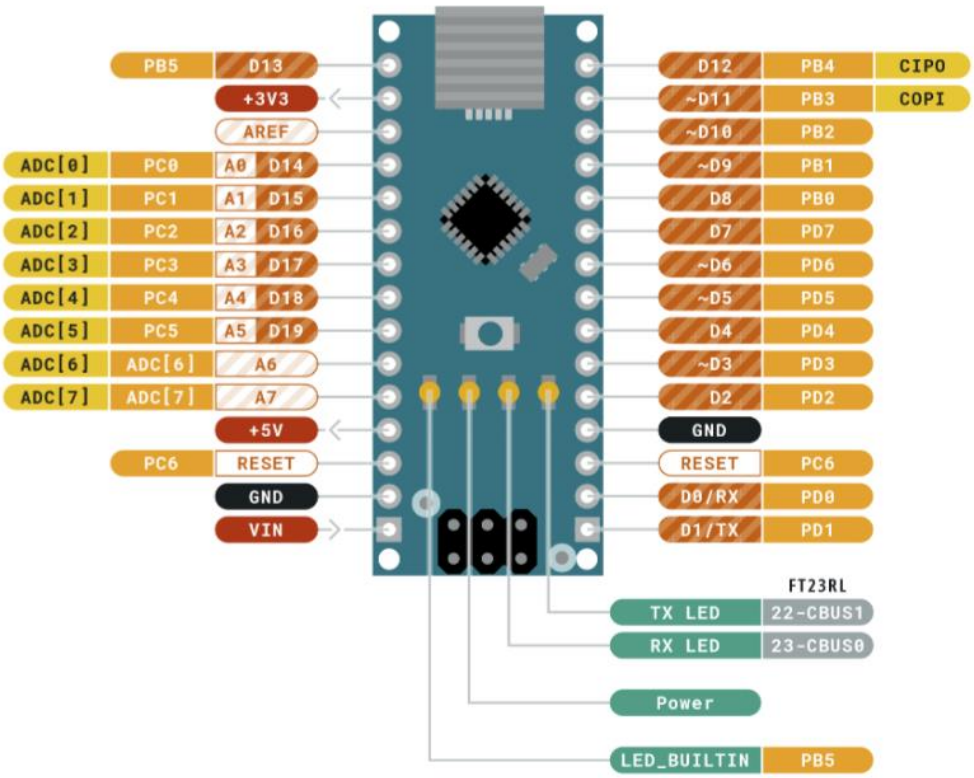
počítač, je to, že můžete do desky Arduino nahrát programy. Jakmile jsou tyto programy nahrány do paměti Arduina, pak mohou běžet samostatně. (Monk, 2018)

Arduino se od běžného počítače liší tím, že nemá téměř žádnou paměť, nemá operační a nemá klávesnici, myš ani rozhraní na obrazovce. Jeho účelem je ovládat věci na základě senzorů a dalších modulů. Dá se připojit například sensor pro měření vlhkosti. (Monk, 2018)

5.1 Mikrokontroler

Srdcem Arduina je mikrokontroler ATmega328. Všechno ostatní na desce se v podstatě se stará o to, aby deska byla napájena a mohla komunikovat se stolním počítačem. Mikrokontroler je skutečně malý počítač na čipu, kde je uložen veškerý kód. Nabízí více výpočetního výkonu a pamětí, než měly první domácí počítače. Má procesor, jeden nebo dva kilobajty paměti s náhodným přístupem. Paměti (RAM) pro ukládání dat, několik kilobajtů mazatelné programovatelné paměti jen pro čtení (EEPROM) a několik kilobajtů paměti pro ukládání dat (FLASH). (Monk, 2011)

Obrázek 1 Schéma uspořádání pinů



Zdroj : https://docs.arduino.cc/hardware/nano

ATmega328 je 28 pinový procesor, piny jsou pomocí Arduino desky vyvedeny na strany Arduina, jak je vidět na obrázku 1. Piny se dají využít jako vstupy nebo výstupy, jsou zde analogové a digitální piny. Analogové dokáží pracovat s hodnotou od 0 V do 5

V, kde dokáží rozlišit a převést hodnotu na přesnost 10 ti bitů, tedy 1024 hodnot, to se dá využít například pro plynulé tlumení LED. Naopak digitální piny dokáží pracovat pouze s hodnotou 0 V a 5 V, tedy zapnuto nebo vypnuto. (Monk, 2011)

Mikrokontroler ATmega328 disponuje 32 kB paměti typu Flash, která slouží pro ukládání samotného kódu, například kód pro funkce. 2 kB z této paměti zabírá samotný bootloader, což je malý program umístěný v paměti mikrokontroleru, umožňuje nahrávání nových programů (sketchů) do Arduina přes sériové rozhraní (USB nebo sériový port). Je umístěn na začátku paměti mikrokontroleru a je spuštěn po každém resetu nebo napájení. To znamená, že pro samotný kód je dostupné 30kB paměti. (Monk, 2018)

Další paměť, kterou obsahuje mikrokontroler je typu RAM. Paměť slouží pro ukládání proměnných, které se v průběhu času mění, například proměnná pro ukládání času. Ta je z tohoto důvodu uložena v této paměti, k dispozici je 2 kB paměti, to je oproti klasickým počítačům zlomek paměti, kde se paměť počítá v Gigabajtech. Každopádně je rozdíl mezi tím, jak se paměť využívá v klasickém počítači a v Arduinu. Po spuštění programu v počítači je do paměti nahrána kopie programu a nadále s tím pracuje. Oproti tomu RAM u Arduina slouží pouze k uchovávání obsahu proměnných a dalších dat týkajících se systému. Paměť RAM není trvalá, to znamená, že po odpojení napájení se paměť RAM vymaže. (Monk, 2018)

Pokud chceme uložit data v Arduinu permanentně je potřeba využít paměť typu EEPROM, elektronicky mazatelná paměť pro čtení, té je zde k dispozici 1 kB, to se rovná například 512 int proměnným. (Monk, 2018)

5.2 Arduino desky

Historie Arduina začíná v roce 2005, kdy byl založený italským inženýrem Massimo Banzim spolu s dalšími kolegy v rámci výukového programu na Interaktivním designovém institutu v Ivree, Itálie. Původně chtěli vytvořit jednoduchou platformu, která by umožňovala studentům snadno a levně experimentovat s elektronikou a programováním. To vedlo k vývoji prvního prototypu, který později dostal jméno "Arduino". První Arduino bylo postaveno na jednočipovém mikrokontroleru Atmel ATmega168, který měl jednoduché vstupy a výstupy, a bylo vybaveno sériovým rozhraním pro komunikaci s počítačem. Tato deska byla určena k výuce a umožňovala studentům snadno programovat a ovládat jednoduché elektronické projekty. (Monk, 2018)

Jedním z klíčových prvků, kterým se Arduino odlišovalo od ostatních vývojových platforem, byl jeho open-source charakter. Zakladatelé se rozhodli, že Arduino bude mít otevřený hardware i software, což znamenalo, že návrhy desek, schémata a zdrojové kódy budou volně dostupné pro všechny. Tento přístup umožnil komunitě rychle růst a přinášet stále nové nápady a inovace. S úspěchem a rostoucím zájmem se Arduino stalo stále populárnějším mezi nadšenci a tvůrci po celém světě. Postupně byly vytvářeny nové verze Arduina s různými funkcemi a možnostmi, což umožňovalo pokrýt širší spektrum projektů a aplikací. (Monk, 2018)

Díky své jednoduchosti, dostupnosti a otevřenosti se Arduino stalo základním nástrojem pro mnoho kreativních a technologických projektů. Bylo použito v různých

oborech, od uměleckých instalací a interaktivních uměleckých děl, přes automatizaci domácností, až po pokročilé IoT (Internet of Things) aplikace. Dnes existuje mnoho variant a verzí Arduina, a komunita kolem tohoto projektu je stále aktivní a rozšiřuje jeho možnosti a využití. Arduino se stalo symbolem otevřeného přístupu k elektronice a programování, a jeho vliv na svět bastlířů a technologických nadšenců je nepopiratelný. (Monk, 2018)

Obrázek 2 Porovnání velikostí mikrokontrolérů Arduino

Board	shape	Dimeter (mm)	Length (mm)	width (mm)	height (mm)	Weight (g)
UNO	Rectangle	-	75	54	13	25
NANO	Rectangle	-	45	17	13	7
MINI	Rectangle	-	30	18	11	2 (without pin header)
Pro MINI	Rectangle	-	34	18	11	2 (without pin header)
MICRO	Rectangle	-	48	18	13	13
ZERO	Rectangle	-	70	54	12	12
MEGA	Rectangle	-	110	53	13	37
DUE	Rectangle	-	103	53	12	36
FIO	Rectangle	-	65	28	13	9
GEMMA	Circle	28	-	-	5	2
LEONARDO	Rectangle	-	70	54	12	20
ETHERNET	Rectangle	-	75	54	13	28
YUN	Rectangle	-	75	54	16	45
UNO wifi	Rectangle	-	75	54	13	25
101	Rectangle	-	75	54	13	34
LILLYPAD	Circle	50	-	-	4	4.5
LILLYPAD USB	Circle	50	-	-	5	4.7

Zdroj : https://electropeak.com/learn/arduino-buying-guide-how-to-choose-right-arduino-board/

5.2.1 Arduino Nano

Arduino Nano je jedním z populárních vývojových zařízení z rodiny Arduino. Je to kompaktní a plnohodnotné Arduino deska, která nabízí široké možnosti pro tvorbu elektronických projektů. (Voda, 2017)

Základní vlastnosti a charakteristiky Arduino Nano:

1. **Mikrokontroler**: Arduino Nano je vybaven mikrokontrolerem ATmega328P, který je srdcem celé desky.
2. **Velikost**: Arduino Nano má velmi malé rozměry, což umožňuje snadné začlenění do menších a mobilních projektů. Standardní velikost je obvykle kolem 45 x 18 mm.
3. **Pinový konektor**: Arduino Nano má ohebné nožky (piny), které umožňují snadné zapojení a propojení s dalšími periferiemi nebo součástkami na prototypovací desce nebo plošném spoji.
4. **Napájení**: Arduino Nano může být napájen z USB nebo externím zdrojem napětí, jako jsou baterie nebo adaptéry. Lze napájet v rozpětí 2,8 V – 12 V
5. **Digitální a analogové piny**: Arduino Nano poskytuje digitální i analogové piny pro připojení senzorů, modulů a dalších elektronických zařízení.
6. **Integrované periferie**: Arduino Nano má integrované různé periferie jako např. LED diody, tlačítka, regulátory napětí.

Obrázek 3 Technické parametry Arduina Nano

Microcontroller	ATmega328
Architecture	AVR
Operating Voltage	5 V
Flash Memory	32 KB of which 2 KB used by bootloader
SRAM	2 KB
Clock Speed	16 MHz
Analog IN Pins	8
EEPROM	1 KB
DC Current per I/O Pins	40 mA (I/O Pins)
Input Voltage	7-12V
Digital I/O Pins	22 (6 of which are PWM)
PWM Output	6
Power Consumption	19 mA
PCB Size	18 x 45 mm

Zdroj : https://store.arduino.cc/products/arduino-nano

5.3 Arduino IDE 2.0

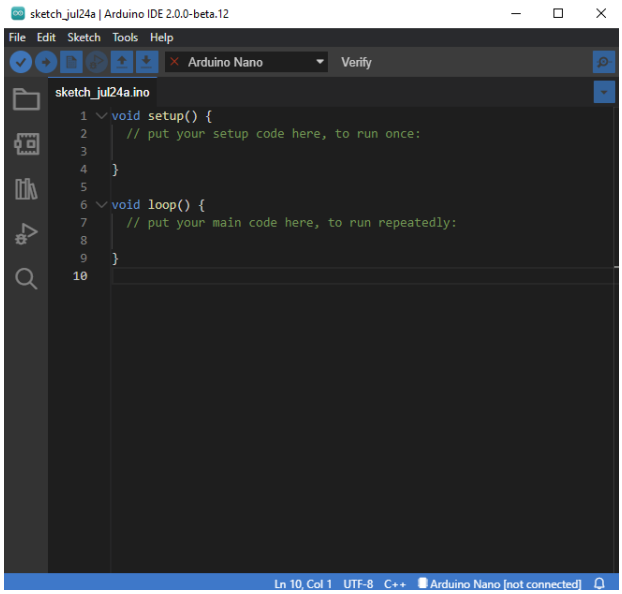
Software, který se používá k programování Arduina, se nazývá Arduino Integrated DevelopmentEnvironment 2.0 (Arduino IDE). Jde o novější verze předešlého prostředí nazývaného 1.0. Jedná se o prakticky stejné prostředí, které však nabízí upravené uživatelské rozhraní a funkce navíc jako jsou:

1. **Přepracované uživatelské rozhraní**: Arduino IDE 2.0 nabízí modernější a vylepšené uživatelské rozhraní s aktualizovaným vzhledem a novými možnostmi ovládání.
2. **Plnohodnotný editor souborů**: Nový editor souborů poskytuje lepší zážitek při programování s funkcemi jako zvýrazňování syntaxe, zalamování řádků, rychlé načítání a další.
3. **Integrace návrhu desek**: Arduino IDE 2.0 integruje nový nástroj pro návrh desek, který umožňuje vytvářet a konfigurovat vlastní desky Arduino.
4. **Plná podpora vylepšených verzí Arduino desek**: Nová verze IDE nabízí plnou podporu pro moderní desky Arduino, včetně Arduino Nano RP2040 Connect, Arduino Nano 33 IoT, a další.
5. **Podpora pro různé programovací jazyky**: Kromě klasického Arduino jazyka (založeného na Wiring) podporuje Arduino IDE 2.0 i další programovací jazyky, jako je C++.
6. **Lepší podpora verzování**: Soubory projektu jsou nyní uspořádány do složek, což umožňuje lepší organizaci a správu verzí. (“Arduino IDE 2 | Arduino Documentation,” b.r.)

Na obrázku 4 jde vidět samotné vývojové prostředí Arduina, hlavní způsob interakce probíhá pomocí horní lišty, kde se nachází klasická funkční tlačítka pro vytvoření a otevření sketche, import a zprávu knihoven, otevření již předdefinovaných sketchů, které obsahují základní kód pro jednoduché použití, například blikání LED.

Níže se potom nachází tlačítko Verify, které plní úlohu kontroly syntaxe a následně vypíše, kde se nachází chyba. Napravo se nachází tlačítko pro nahrání kódu do připojeného zařízení, to je možno vidět více napravo v kolonce Arduino Nano, dále se zde nachází tlačítko pro vytvoření nového sketche. Další dvě tlačítka slouží pro otevření a uložení sketchů. Poslední ikona na pravé straně slouží pro otevření sériové komunikace, kde se zařízením komunikuje, dají se odesílat hodnoty, ale i přijímat, hlavní využití bylo pro debugování.

Obrázek 4 Nový sketch připravený pro pasní kódu



Zdroj : Vlastní

5.3.1 Programování

Běžnou mylnou představou o Arduinech je, že mají vlastní programovací jazyk. Ve skutečnosti se programují v jazyce jednoduše nazývaném C. Tento jazyk je známý již od dob počátků výpočetní techniky. Co však Arduino poskytuje, je pěkná a snadno použitelná sada příkazů napsaných v jazyce C, které můžete používat ve svých programech. Často se poukazuje na to, že Arduino používá C++, objektově orientované rozšíření jazyka C. I když, přísně vzato, je to pravda, mít k dispozici pouze 1 nebo 2 kB paměti obecně znamená, že běžná struktura využívaná v objektově orientovaném programování, není obvykle dobrým nápadem s Arduino, takže kromě několika specializovaných oblastí fakticky je programováno v jazyce C. (Monk, 2018)

5.3.2 Jazyk C

Programovací jazyk C hraje klíčovou roli v kontextu programování s Arduinem. Arduino IDE (Integrované vývojové prostředí) podporuje programovací jazyk C, což umožňuje vývojářům psát kód pro Arduino desky v tomto jazyce. C je strukturovaný programovací jazyk s důrazem na efektivitu a flexibilitu. Vzhledem k tomu, že Arduino desky mají omezené prostředky (omezená paměť a výpočetní kapacita), je C vhodným jazykem pro jejich programování. Pomocí jazyku C, lze přímo ovládat hardware a pracovat s nízko úrovňovými detaily. (Monk, 2011)

5.4 Datové typy

V programovacím kontextu jsou datové typy specifikace, které určují, jaký typ dat může být uložen v proměnných nebo jaký typ hodnot mohou být použity v různých operacích. Datové typy jsou důležité pro to, jakým způsobem jsou data reprezentována, uložena a zpracovávána v paměti počítače. (Arduino Data Types – JavaTPoint, b.r.)

5.4.1 Int

Datový typ pro celá čísla může zahrnovat kladná, záporná a nulová čísla. Na Arduino Nano (a dalších deskách založených na ATmega) ukládá int 16bitovou (2bajtovou) hodnotu. To dává rozsah -32 768 až 32 767 (minimální hodnota -2^{15} a maximální hodnota (2^{15}) - 1). (Arduino Data Types – JavaTPoint, b.r.)

5.4.2 Unsigned Long

Proměnné Unsigned long jsou proměnné s rozšířenou velikostí pro ukládání čísel a ukládají 32 bitů (4 bajty). Na rozdíl od standardních dlouhých proměnných unsigned long neukládají záporná čísla, takže jejich rozsah je od 0 do 4 294 967 295 (2^{32}- 1). Hlavní použití proměnných typu unsigned je pro předcházení tzv „přetečení“. (Arduino Data Types – JavaTPoint, b.r.)

5.4.3 Boolean

Proměnná bool nabývá jedné ze dvou hodnot, true nebo false. Každá proměnná bool zabírá jeden bajt paměti, přestože hodnoty mohou nabývat pouze 1 nebo 0, tudíž 1 bit, to je zapříčiněno jednodušším zpracováním paměti. (Arduino Data Types - JavaTPoint, b.r.)

5.4.4 Byte

V bajtu je uloženo 8bitové číslo bez znaménka od 0 do 255.(Arduino Data Types - JavaTPoint, b.r)

Většinou je právě pro tyto malé hodnoty použit Byte místo int, dojde k ušetření polovině RAM paměti, je zbytečné ukládat proměnnou o které víme, že nepřesáhne hodnotu 255, často se používá pro uchování čísel pinů.

5.4.5 Double

Datový typ pro čísla s plovoucí desetinnou čárkou se často používají k aproximaci analogových a spojitých hodnot, protože mají větší rozlišení než celá čísla. Čísla s pohyblivou desetinou čárkou mohou být velká až 3,4028235E+38 a malá až -3,4028235E+38. Jsou uložena jako 32 bitů (4 bajty) informace. (Float - Arduino Reference, b.r.)

Proměnné Float a Double jsou u Arduina Nano naprosto stejné, rozdíl je, pokud se používá Arduino s jiným typem procesoru, například Arduino Due, které má 32bitový procesor SAM a větší kapacity pamětí. (Voda, 2017)

5.5 Modifikátory

V jazyce C existují modifikátory, které umožňují upravit základní chování datových typů nebo proměnných. Tyto modifikátory slouží k přidání dalších vlastností nebo omezení k datovým typům, které se používají v programu.

Často používané modifikátory jsou:

1. **Const** je použito před proměnnou v případě, kdy se hodnota proměnné nemění, tímto modifikátorem kompiler zjistí, že je zbytečné tuto proměnnou dávat do RAM paměti, namísto toho je proměnná uložena do FLASH paměti, pro ušetření. (Arduino Reference - Arduino Reference, b.r.)
2. **Unsigned** Tento modifikátor se používá pro celočíselné typy (např. int, short, long), aby indikoval, že proměnná nemůže mít zápornou hodnotu, a tudíž její rozsah začíná od 0, tím se zvýší rozsah maximální hodnoty o zápornou část proměnné. (Arduino Reference - Arduino Reference, b.r.)
3. **Volatile** Tento modifikátor indikuje, že hodnota proměnné může být změněna nějakým externím prostředím, mimo průběh aktuálního programu. To předchází kompilátoru, aby provedl optimalizace, které by mohly vést ke špatnému chování, pokud se hodnota změní externě. (Arduino Reference - Arduino Reference, b.r.)

5.6 Funkce

Funkce představují způsob, jak seskupit sadu programovacích příkazů do užitečného celku. To pomáhá rozdělit sketch na přehlednější části, což usnadňuje jeho používání. Například funkce, která obsahuje složitý algoritmus pro výpočet nějaké hodnoty je napsána jednou a nadále volána v případě potřeby. (Monk, 2018)

5.7 Knihovny

Knihovny v Arduino IDE jsou klíčovým nástrojem, který vám umožňuje rychle a efektivně vytvářet složitější projekty bez nutnosti psát vše od základu. Využití knihoven ušetří čas a usnadní proces vývoje, což je jedním z důvodů, proč je Arduino tak oblíbenou platformou pro tvůrce, studenty a vývojáře. Existuje obrovské množství knihoven, které pokrývají různé aspekty elektroniky, programování a komunikace. Mnoho knihoven je vyvíjeno komunitou Arduino, a to zahrnuje nejen oficiální knihovny, ale i knihovny od nezávislých tvůrců a nadšenců. Většina knihoven je zdarma k použití a je k dispozici online, což umožňuje snadnou instalaci a aktualizaci. (Ralph S Bacon, 2022)

Použití knihovny je velmi jednoduché. Po stažení a instalaci knihovny stačí vložit příslušný řádek na začátek vašeho programu: #include <nazev_knihovny.h>. Knihovna je například využita pro ovládání krokového motoru.

5.8 Kompilace

Kompilace se spojitostí (anglicky "Compile with Linked Dependencies") je nová funkce, která byla představena v Arduino IDE 2.0. Tato funkce je navržena tak, aby zefektivnila a urychlila proces kompilace a zároveň umožnila snazší správu knihoven

ve vašem projektu. Tradičně, v předchozích verzích Arduino IDE (jako například Arduino IDE 1.x), byly závislosti (knihovny) kompilovány znovu při každém sestavení projektu. To znamená, že pokud jste měli více knihoven použitých ve vašem projektu, bylo nutné kompilovat každou knihovnu zvlášť, i když se ve vašem kódu žádná knihovna nezměnila. To mohlo způsobit zbytečnou prodlevu a zpomalení procesu kompilace. (Sabarish.M, 2021)

Kompilace se spojitostí řeší tento problém tím, že se knihovny kompilují pouze tehdy, když dojde ke změně ve zdrojovém kódu samotné knihovny. Pokud se vaše kódová část nemění, nebo se nemění vlastnosti knihoven, které jste použili, kompilace knihoven se přeskočí. To znamená, že kompilace projektu je rychlejší, protože se znovu nekompilují nezměněné knihovny.

Tato funkce umožňuje také využití optimalizace při kompilaci knihoven, což vede ke snížení celkové velikosti binárního souboru vašeho projektu a také zlepšení výkonu.

Kompilace se spojitostí je tedy vítanou novinkou v Arduino IDE 2.0, která zvyšuje efektivitu a usnadňuje správu knihoven ve vašich projektech. Díky této funkci se proces vývoje stává rychlejším a pohodlnějším pro tvůrce a vývojáře na platformě Arduino.

5.9 Debouncing

Debouncing představuje výzvu, kterou se v oblasti elektroniky a programování setkáváme. Při stisknutí tlačítka není signál vždy bezchybný a nepřerušený, nýbrž může být zkreslený šumem a nežádoucím fyzickým rušením, což může vést k nežádoucím akcím a nepředvídatelnému chování, které může způsobovat nechtěné akce. Na toto existují dva hlavní přístupy, jak s tímto problémem nakládat. (Wright, 2022)

První je takzvané hardwarové ladění, kdy jsou do obvodu přidány další elektronické součástky jako například kondenzátory nebo rezistory, ty slouží jako vyhlazená a potlačení šumu ještě před přečtením a interpretováním mikrokontroleru. (Wright, 2022)

Druhý přístup je takzvaný softwarový, kde se problém řeší až v softwarové části systému, tedy pomocí nějakého kusu kódu. Řešení většinou spočívá v ignorování vstupů po zjištění první změny tlačítka, dále program počká daný čas, který je nutný pro ustálení kontaktu mezi vodiči a přečte znovu vstup pro potvrzení opravdového stisku tlačítka. Tímto dojde k „dvojí“ kontrole stisku, tato metoda je velmi často využívaná právě na Platformě Arduinu, protože nabízí jednoduché a rychlé řešení. (Debounce on a Pushbutton | Arduino Documentation, 2015)

5.10 Decoupling

Decoupling (také známé jako decoupling capacitance) je koncept v elektřině a elektronice, který se používá k minimalizaci nebo eliminaci rušení a nežádoucích vedlejších efektů v elektrických obvodech, zejména v digitálních integrovaných obvodech (IC) a desce plošných spojů (PCB). Když digitální obvod, například mikroprocesor nebo integrovaný obvod, pracuje, může vyvolat rychlé změny v proudu, což způsobuje fluktuace napájecího napětí a vytváří rušení. Toto rušení může mít

negativní vliv na chování dalších součástí v obvodu a vést k nespolehlivosti, šumům, nežádoucím signálům nebo poruchám. (Keim, b.r.)

Decoupling kapacity, obvykle reprezentované malými keramickými nebo tantalovými kondenzátory, jsou umístěny blízko napájecích pinů integrovaných obvodů. Tyto kondenzátory slouží jako rychlý zdroj dodávání proudu pro okamžité vyrovnání fluktuací a napěťových špiček v obvodu. Když digitální obvod spotřebuje proud, kondenzátory ho dodávají, což minimalizuje vliv fluktuací na zbylé součásti. Tímto způsobem decoupling kapacity přispívají k stabilnímu napájení obvodu a snižují nežádoucí rušení, což zlepšuje celkovou spolehlivost a výkon elektronického zařízení. Správné umístění a dimenzování decoupling kapacity jsou klíčové pro úspěšné minimalizování rušení a zajistí správnou funkci složitých elektronických obvodů. (Keim, b.r.)

6 Návrh systému - praktická část

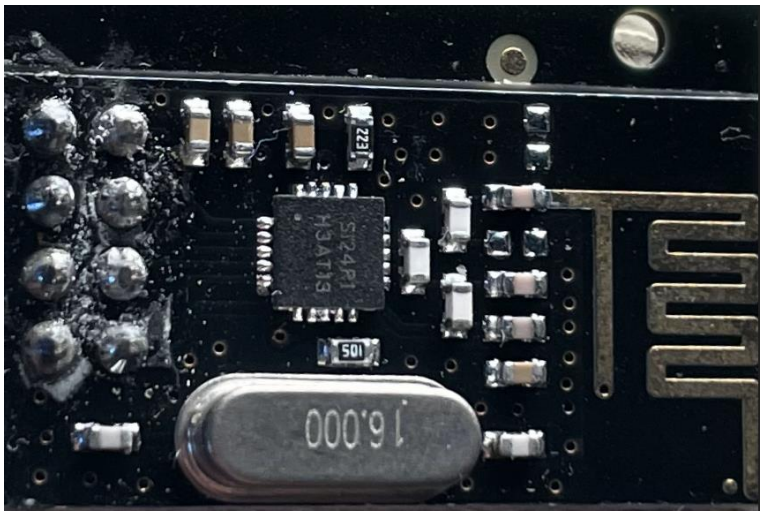
6.1 Výběr komponent

Důležité je si výběr vyhovujících komponentu pro náš systém, které budeme následně ovládat pomocí Arduina.

6.1.1 NRF24L01

Je bezdrátový modul, který umožňuje navázat bezdrátovou komunikaci mezi Arduino deskami, používá stejnou frekvenci jako běžný Wifi router, tudíž 2,4GHz. Tato frekvence ale neumožňuje komunikaci s běžnými síťovími prvky jako například zmiňovaný router nebo mobilními zařízeními, jen pracují na stejné frekvenci. To sebou přináší i nevýhody v prostředí, kde se nachází několik dalších zařízení používající stejnou frekvenci například v panelácích. Modul vyžaduje napájení 3,3V, kterým právě Arduino nano disponuje, datové piny jsou schopné pracovat s 5 V logikou Arduina a není nutno použít žádné převodníky. Vysílací výkon je možno měnit pomocí programu v arduinu, je možné nastavit 4 úrovně vysílání. Pro větší vysílací výkony je doporučeno použít externí zdroj napájení kombinací s kondenzátorem kvůli stabilitě přenosu. Teoreticky dosažitelný přenos je 100 m v ideálních podmínkách, což je pro mé použití dostačující. Modul se také vyznačuje svojí nízkou klidovou spotřebou. (M, b.r.)

Obrázek 5 deska plošných spojů rádiového modulu



Zdroj : vlastní

V tabulce níže je ukázáno, jak jsou piny modulu propojené s piny na Arduinu. Toto zapojení modulu je použito i na zapojení přijímače do druhého Arduina.

Tabulka 1 tabulka zapojení NRF24L01 do Arduina

Označení pinu	Zapojení na arduinu
GND	GND
VCC	3,3V
CE	D9
CS	D10
SCK	D13
MOSI	D11
MISO	D12
IRQ	Nepřipojen

Zdroj : vlastní

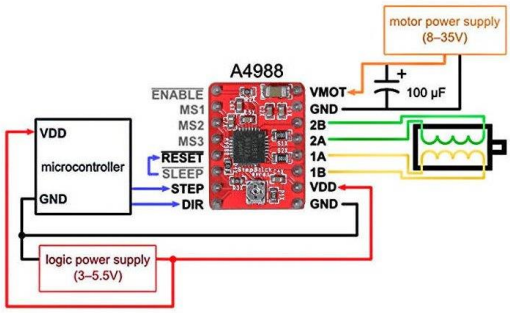
6.1.2 A4988

Je řadič pro krokové motory, který nabízí 5 různých módů pro ovládání krokového motoru, zároveň disponuje malým potenciometrem pro nastavení maximálního proudu pro motor, tepelnou ochranou a ochranou proti zkratu. Operativní napětí pro logickou část se pohybuje od 3 V do 5,5V, opět ideální rozhas pro použití s Arduinem, bez nutnosti nějaké konverze. Napájení pro samotný motor se pohybuje od 8 V do 35 V, přičemž maximální proud na fázi je 2 A s přídavným malým chladičem na čipu, bez chladiče okolo 1A, pro použitý motor dostatečné. (Allegro, 2014)

Na obrázku níže můžeme vidět schéma zapojení pinů. Pin VMOT a GND jsou piny, které se používají pro napájení a hlavní zdroj energie pro samotný motor, mezi tyto 2 piny se doporučuji nainstalovat kondenzátor pro tzv. “decoupling”. Piny 1A, 1B, 2A,

2B, jsou použity přímo na zapojení motoru, 1B a 1 A jsou piny pro jednu fázi motoru a piny 2A, 2B jsou použity pro druhou fázi motoru.

Obrázek 6 schéma zapojení řadiče, motoru a kontroléru



Zdroj : https://www.laskakit.cz/a4988-driver-pro-krokove-motory/

Pin VDD a GND jsou uzemnění a zdroj referenčního napětí pomocí kterého se nastavuje výkon motoru, tyto piny se připojí do Arduina. Pin RESET po aplikování napětí 5V slouží pro ignorování veškerého inputu od mikrokontroleru, pin sám o sobě není uzemněn, takže je potřeba ho uzemnit aby nedocházelo k milné interpretaci inputů, na obrázku je ukázáno propojení tohoto pinu s pinem SLEEP, to je z důvodu, že pin SLEEP má interní pulldwon rezistor a tudíž jeho hodnota je vždycky "vypnuto", propojením těchto dvou pinů se řadič aktivuje a SLEEP pin nikdy nemůže být použit k jeho funkci, která vypíná fáze motoru, tím pádem motorem neprochází proud a motor je "vypnutý". Tuto funkci jsem chtěl do svého systému integrovat, takže jsem SLEEP pin připojil k Arduinu na pin D3, tímto způsobem budu moci tento "mód" zapínat a vypínat. Pin RESET je připojen k pozitivnímu 5 V pinu na Arduinu, to znamená, že řadič nebude nikdy ignorovat signály od Arduina.

Pin STEP se používá pro krokování samotného motoru, jeden signál "zapnuto" řadič interpretuje a otočí motor o 1 krok v závislosti na módu krokování, takže tímto pinem posílá Arduino informace o počtu kroků a je připojen k Arduinu na pin D5. Pin DIR se používá na změnu orientace motoru a je připojen k Arduinu na pin D4.

Další zbývající piny jsou MS1, MS2, MS3, které slouží pro již zmíněný mód krokování, na obrázku se nachází logická tabulka pro nastavení krokovacích módů. Jednotlivé módy se nastavují pomocí přivádění 5 V na jednotlivé piny. Pro příklad, že bychom chtěli Half Step mód, tak přivedeme napětí na pin MS1 a ostatní piny necháme bez napětí, tímto si můžeme vybrat vyhovující mód pro nás. Pro moje použití chci používat mód první Full Step, takže piny mohu nechat bez napětí a nikam je nepřipojovat.

Obrázek 7 přehled krokových módů řadiče

Table 1: Microstepping Resolution Truth Table

MS1	MS2	MS3	Microstep Resolution	Excitation Mode
L	L	L	Full Step	2 Phase
H	L	L	Half Step	1-2 Phase
L	H	L	Quarter Step	W1-2 Phase
H	H	L	Eighth Step	2W1-2 Phase
H	H	H	Sixteenth Step	4W1-2 Phase

Zdroj : https://www.pololu.com/file/0J450/a4988_DMOS_microstepping_driver_with_translator.pdf

6.1.3 Krokový motor

Jako hlavní motor jsem zvolil Krokový motor NEMA 11 11HS2810 0,08Nm. Jedná se o bipolární krokový motor, který má pouze 2 fáze na rozdíl od unipolárních. Bipolární mají větší sílu a jsou náročnější na provoz, protože potřebují změnu směru proudu fázemi. O tuto komplikaci se postará řadič, který tyto změny provádí. Motor jsem také zvolil na základě velikosti, je potřeba co nejmenší a nejefektivnější motor, tento má rozměry 28x28mm.

Obrázek 8 Krokový motor

Zdroj : https://www.laskakit.cz/krokovy-motor-nema-11-11hs2810-0-08nm/

Motor disponuje úhlem jednoho kroku 1,8°. Když vydělíme jednu otáčku, tedy 360° velikostí jednoho kroku 1,8°, dostaneme rovných 200 kroků, takže aby motor udělal jednu celou otočku. je potřeba řadiči poslat 200 signálů pro otočení. Toto tvrzení platí, pokud máme nastavený první mód, tedy Full Step, při dalších módech jsou kroky dvojnásobné. V tabulce níže je zobrazeno kolik kroků v daném módu je potřeba pro jedno otočení. Vyšším počtem kroků získáme větší přesnost, ale za cenu síly motoru, proto jsem zvolil první mód, který je naprosto dostačující svojí přesností.

Nejlepší způsob, jak zvýšit přesnost a zachovat si zároveň i sílu, je použití nějakého převodu, to sice zredukuje rychlost provozu, ale velká rychlost není podmínečná pro funkci, použitý převod je 1:5, 10 zubů na pastorku a 50 zubů na hlavním ozubeném kole.

Tabulka 2 přehled počtů kroků pro vybraný mód

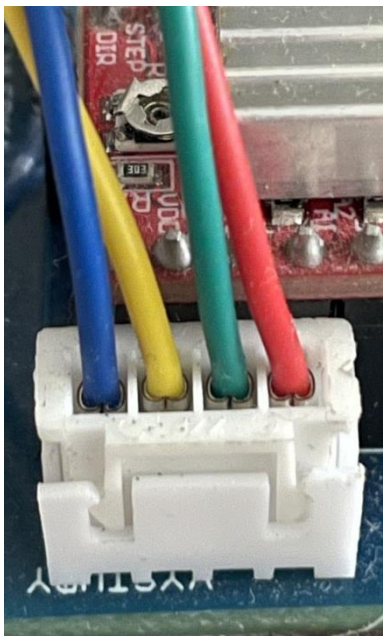
Krokový Mód	Počet kroků pro jedno otočení
Full Step	200

Half Step	400
Quarter Step	800
Eight Step	1600
Sixteenth Step	3200

Zdroj : vlastní

Další krok, potřebujeme zjistit pro správné zapojení motoru společné cívky motoru. Když se podíváme na obrázek 9.

Obrázek 9 Piny motoru



Zdroj : Vlastní

můžeme vidět, že z motoru vychází 4 vodiče, ale 2 vodiče jsou pro jednu cívku, takže potřebujeme zjisti, které jsou propojené. Toto můžeme zjistit díky propojování vodičů mezi sebou a následném otáčení hřídelí motoru. Pokud cítíme nějaký odpor hřídele, našli jsme společné vodiče pro jednu cívku. Takto vyzkoušíme veškeré vodiče a tím se ujistíme, že motor zapojíme správně do řadiče, protože každý výrobce může mít různě prohozené vodiče. U tohoto motoru to naštěstí vychází úplně stejně jako u daného řadiče a to v pořadí 2B, 2A, 1B, 1A. To znamená, že červený vodič je 2B a zelený je 2A, žlutý vodič 1B a modrý 1A.

Díky tomuto zjištění můžeme motor správně připojit k řadiči a následně pokračovat v testování.

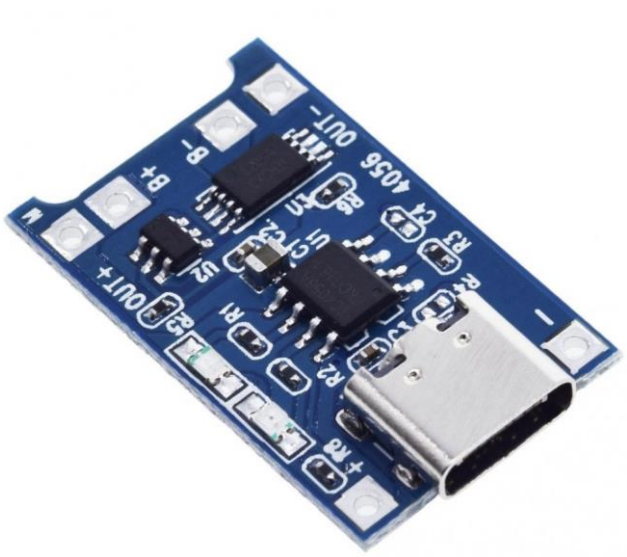
Tabulka 3 technické parametry motoru

Model: 11HS2810
Úhel kroku: 1,8°
Délka motoru: 28mm
Proud vinutím: 1A
Odpor vinutí: 2,7Ω
Indukčnost vinutí: 2,0mH
Přídržný moment: 80mN.m
Typ hřídele: D
Počet vodičů: 4
Délka vodičů s konektorem: 34cm
Váha: 94g

Zdroj : https://www.laskakit.cz/krokovy-motor-nema-11-11hs2810-0-08nm/

6.1.4 Modul pro správu baterie

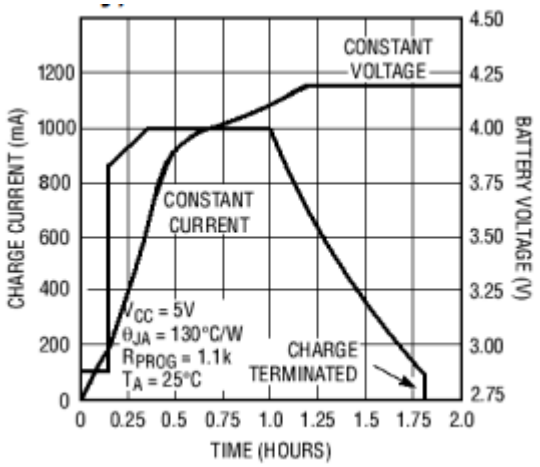
Obrázek 10 Modul pro správu baterie



Zdroj : https://pajenicko.cz/nabijecka-lipol-liion-clanku-tp4056-s-usb-c

Pro správu baterie jsem vybral modul TP4056 s USB-C (obrázek 11), je to modul pro nabíjení Li-Pol/Li-Ion článků, disponuje funkcí pro obnovu baterie. Pokud klesne napětí pod 2,5V, umí tzv. “oživit” baterii, kterou by klasická nabíječka už nemusela nabíjet. Je zde nastavitelný nabíjecí proud pomocí odporu na desce, také se zde nachází ochrana proti pod vybití baterie, které nenávratně zničí článek. Nabíjecí konektor je zde USB-C, takže nabízí moderní konektivitu a nabíjení ze všech powerbank. Má speciální nabíjecí cyklus pro nejšetrnější nabíjení, viz. obrázek dole.

Obrázek 11 cyklus nabíjení baterie



6.1.5 Baterie

Jako zdroj energie pro vysílač jsem zvolil baterii SONY VTC6, baterie 18650 30A 3000 mAh. Jedná se o Li-Ion článek, který je kompatibilní s nabíjecím modulem. Baterie je původně určená do elektronických cigaret, což znamená, že dokáže dodávat obrovský vybíjecí proud 15 A. Toto je pro Arduino zcela zbytečné, ale ničemu to nevadí. Pro chod Arduina je potřeba proud zhruba 0,04 A, takže baterie při kapacitě 3Ah dokáže napájet systém zhruba 65 hodin, to je pro mě naprosto dostačující.

6.1.6 Modul pro konverzi napětí z nabíjecího modulu (5V step up)

Další modul, který je potřeba, je modul pro konverzi a stabilizaci napětí na 5 V, potřebný pro chod Arduina. Modul pro správu baterie neposkytuje stabilní výstup 5 V, ale napětí podle nabití baterie, zhruba 3,2 V až 4,2 V. Tento modul tedy dává konstantní výstup 5V a dostatečný proud okolo 0,38 Ah.

6.1.7 Seznam komponentů s jejich cennou

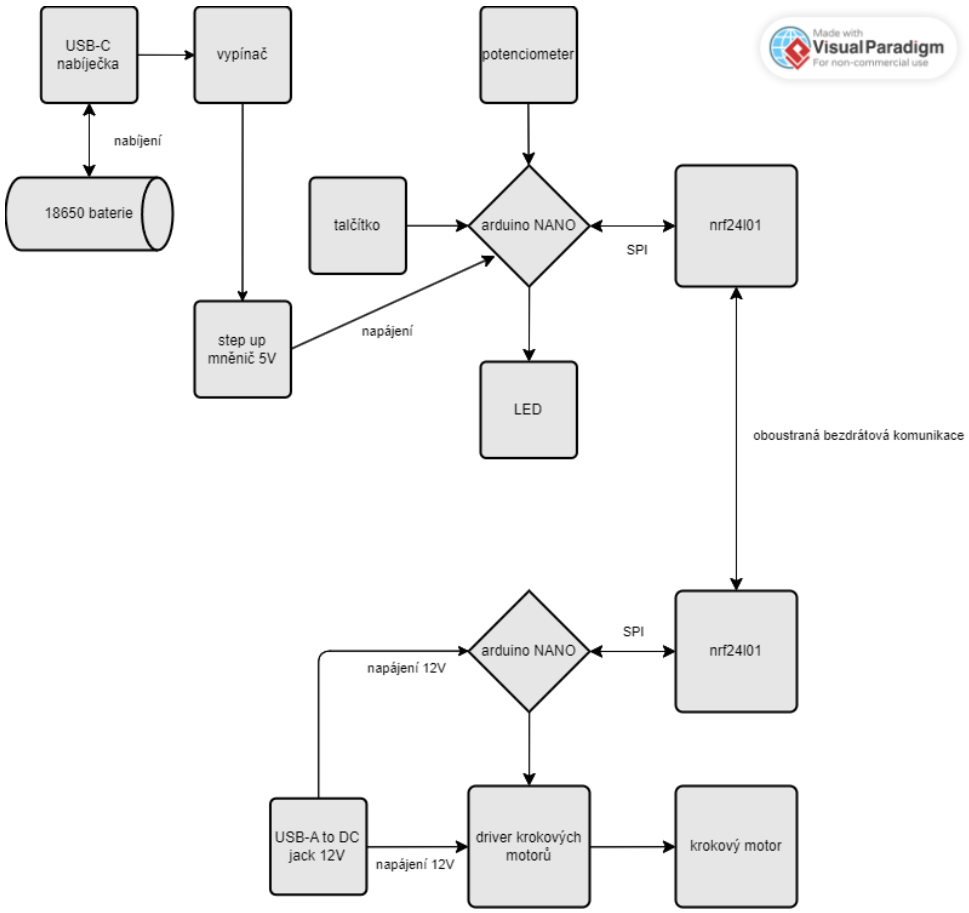
Tabulka 4 Seznam komponentů a jejich cen

Název komponentu	Počet kusů	Celková cena v Kč
Arduino Nano	2	200
A4988 řadič	1	25
100uf kondenzátor	1	3
10uf kondenzátor	2	6
220Ohm rezistor	2	2
Červená LED	1	1
Držák na baterii 18650	1	15
18650 baterie	1	140
Spínač kolébkový	1	10
NRF2401	2	25
Mikro spínač	1	2
TP4056 s USB-C	1	21
5V step up	1	24
Krokový motor	1	358
USB-A to DC jack 12V kabel	1	30
PCB tisk	1	70

10kΩ potenciometr	1	8
Nepájivé pole 400pinů	2	90
PCB	1	78
Ostatní malé komponenty (kabely, pájecí cín, konektory)		60
		1 168

6.2 Základní kompozice komponent

Obrázek 12 Kompozice modulů



Zdroj : vlastní

Na obrázku 12 je znázorněno, jaké komponenty a moduly jsou použity a jak spolu komunikují. Horní část obrázku zobrazuje vysílač a jeho jednotlivé komponenty. Je zde použitý modul TP4056 pro nabíjení a správu baterie typu 18650, dále se zde nachází vypínač pro zapnutí a vypnutí zařízení, který pomocí rozpojení kontaktů mezi modulem TP4056 a 5V step up měničem, sloužícím pro převod nižšího napětí z TP4056 modulu na 5V pro Arduino. Tlačítko je využito společně s potenciometrem pro interakci se zařízením a červené LED pro výstup uživatele. Dále je zde bezdrátový modul NRF2041 pro oboustrannou komunikaci mezi totožným modulem v druhém zařízení, zde se používá SPI protokol pro komunikaci s Arduinem a modulem, přijímač obsahuje krokový motor, který je připojen do řadiče krokových motorů pro jednodušší ovládání motoru pomocí Arduina, které posílá řadiči informace o tom, jakým směrem a kolik kroků má motor udělat, řadič tyto informace interpretuje a provede. Řadič potřebuje napětí 12 V, z

tohoto důvodu je zde přítomný napájecí kabel typu USB-a to DC jack 12 V, ten dodává napájení krokovému řadiči a zároveň Arduinu.

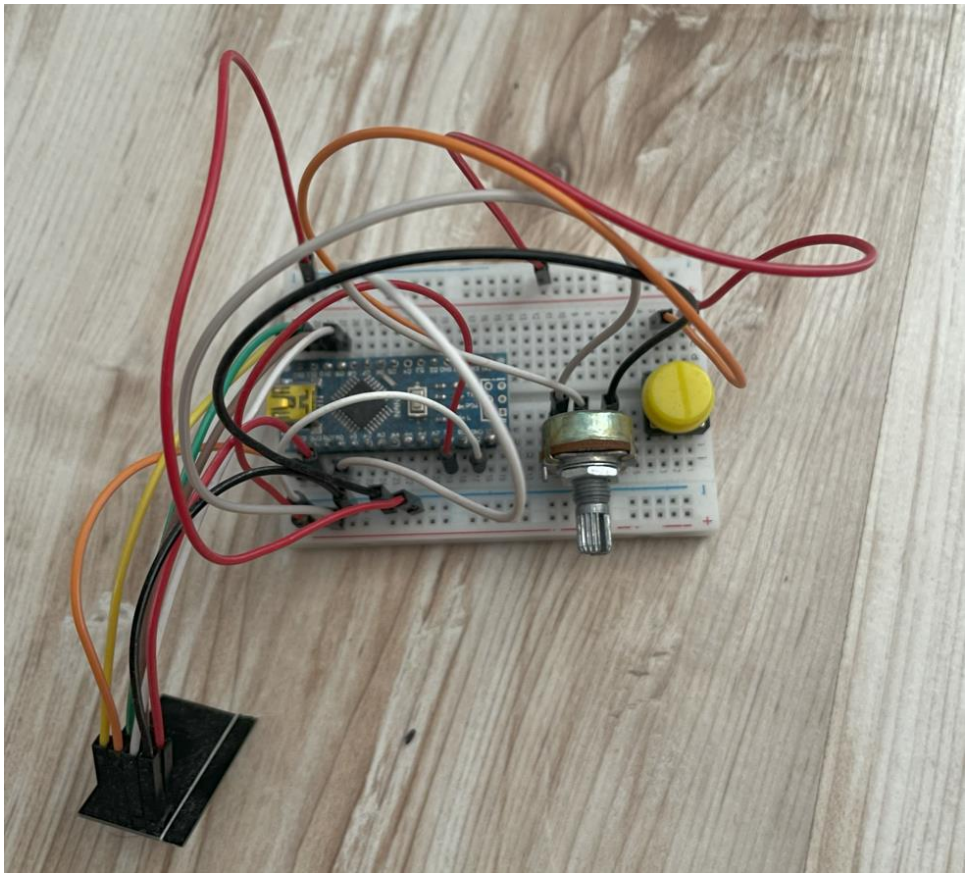
6.3 Vytvoření prototypu

Po výběru komponentů a navržení systému je dalším krokem spojit jednotlivé komponenty systému dohromady, to udělám pomocí breadboard prototypování, tedy propojení veškerých modulů mezi s sebou v nepájivém poli.

6.3.1 Prototyp vysílače

Na obrázku 14 je prototyp vysílače v nepájivém poli. V tomto stavu se provádí testování, Arduino se připojí k počítači pomocí USB, kde probíhá psaní testovacího kódu.

Obrázek 13 Prototyp vysílače

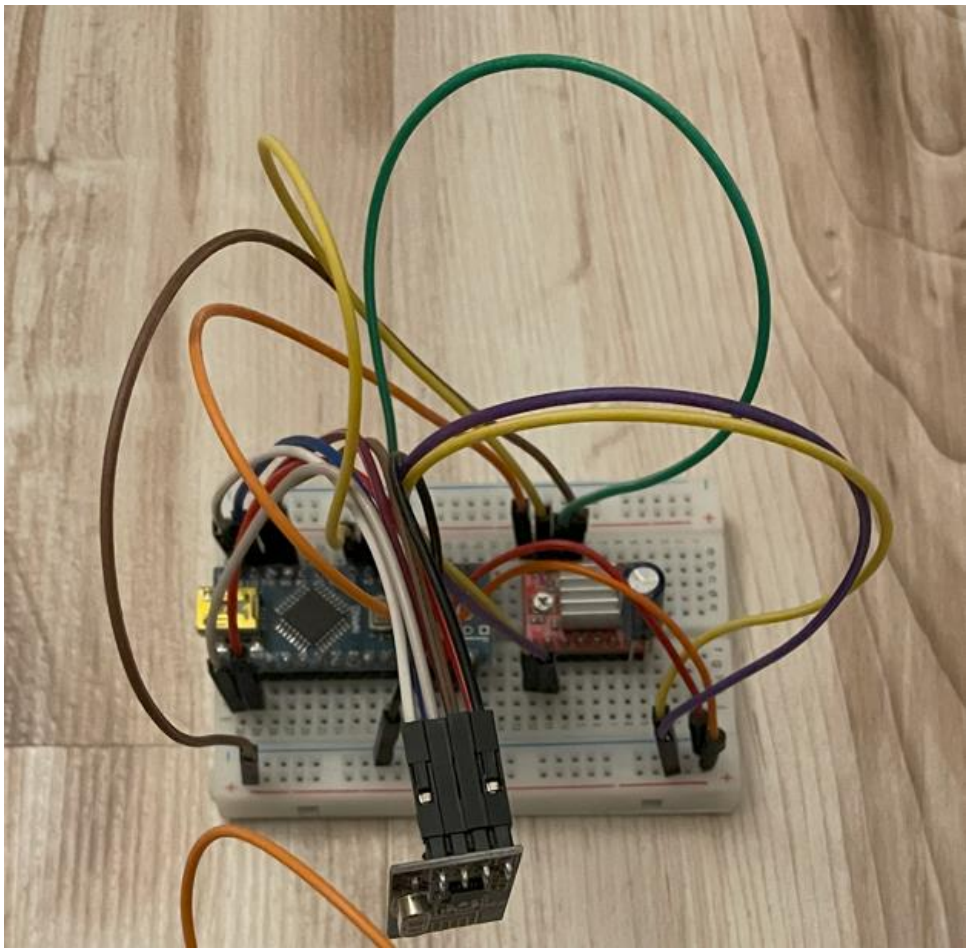


Zdroj: vlastní

6.3.2 Prototyp přijímače

Na obrázku 14 je prototyp přijímače v nepájivém poli.

Obrázek 14Prototyp přijímače



6.4 Program pro Arduino vysílač

Po vytvoření prototypu nastává nejdůležitější část a to je vytvoření kódu, který se následně zkompiluje a nahraje do paměti mikrokontroleru, bude potřeba napsat 2 různé kódy pro obě zařízení.

6.4.1 Čas

Mikrokontroler bude muset být schopen nějakým způsobem zaznamenávat a ukládat čas, toto je podstatné pro funkčnost vlastností jako je načítání dat z potenciometru, blikání LED a pouštění určitých funkcí, které závisí na čase, například budu chtít, aby mikrokontroler každý n-tý interval přečetl hodnotu senzorů nebo každý interval zapnul LED.

Pro toto použiju funkci `millis()`, tato funkce po zavolání vrátí hodnotu v milisekundách od napájení zařízení, tímto způsobem dokážeme měřit a pracovat s časem. Hlavní věc je správně zvolit proměnou pro uložení vrácené hodnoty, jelikož každá sekunda má 1 000 milisekund tato hodnota se lehce nasčítá a mohla by jednoduše přetéct, viz. proměnná typu int, která má 2 byte pro uložení hodnoty, tedy 2^{16} = 65 536

hodnot a zároveň použijeme modifikátor unsigned dostaneme zhruba pouhých 65,5 sekund, než proměnná přeteče a začne počítat znovu.

Řešení tohoto problému je použití proměnné typu unsigned long, která nabízí velikost 4 byte, tedy 2^{32} = 4,294,967,295 hodnot, to nám dává zhruba 49 dní, než proměnná dosáhne maximální hodnoty, takto dlouho zařízení ani běžet nebude, pokud by se tak stalo, tak díky tomu, jak je kód napsán by to ani nezpůsobilo žádnou závadu.

Nejdříve si vytvořím proměnné, viz. ukázka kódu 1, které budu využívat po celou dobu běhu programu. Nejdůležitější proměnná je actualTime, která slouží pro ukládání času aktuálního času v milisekundách od spuštění Arduina, nachází se v cyklu `loop()`, který běží donekonečna, to platí v obou zařízeních, můžeme vidět v ukázce kódu 2.

Proměnná startTime poskytuje paměť pro ukládání časového razítka, které je v kódu použito pro opakované pouštění cyklů. LastCalibTime slouží pro časové razítko u módu kalibrace, to rozeberu v kapitole kalibrace, to stejné platí pro proměnnou time. LastDeounceTime se používá u softwarového řešení tlačítka. Jak je vidět, tak všechny proměnné používají typ unsigned long, dohromady zabírají 20 Bytů v paměti RAM.

Ukázka kódu 1 časové proměnné

```
unsigned long startTime;   //timestamp pro čas
unsigned long actualTime;  //čas od startu zařízení
unsigned long lastCalibTime;  //timestamp poslední kalibrace
unsigned long lastDebounceTime;  //čas posledního debouncu
unsigned long time;  //timestamp pro kalibrační mód
```

Zdroj : vlastní

Ukázka kódu 2 loop cyklus

```
void loop() {
  actualTime = millis();
```

Zdroj : vlastní

6.4.2 Tlačítko

Tlačítko je způsob, kterým uživatel interaguje s vysílačem, tlačítko má funkci kalibračního módu, nejprve je nutno zprovoznit tlačítko, poté na něj můžeme navázat další funkce. První krok je nadefinování konstanty pinu, kam připojíme tlačítko, tedy na pin D2, který byl zvolen z důvodu integrovaného pullup rezistoru. Tato skutečnost nám dovolí ušetřit jeden rezistor.

Ukázka kódu 3 uložení pozice pinu tlačítka

```
const int button = 2;   //pin tlačítka
```

Zdroj : vlastní

Ke tlačítku je potřeba vytvořit ještě další 2 proměnné, do kterých budou ukládány pomocné hodnoty, bez kterých by nebylo možné vytvořit funkční tlačítko. Proměnné jsou typu bool, kterým je přiřazena hodnota true.

Ukázka kódu 4 pomocné proměnné pro funkci tlačítek

```
bool zmacknuto = true;   //pomocná proměnná pro tlačítko
```

```
bool lastState = true;   //pomocná proměnná pro stav tlačítka
```

Zdroj : vlastní

Vnitřní rezistor aktivujeme pomocí funkce `pinMode()` (ukázka kódu 5), kde funkci předáme číslo pinu a mód ve kterém má pin operovat. Tímto řádkem provedeme přiřazení funkce na daný pin, tuto skutečnost využíváme dále.

Ukázka kódu 5 nastavení pinu pro vstup

```
pinMode(button, INPUT_PULLUP);
```

Zdroj : vlastní

Momentálně při zmáčknutí tlačítka je pin propojen se zemí a při přečtení hodnoty pinu dostaneme hodnotu LOW, neboli vypnuto. Teď je potřeba napsat funkci, která dokáže správně vracet hodnoty, zda bylo tlačítko zmáčknuto, nezmáčknuto nebo zmáčknuto dlouze, dostáváme tedy 3 různé stavy, které mohou nastat. Celá funkce vypadá takto. (ukázka kódu 6)

Ukázka kódu 6 funkce pro tlačítko

```
int stavTlacitka() {
  zmacknuto = digitalRead(button);


  if (zmacknuto != lastState) {
    lastDebounceTime = actualTime;
    lastState = zmacknuto;
  }

  if ((actualTime - lastDebounceTime) > longPressTime && zmacknuto == false)
    //Serial.println("ZMACKLE-DLOUZE");
    return 2;
  }
  if ((actualTime - lastDebounceTime) > debounceTime && zmacknuto == false) {
    //Serial.println("ZMACKLE");
    return 1;
  }
  return 0;
}
```

Zdroj : vlastní

Funkce vrací hodnotu int a jmenuje se stavTlačítka. První řádek nám načte digitální hodnotu pinu tlačítka, která se ukládá do proměnné zmáčknuto, pokud je tlačítko zmáčknuté uloží se LOW, při zmáčknutí hodnota HIGH, hodnoty jsou prohozené právě kvůli pullup rezistoru, to my ale nevadí, jen si této skutečnosti musím být vědom. Dále je zde if podmínka (ukázka kódu 6), která porovnává hodnoty dvou proměnných. Logika za touto podmínkou spočívá v porovnání dvou pravd, ale pouze první může být stále měněna, pokud je první proměnná změněná dostaneme se dovnitř podmínky, kde se

provede uložení aktuálního času do proměnné lastDebounceTime, která slouží právě pro tzv. Debounce, poté do lastState uložíme stav tlačítka.

Dvě další if podmínky (ukázka kódu 6) dále slouží pro eliminaci nechtěných signálů, první if nám porovnává aktuální čas a čas, kdy bylo tlačítko naposledy stisknuto, tato doba musí být delší než předem určená proměnná, tady jsou jednotlivé hodnoty v milisekundách.

Ukázka kódu 7 vytvoření časových konstant

```
const int debounceTime = 30;      //čas potřebný pro debouncing tlačítka
const int longPressTime = 1500;    //čas pro registraci dlouhého stisku
```

Zdroj : vlastní

Z této části kódu (ukázka kódu 7) je vidět, že pro detekci stisku tlačítka musí být stisklé po dobu 30 milisekund pro registraci stisku, pro dlouhý stisk je registrace nastavená za dobu 1 500 milisekund, tedy 1,5 vteřiny. Funkce vrací hodnotu 0 při nedetekování žádného inputu, hodnotu 1 při stisku a hodnotu 2 při dlouhém stisku tlačítka, tyto informace jsou dále využity v jiných funkcích.

6.4.3 Potenciometr

Nejvíce používaná interakce se zařízením je právě pomocí potenciometru, pomocí jeho hodnoty je dále vypočítávána budoucí hodnota pro motor. K funkci samotného potenciometru stačí jedna proměnná pro uložení hodnoty, a ještě nějaké proměnné k fungování funkcí, které na něj navazují. Proměnná sensorValue se používá pro ukládání jednoho vzorku z potenciometru, jedná se o int, protože Arduino disponuje 10bitovým analog to digital konvertorem, takže hodnoty jsou celá čísla s rozsahem 0 až 1023. Ostatní proměnné jsou typu double, protože se u nich může objevit desetinná hodnota. (ukázka kódu 8)

Ukázka kódu 8 proměnné k potenciometru

```
int sensorValue;
double prevValue = analogRead(A0);
double sum = 0;    //pomocná proměnná pro součet hodnot
double prumer;    //průměr hodnost sum
```

Zdroj : vlastní

Další dvě proměnné (ukázka kódu 9) jsou využívány pro funkci `pridejHodnotu()`, velikost PoleHodnot určuje velikost vzorků pro průměr hodnot z potenciometru. Na základě hodnoty přiřazené této proměnné se vytvoří pole int o velikosti hodnoty této proměnné.

Ukázka kódu 9 vytvoření pole pro průměr hodnot potenciometru

```
const byte velikostPoleHodnot = 25;  //pocet samplu pro vyhodnocení aktuální
hodnoty potenciometru
int values[velikostPoleHodnot];    //pole hodnot pro potenciometr
```

Zdroj : vlastní

Dále si definujeme, jak bude probíhat vzorkování hodnot z potenciometru, to probíhá pomocí už zmiňovaného času, to se v tomto případě využívá pro vzorkování

každou druhou milisekundu. Dále je zavolána funkce `pridejHodnotu()`, s argumentem aktuálně přečtené hodnoty z potenciometru. (ukázka kódu 10)

Ukázka kódu 10 čtení hodnoty potenciometru

```
if(actualTime % 2 == 0) {
sensorValue = analogRead(A0);
pridejHodnotu(sensorValue);
prumer = spocitejPrumer();
}
```

Zdroj : vlastní

Funkce `pridejHodnotu()` přidává parametr do pole hodnot poslední pozici v poli a zároveň posune všechny hodnoty o mínus jednu pozici, poslední hodnota v poli v tomto případě tedy 24. se přesune na pozici 23. A hodnota na pozici 0 se vymaže. Na přepis všech hodnot je potřeba 25*2 milisekundy + samotný čas na provedení operací, zhruba tedy 60 milisekund, to nám vytváří druh vyhlazování, dále je s tímto polem potřeba pracovat dále, o to se stará další funkce v pořadí. (ukázka kódu 11)

Ukázka kódu 11 funkce pro přidání hodnoty do pole

```
void pridejHodnotu(int hodnota) {
  for (int i = 0; i < velikostPoleHodnot - 1; i++) {
    values[i] = values[i + 1];
  }
  values[velikostPoleHodnot - 1] = hodnota;
}
```

Zdroj : vlastní

Jde o funkci `spocitejPrumer()`, která nám vrací aritmetický průměr ze součtu hodnot pole. To je docíleno pomocí lokálně vytvořené proměnné sum, do které je pomocí for cyklu, který sčítá hodnotu po hodnotě z daného pole a následně tento součet vydělí počtem míst v poli. Tímto získáme aritmetický průměr, který je dynamický, to znamená, že není potřeba přečtení všech 25 vzorků pro změnu výsledného výsledku. Například současná hodnota všech hodnot v poli bude číslo 100, pro které platí, že výsledná hodnota bude 100*25/25 = 100, po iteraci cyklu se změní načtená hodnota na 1000, to nám dává výsledek (100*24 + 1000)/25 = 136. Na tomto příkladu je demonstrována funkčnost, která spočívá ve vyhlazeném přechodu mezi výslednou hodnotou, to všechno bez výrazného zpoždění motoru, protože na výslednou změnu hodnoty ze 100 na 1000 dojde za 50 milisekund, za tuto dobu motor stejně nestihne takovou rychlost replikovat, zjednodušeně algoritmus je rychlejší než fyzický limit motoru. (ukázka kódu 12)

Ukázka kódu 12 funkce pro výpočet průměru z pole

```
double spocitejPrumer() {
  int sum = 0;
  for (int i = 0; i < velikostPoleHodnot; i++) {
    sum += values[i];
  }
  return sum / velikostPoleHodnot;
}
```

Zdroj : vlastní

Mapování hodnoty

Výsledná hodnota se používá dále, v hlavním loop cyklu se nachází tato podmínka (ukázka kódu 13), která slouží pro kontrolu proměnné průměr, v podmínce se kontroluje hodnota průměru, což je výsledná hodnota potenciometru, v podmínce se kontroluje, zdali se proměnná průměr liší od předešlé hodnoty, která má název prevValue, je zde tolerance v rozsahu +0.40 až -0.40, tyto hodnoty byli vhodně zvoleny po analyzaci grafu této určité hodnoty, pokud se s potenciometrem nehýbe hodnota jeho průměru se nevychýlí v tomto rozmezí, to zaručí, že podmínka je splněna pouze v případě pohybu potenciometru, podmínka při splnění zavolá funkci `mapValues()`.

Ukázka kódu 13 podmínka pro zavolání mapovací funkce

```
if (průměr > (prevValue + 0.40) || průměr < (prevValue - 0.40)) {
      mapValues();
      }
```

Zdroj : vlastní

Funkce mapValues() obsahuje pouze 2 řádky kódu (ukázka kódu 14). Nachází se zde proměnná kroky, její hodnota se rovná výsledku zavolané funkce `map()`, ta vrací jednu číselnou hodnotu, což je výpočet finálního počtu kroků, který je následně posílán druhému zařízení.

Ukázka kódu 14 funkce pro mapování hodnot potenciometru na kroky pro motor

```
vod mapValues(){
  kroky = map(průměr, 0, 1023, minPos, maxPos);
  provalu = prumer;
}
```

Zdroj : vlastní

6.4.4 Kalibrace

Kalibrace slouží pro nastavení rozsahu kroků, po který se motor pohybuje, rozsah kroků je 0 až 1 000, to vychází na 5 otáček motoru. Důležité proměnné pro fungování této části kódu jsou 4 proměnné A, B, minPos, maxPos. Proměnné A je přiřazena hodnota 0, naopak proměnné B je přiřazena hodnota 1 000, to je právě počet kroků motoru. Další dvě proměnné jsou pomocné proměnné pro další fungování kalibrace. (ukázka kódu 15)

Ukázka kódu 15 deklarace proměnných pro kroky

```
int A = 0, B = 1000, minPos, maxPos;
bolo indexBodu = false;
bolo kalibrace = false;
```

Zdroj : vlastní

Kalibrace je podmíněná předchozí funkcí `stavTlacitka()` (ukázka kódu 17), je zde if podmínka s dalšími dvěmi větvemi pomocí else if, první if podmínka kontroluje, zda bylo tlačítko zmáčknuto dlouze, pokud je tato podmínka pravdivá, vnoří se kód dovnitř a zavolá funkci `kalibraceRozsahu()`, tato funkce obsahuje další kód, který je podmíněný pro následující else if podmínku, která kontroluje, zda je proměnná kalibrace pravdivá, to je právě změněno v zavolané funkci.

V této funkci `kalibraceRozsahu()`se nastaví hodnota proměnné minPos na hodnotu 0 a maxPos na hodnotu 1 000, to způsobí přepsání dvou hlavních hodnot, které měli jiné hodnoty podle provedené předchozí kalibrace. Tyto dvě hodnoty se promítnou ve funkci, kde se mapuje hodnota potenciometru na počet kroků. Další proměnná kalibrace se nastaví na true, to znamená, že zařízení vstoupilo do módu kalibrace. Proměnná lastCalibTime je nastavena na hodnotu aktuálního času, ta je použita pro limitaci doby, po kterou je možno zařízení kalibrovat. Na dalším řádku je zavolána funkce mapValues(). (ukázka kódu 16)

Ukázka kódu 16 pomocná funkce pro kalibrování bodu A a B

```
void kalibraceRozsahu() {
  minPos = 0;
  maxPos = 1000;
  kalibrace = true;
  lastCalibTime = actualTime;
  mapValues();
}
```

Zdroj : vlastní

Proměnná kalibrace je rovna hodnotě true, to znamená, že else if podmínka bude splněna každý cyklus, po dobu hodnoty proměnné kalibrace na true.

V cyklu se nachází další if podmínka (ukázka kódu 17), která kontroluje obyčejný stisk tlačítka, po spuštění kalibrace se hýbe potenciometrem pro nastavení správné polohy motoru, po stisku tlačítka se podmínka if splní, tím se do proměnné A uloží aktuální počet kroků, tudíž pozice prvního bodu, dále se pomocná proměnná indexBodu, díky které je možno vkročit do další větve else if podmínky, nastaví na hodnotu true a do proměnné time, sloužící jako časové razítko se uloží aktuální čas.

Poslední else if podmínka obsahuje 3 podmínky, které musí být současně splněny, pro vnoření. První je kontrola stisku tlačítka, ta kontroluje opět obyčejný stisk tlačítka, druhý kontroluje, zdali byla splněna předchozí else if podmínka, třetí kontroluje, zdali časový rozestup mezi prvním stiskem tlačítka, tedy první else if podmínkou, je větší než nastavený interval.

Po splnění všech 3 podmínek je nastaven bod druhý pomocí prvního řádku, kde se proměnná B rovná proměnné kroky, tím se uloží druhý počet kroků pro druhý bod, dále se proměnná kalibrace nastaví na false, to znamená ukončení cyklu pro kalibraci, dále je proměnná indexBodu taktéž nastavená na false. Do proměnných minPos a maxPos jsou uloženy hodnoty, který byly zvoleny uživatelem a nadále se tyto hodnoty pomocí knihovny EEPROM a funkce `put()` uloží do paměti pro trvalé zachování těchto zvolených hodnot, tedy po restartu zařízení budou hodnoty z paměti načteny a opět použity pro pohodlí uživatele. (ukázka kódu 17)

Ukázka kódu 17 kód pro kalibraci

```
if (stavTlacitka() == 2) {
    kalibraceRozsahu();
  } else if (kalibrace == true) {
    if (stavTlacitka() == 1 && indexBodu == false) {
```

```
      A = kroky;
      indexBodu = true;
      time = actualTime;
    } else if (stavTlacitka() == 1 && indexBodu == true && actualTime -
time > interval) {
      B = kroky;
      kalibrace = false;
      indexBodu = false;
      minPos = A;
      maxPos = B;
      EEPROM.put(0, minPos);
      EEPROM.put(2, maxPos);
    }
}
```

Zdroj : vlastní

6.4.5 Blikání LED

LED slouží pro interakci s uživatelem a poskytuje informaci o tom, zda se zařízení nachází v módu kalibrace, pokud tomu tak je, LED bliká, v opačném případě je zhasnutá. Je zde podmínka if (ukázka kódu 19), která kontroluje, zda uplynul čas ve velikosti proměnné interval, pokud tento čas uběhl, je podmínka splněna. Pin ledky je definován pomocí proměnné led (ukázka kódu 18).

Ukázka kódu 18 deklarace proměnné pro pin LED

```
const int led = 4;     //pin ledky
```

Zdroj : vlastní

Proměnná startTime se nastaví na aktuální čas, pro obnovení intervalu cyklu. Je zde další podmínka if a else, která kontroluje, zda je zařízení v módu kalibrace, pokud ano je zavolána funkce `ledBlinking()`, která se stará o blikání LED, v opačném případě je zavolána funkce, která LED vypne. (ukázka kódu 19)

Ukázka kódu 19 podmínka pro spouštění kódu každý interval

```
 if (actualTime - startTime > interval) {//kazdy jeden interval se spusti
    startTime = actualTime;
    if(kalibrace == true){
      ledBlinking();
    }else{
      digitalWrite(led, LOW);
    }
  }
```

Zdroj : vlastní

Funkce `ledBlinking()` načítá hodnotu z led pinu, který může mít hodnoty HIGH nebo LOW, tato hodnota je uložena do lokální proměnné value, následně je pomocí if else podmínky zkontrolováno, zda je LED rozsvícená nebo zhasnutá, následně buď LED vypne nebo zapne, tím se docílí blikání LED. (ukázka kódu 20)

Ukázka kódu 20 funkce pro blikání LED

```
void ledBlinking(){
  bool value = digitalRead(led);
  if(value == HIGH){
     digitalWrite(led, LOW);
  }else{
    digitalWrite(led, HIGH);
    }
}
```

Zdroj : vlastní

6.4.6 Bezdrátová komunikace

Je potřeba komunikace mezi oběma zařízeními, k tomu použiji Arduino knihovnu RF24, která je vytvořená pro jednoduší ovládání bezdrátového modulu NRF24L01, vytvoříme na prvním řádku vytvořím objekt s názvem radio (ukázka kódu 21), s parametry čísla pinů, kde je rádio připojeno. Dále je proměnná rslt, je to pomocná proměnná, která se používá dále v kódu pro validaci přijmutí zprávy.

Dále je potřeba přiřadit adresy jednotlivých „pipe“, které slouží při inicializaci rádia pro správné rozeznání protějšího modulu a naopak, adresy slouží pro správné fungování oboustranné komunikace pomocí přepínání mezi módy pro poslouchání a vysílání. (ukázka kódu 21)

Ukázka kódu 21 vytvoření instance objektu pro rádio

```
RF24 radio(9, 10);  // CE, CSN
bool rslt;  //bool pro potvrzení ackowledge
const byte addresses[][6] = { "00001", "00002" };  //adresy pro bezdrátov
ou komunikaci
```

Zdroj : vlastní

Dále je potřeba definovat proměnné pro příjem a odesílání dat k tomu použiji kolekci různých proměnných s názvem struct (ukázka kódu 22). Jsou zde vytvořeny dvě struktury, dataSend a dataRecieved. Obsahují dvě proměnné, KROKY pro ukládání počtu kroků a kanal pro uložení kanálu, tato proměnná se ale nevyužívá, je zde přítomna pouze pro možné rozšíření pro přepínání frekvence.

Ukázka kódu 22 návrh struktur pro rádio

```
struct dataSend{                //struktura pro posílání dat bezdrátově
  unsigned int KROKY : 10;      //proměnná kroků
  unsigned int kanal : 7;      //proměnná kanálů
};

struct dataRecieved{    //struktura pro příjem dat
  unsigned int KROKYrecieved : 10;
  unsigned int kanalrecieved : 7;
};
```

Zdroj : vlastní

Po definici struktur je potřeba vytvořit jednotlivé struktury a přiřadit jim jména, tím se zapíšou do RAM paměti (ukázka kódu 23). Struktura pro posílání dat má jméno hodnoty, pro příjem hodnotyPrijmuty.

Ukázka kódu 23 vytvoření struktur

```
dataSend hodnoty;       //vytvoření struktury
dataRecieved hodnotyPrijmuty;     //vytvoření struktury
```

Zdroj : vlastní

6.4.7 Funkce setup

Funkce setup() je spouštěna pouze jednou před zacyklením, proto se využívá pro inicializaci proměnných a objektů (ukázka kódu 24). Prvně je potřeba inicializovat samotný bezdrátový model, k tomu je využita funkce `begin()`, dále je modulu nastaven kanál, na kterém bude operovat, v tomto případě je to poslední možný kanál 124.

Funkce `radio.openWritingPipe(addresses[0])` slouží k otevření „potrubí“ pro odesílání dat. Argumentem této funkce je adresa, která určuje, na kterou adresu budou data odesílána, konkrétně adresu 00001. V tomto případě se jedná o multireciever mode, takže je potřeba otevřít ještě jeden kanál pro příjem dat.

Funkce `radio.openReadingPipe(1, addresses[1])` slouží k opaku a to otevření kanálu pro přijimání dat. Argumenty této funkce jsou index „potrubí“ a adresa. Každé „potrubí“ na přijímací straně má svůj vlastní index, který slouží k identifikaci potrubí. V tomto případě je použit index 1 a obsahuje adresu pro přijímání dat z rádiového modulu na straně přijímače, tedy 00002. V druhém zařízení, tedy přijímači jsou adresy prohozeny naopak.

Funkce `setPALevel()` nastavuje vysílací výkon pomocí jednoho argumentu, tady jsem použil nejvyšší možný vysílací výkon, pomocí argumentu RF24_PA_MAX, dále je potřeba nastavit přenosovou rychlost, která je v tomto případě nastavená na 250 kilobitů za sekundu, pomocí funkce `setDataRate()` to je rovno 31 250 Bytů za sekundu, posílaná struktura zabírá 4 Byte, po vydělení teoretická rychlost přesunu dat je okolo 7000, to je naprosto naddimenzovaná hodnota, to znamená, že odezva mezi zařízeními je co nejnižší.

Dále se pomocí funkce `setRetries()`, která má dva argumenty, první je pro nastavení času mezi jednotlivými pokusy o poslání payloadu, zde je hodnota 5, to nám po výpočtu (5 * 250 + 250) dává 1500us tedy 1,5 ms, další parametr je počet pokusů pro poslání daného payloadu, zde jsem zvolil taky hodnotu 5, takže maximální doba pro poslání jednoho payloadu je 4 * 1,5 ms, tedy 6 ms, to v případě, že ani jeden pokus nebude úspěšný, pokud tato situace nastane, tak to prakticky není poznat, protože se posílání nachází v cyklu. Tímto je celé rádio inicializováno.

Pomocí knihovny EEPROM a funkce `get()` je z této paměti načteno do proměnných minPos a maxPos pozice, která tam byla dříve zapsána, pomocí funkce pro kalibrování, tímto se zachovává předchozí nastavení dvou pozic bodů od uživatele.

Další dva řádky slouží pro nastavení pinů vstupu a výstupu, pomocí funkce `pinMode()`, kde první argument je přímo číslo pinu nebo číslo uložené v proměnné, to jsem použil já, druhý parametr nám určuje jak se bude pin chovat, v prvním případě díky argumentu INPUT_PULLUP, který nám říká, že se pin bude chovat jako vstup

a zároveň se použije interní pullup rezistor, takže pokud pin není uzemněn, je jeho hodnota rovna HIGH.

Poslední řádek přiřazuje pevnou hodnotu pro používaný kanál, který používají obě zařízení. (ukázka kódu 24)

Ukázka kódu 24 příkazy v begin funkci

```
radio.begin();
radio.setChannel(124);
radio.openWritingPipe(addresses[0]);
radio.openReadingPipe(1, addresses[1]);
radio.setPALevel(RF24_PA_MAX);
radio.setDataRate(RF24_250KBPS);
radio.setRetries(5, 5);
EEPROM.get(0, minPos);
EEPROM.get(2, maxPos);
pinMode(button, INPUT_PULLUP);
pinMode(led, OUTPUT);
hodnoty.kanal = 124;
```

Zdroj : vlastní

6.4.8 Posílání dat

Důležitou vlastností je společná komunikace, posílání dat probíhá pomocí funkce `posliData()`, první je potřeba modul přepnout do módu, kdy je schopen posílat data, protože pokud nic neposílá, je v módu poslechu, ten se ukončí pomocí funkce `stopListening()`, následně je modul schopen data posílat, na dalším řádku je aktualizován počet kroků v posílané struktuře, proměnná rslt zde slouží pouze pro debugging, pokud byla data úspěšně odeslána, bude hodnota true, k samotnému přenosu dat je použita funkce `radio.write()`, která se používá k odesílání dat. Prvním argumentem je adresa datové struktury, proto symbol &, v druhém argumentu je potřeba znát počet bajtů, které datová struktura zabírá, k tomu je využita funkce `sizeof()`, která vrací přesně tuto hodnotu. (ukázka kódu 25)

Ukázka kódu 25 funkce pro posílání hodnot

```
void posliData() {

  radio.stopListening();
  hodnoty.KROKY = kroky;
  rslt = false;
  rslt = radio.write(&hodnoty, sizeof(hodnoty));
 }
```

Zdroj : vlastní

Dalším krokem je zjistit, kdy a proč se má funkce na posílání dat zavolat, k tomu je využita if podmínka (ukázka kódu 26), která se nachází opět v hlavním loopu, takže se provádí neustále dokola, kontroluje zda se aktuální počet kroků nerovná předchozímu počtu kroků, pokud je podmínka splněna funkce `posliData()` je zavolána, hned následně se předchozí počet kroků přepíše na aktuální počet a dále se čeká na změny.

Ukázka kódu 26 podmínka pro kontrolu změny kroků

```
    if (kroky != prevKroky) {
      posliData();
      prevKroky = kroky;
    }
  }
```
Zdroj : vlastní

Je zde přítomna ještě jedna if podmínka (ukázka kódu 27), která slouží jako pojistka, kontroluje se zde proměnná realnaPozice, která obsahuje hodnotu přijatou z druhého zařízení, pokud se tato hodnota neslučuje s reálnou hodnotou kroků na vysílači je splněna první část podmínky, druhá část slouží pro maximální rychlost posílání 50 ms, pomocí kontroly zda zbytek po dělení 50 je roven nule, při splnění je opět zavolána funkce `posliData()`.

Ukázka kódu 27 podmínka pro zpětnou komunikaci

```
if (realnaPozice != kroky && actualTime %50 == 0) {

      posliData();
    }
```
Zdroj : vlastní

6.4.9 Přijímání dat

Data je potřeba taktéž přijímat k tomu slouží funkce `getData()` (ukázka kódu 28), tato funkce se nachází v hlavním loop cyklu, takže je volána pokaždé, kdy se kód iteruje, prvně se pomocí funkce `startListening()` přepne do módu poslechu, následně se v if podmínce zavolá funkce `available()`, která vrací hodnotu true, pokud byla modulu doručená nějaká data, ta jsou uložená v bufferu samotného modulu, to zároveň umožňuje asynchronně čekat na přijetí dat a není tedy potřeba zastavovat exekuci ostatního kódu, po splnění podmínky se zavolá funkce `read()`, která načte data z bufferu a uloží je do struktury hodnotyPrijmuty, dále se do proměnné realnaPozice načtou nové kroky ze struktury.

Ukázka kódu 28 funkce pro příjem dat

```
void getData(){
    radio.startListening();
  if(radio.available()) {
    radio.read(&hodnotyPrijmuty, sizeof(hodnotyPrijmuty));
    realnaPozice = hodnotyPrijmuty.KROKYrecieved;
  }
}
```
Zdroj : vlastní

6.5 Program pro arduino přijímač

Přijímač využívá stejný princip pro evidenci času a spouštění časových událostí. Důležité proměnné s jejich komentáři pro správnou funkci.

Ukázka kódu 29 vytvoření časových konstant

```
unsigned long lastActionTime; //timestamp pro ukládání poslední akce
const int interval = 1000;  //čas pro jeden interval v milisekundách
const int timeToSend = 50; //ms čas pro kterém se pošlou data
```

```
const unsigned int sleepTime = 15000; //ms čas po kterém se vypne motor
```
Zdroj : vlastní

6.5.1 Start systému

Veškerý funkční kód se nachází v této if podmínce (ukázka kódu 30), to zapříčiní, že systém není funkční dokud podmínka není splněna.

Ukázka kódu 30 hlavní podmínka pro spuštění systému

```
if(readyToOperate){}
```
Zdroj : vlastní

Podmínka je splněna ve funkci `getData()`, ta je totožná s funkcí u vysílače, ale obsahuje navíc řádky kódu právě pro tuto podmínku, jak je vidět na prvním řádku kódu, pokud modul zachytí signál z vysílače nastaví podmínku na true a tím je spuštěn celý kód systému.

V další podmínce (ukázka kódu 31) se kontroluje, zda je motor ve spánkovém módu, pokud ano, nastává jeho probuzení pomocí funkce `digitalWrite()`, s argumentem číslo sleep pinu u řadiče, který se nastaví na hodnotu HIGH, tedy zapnuto, další řádek kódu vypne proměnnou pro kontrolu spánkového módu, to se děje vždy. Dále se pomocí funkce z knihovny `moveTo()`, kde je předán argument v podobě počtu kroků, které má motor vykonat. Tato hodnota je získána z vysílače.

Poslední řádek obsahuje proměnnou lastActionTime, která slouží jako časové razítko pro zaznamenání poslední akce motoru, ta je dále využita pro uspání motoru.

Ukázka kódu 31 kód pro kontrolu spánkového módu

```
readyToOperate = true;
    if(sleep){
    digitalWrite(sleepPin,HIGH);
    }
    sleep = false;
    stepper.moveTo(kroky);
    lastActionTime = actualTime;
    }
```
Zdroj : vlastní

6.5.2 Krokový motor

Nejdůležitější část přijímače je samotný pohyb ozubených kol, to zajišťuje krokový motor a jeho řadič, pomocí knihovny AccelStepper, první krok je nadefinování proměnných potřebných ke správné funkci. První řádek je proměnná, do které je uloženo číslo pinu pro uspání motoru. Další řádek je proměnná sleep, sloužící pro evidenci módu uspání motoru, s přiřazenou hodnotou true. Dále jsou vytvořeny dvě proměnné motorAction a saveKroky, obě jako pomocné proměnné u dalších funkcí.

Dále se vytvoří objekt AccelStepper s názvem stepper, jedná se o instanci objektu pro krokový motor se třemi argumenty, první argument s hodnotou 1 znamená, že používám řadič ke krokovému motoru, druhý je pin pro posílání kroků, třetí pin pro směr otáčení.

Vytvoří se proměnná kroky pro ukládání příchozích počtu kroků, dále dvě konstanty, které ukládají výše zmíněná čísla pinů pro kroky a směr. (ukázka kódu 32)

Ukázka kódu 32 vytvoření proměnných pro funkci motoru

```
const int sleepPin = 3;  //číslo pinu pro spánek
bool sleep = true;

bool motorAction;
bool saveKroky;

AccelStepper stepper = AccelStepper(1, steps, dir);
unsigned int kroky; //proměná pro počet kroků pro motor

const int steps = 5;  //číslo pinu pro kroky
const int dir = 4;   //číslo pinu pro směr
```

Zdroj : vlastní

6.5.3 Pohyb motoru

Pohyb motoru zajišťuje funkce `runMotor()`, je zde if podmínka (ukázka kódu 33), která kontroluje zda se motor má pohnout, pokud tato hodnota není rovna 0, je tato podmínka splněna a kód pokračuje vnořením, první řádek nastaví motoru rychlost krokování pomocí funkce `setSpeed()`, s argumentem 200, dále se pomocí funkce `runSpeedToPosition()` rozhýbe samotný motor a začne se pohybovat na potřebnou pozici danou rychlostí. Další řádek kódu zapíše časové razítko do proměnné lastActionTime, dále využité pro uspání motoru. Pomocné proměnné motorAction a saveKroky jsou nastaveny na true. (Ukázka kódu 32)

Ukázka kódu 33 funkce pro pohyb motoru

```
void runMotor() {
  if (stepper.distanceToGo() != 0) {
    stepper.setSpeed(200);
    stepper.runSpeedToPosition();
    lastActionTime = actualTime;
    motorAction = true;
    saveKroky = true;
  }
}
```

Zdroj : vlastní

6.5.4 Uspání motoru

Obrázek 15 Měření spotřeby

Zdroj : vlastní

Po provedeném měření viz. obrázek jsem změřil spotřebu přijímače, kde většina je spotřeba právě motoru, činí 1,21 A při napětí 5 V z powerbanky, což se rovná výkonu 6 W. To by znamenalo možný provoz okolo tří hodin v závislosti na kapacitě powerbanky. Já jsem konkrétně použil 19 Wh, to by znamenalo provoz zhruba tři hodiny i při nepoužívání zařízení, protože motor zůstane pod proudem bez ohledu na to, jestli zařízení bylo po nějakou dobu nepoužíváno. Toto se opraví pomocí if podmínky, která se nachází v hlavním loop cyklu.

V podmínce (ukázka kódu 34) se kontrolují dvě skutečnosti, první, zda byl čas od posledního pohybu motoru větší než nastavený limit pomocí proměnné sleepTime, s konkrétní hodnotou 15 000 milisekund, druhá podmínka kontroluje, jestli byl motor už uspán nebo ne. Uvnitř podmínky se pomocí funkce `digitalWrite()` motor uspí a pomocná hodnota spánku je nastavená na true, tedy po 15ti sekundách nečinnosti se motor vypne, tím se ušetří skoro veškerá spotřeba, takže zařízení dokáže fungovat ze stejného zdroje mnohem déle.

Ukázka kódu 34 kód pro spánkový mód

```
if (actualTime - lastActionTime > sleepTime && sleep == false) {
  digitalWrite(sleepPin, LOW);
  sleep = true;
}
```

Zdroj : vlastní

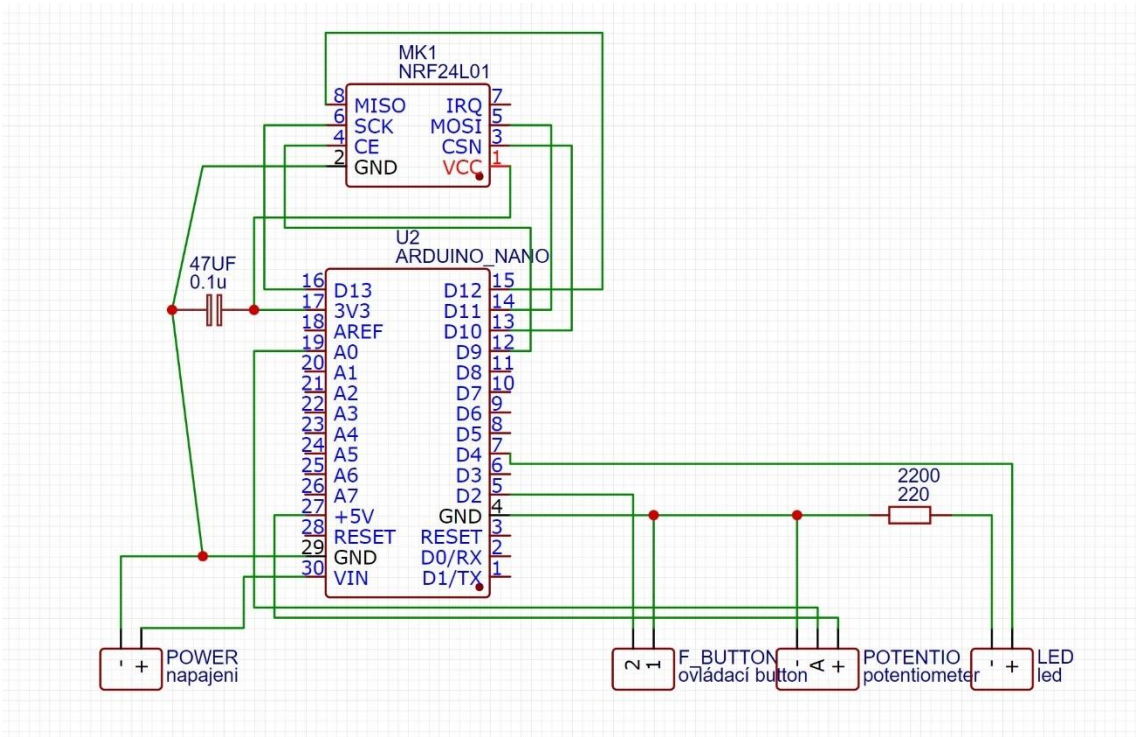
6.6 EasyEDA

Dalším krokem je navržení a vytvoření základní desky pro komponenty a moduly.

6.6.1 Návrh schématu

Prvním krokem je navržení schématu, to znamená propojit vše správně, na obrázku níže je vidět kompletní návrh zapojení pro vysílač.

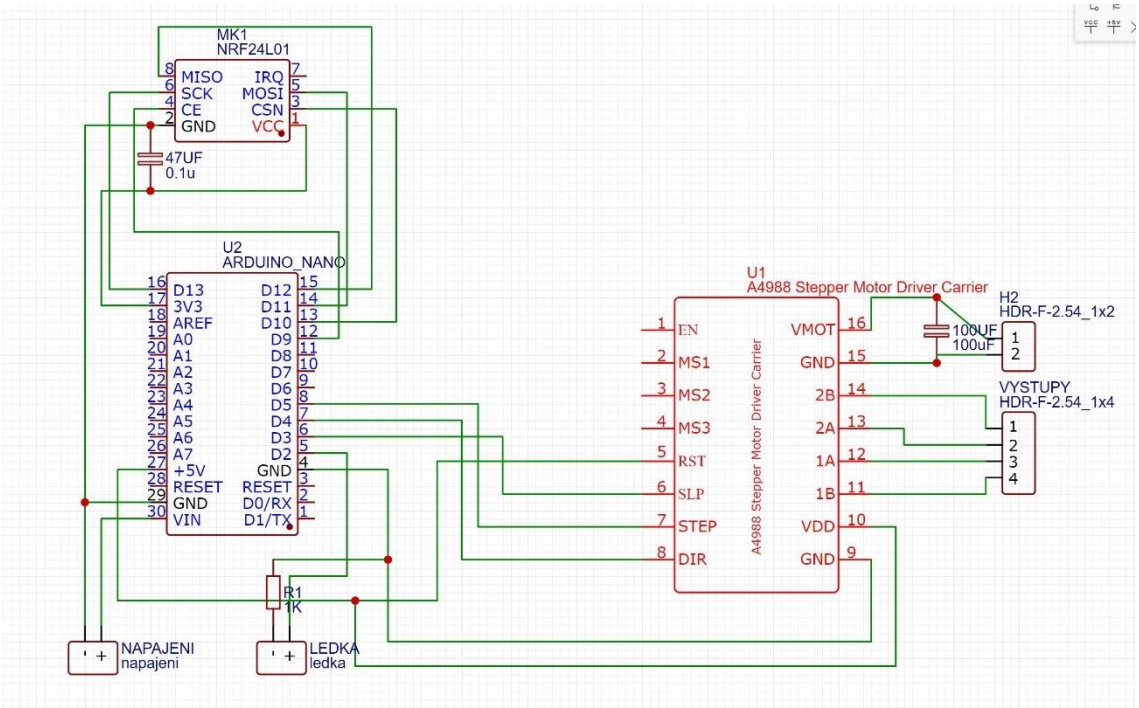
Obrázek 16 Schéma vysílače



Zdroj : vlastní

To stejné udělám i pro přijímač.

Obrázek 17 Schéma přijímače

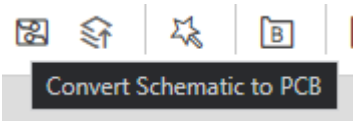


Zdroj : vlastní

6.6.2 Vytvoření desky plošných spojů (PCB)

Dále je potřeba vytvořit samotnou základní desku a připravit ji pro tisk, easyEDA nabízí jednoduché tlačítko pro automatické převedení schématu do návrhu desky

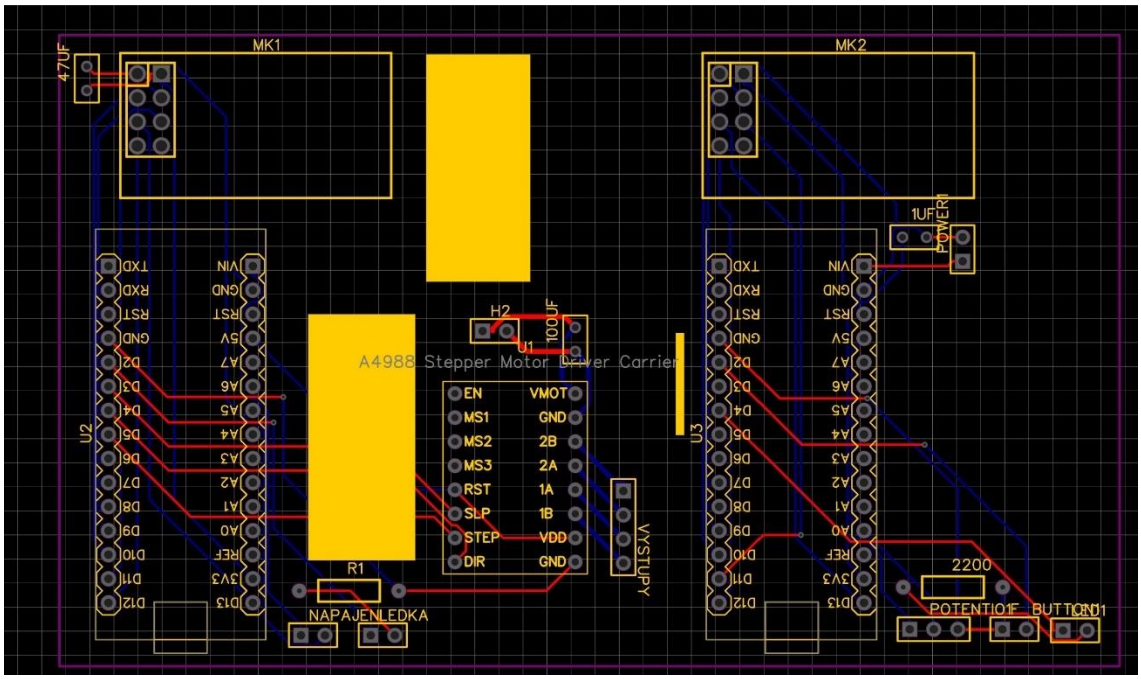
Obrázek 18 Převod Schématu na PCB



Zdroj : vlastní

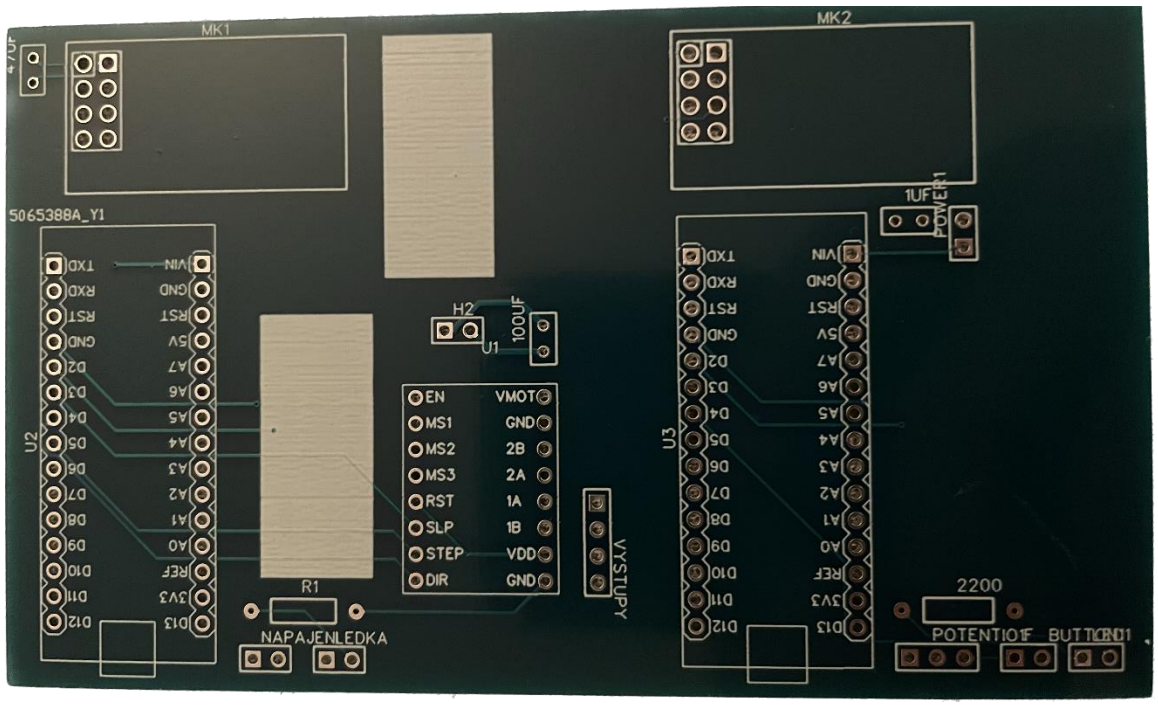
Toto tlačítko ze schématu vytvoří nový dokument PCB (obrázek 19), kde se přesunou veškeré navržené komponenty a dále je potřeba rozmístit jednotlivé komponenty, případně něco doupravit. Já jsem zvýšil šířku vodiče pro napájení řadiče, protože tam teče největší proud. Takto vypadá finální návrh desky, obě zařízení jsem přesunul na jednu základní desku, kterou následně rozdělím na dvě, protože je to takto levnější než objednávat dvě rozdílné.

Obrázek 19 Obě zařízení na jedné PCB



Zdroj : vlastní

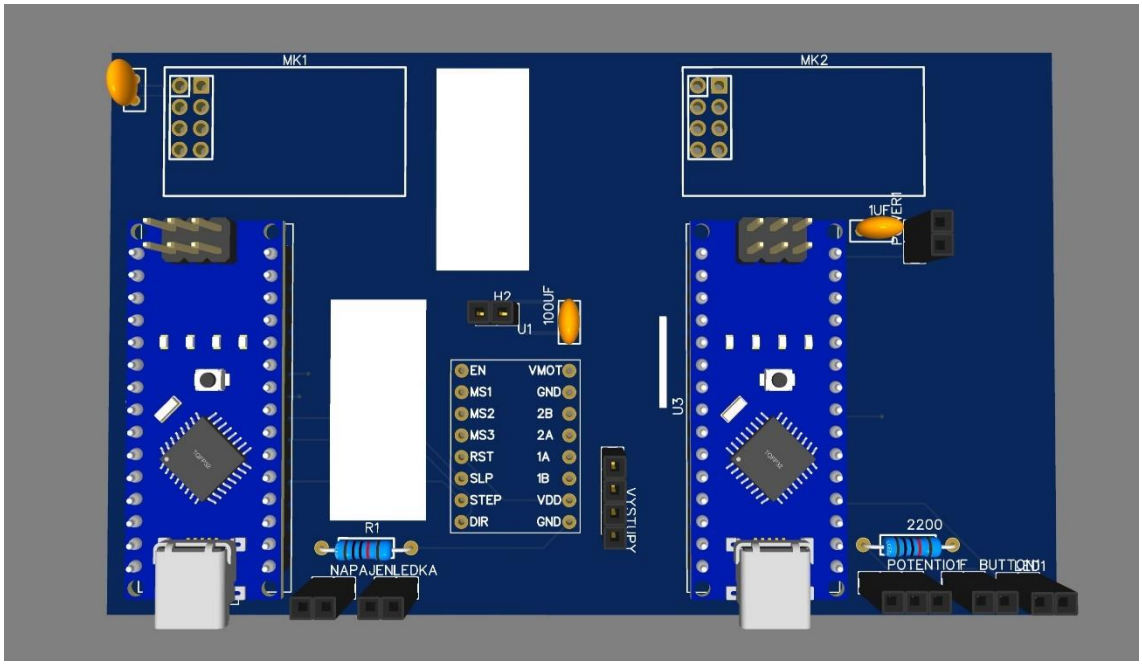
Obrázek 20 Finální PCB



Zdroj : vlastní

Zde ještě pohled na 3D verzi desky.

Obrázek 19 3D verze PCB



Zdroj : vlastní

Dále je potřeba desku zhotovit, to zařídí online služba JLCPCB.com, která desku kompletně vyrobí a doručí. Platil jsem 78 Kč za pět kusů této desky, s využitím jejich kupónu pro nové uživatele. Fotka již hotové a fyzické základní desky, následně na ně pomocí páječky připájím konektory a komponenty (obrázek 20).

6.7 Blender

Další část je potřeba vytvořit potřebné součástky k fungování zařízení, nějaký obal pro zařízení a ozubená kola, která budu pohybovat objektivem.

6.7.1 Modelování

Veškeré části jsem vymodeloval v programu Blender, (obrázek 23) veškerých modelů, které jsem vytvořil. Většina modelů je vytvořena z obyčejných krychlí a válců, které jsem mezi s sebou odečítal nebo přičítal pomocí modifieru boolean. Ozubená kola jsou vytvořena pomocí zdarma pluginu Precision Gears Lite.

Obrázek 20 Parametry ozubeného kola

Zdroj : Vlastní

Zde jsou vidět parametry (obrázek 22), které se zadávají do pluginu, který následně vymodeluje ozubené kolo, toto nastavení je konkrétně pro velké ozubené kolo, které pohání pastorek, který je na hřídeli motoru. Nastavil jsem parametry 50 zubů, modul zubů je 0,8 pro zachování kompatibility s ostatními zařízeními. Tloušťka 5 mm a pressure angle je taky standartních 20 stupňů.

Obrázek 21 Pohled z vrvhu na všechny 3D modely

Zdroj : Vlastní

6.7.2 3D tisk

Veškeré modely byly následně vytištěné na základní 3D tiskárně Creality ENDER 3 V2, kombinací filamentu PLA a PETG. Níže jsou nějaké vybrané vytisklé součástky.

Obrázek 24 Obal pro přijímač

Zdroj : Vlastní

Obrázek 23 Násada na motor

Zdroj : Vlastní

Obrázek 22 Pastorek

Zdroj : Vlastní

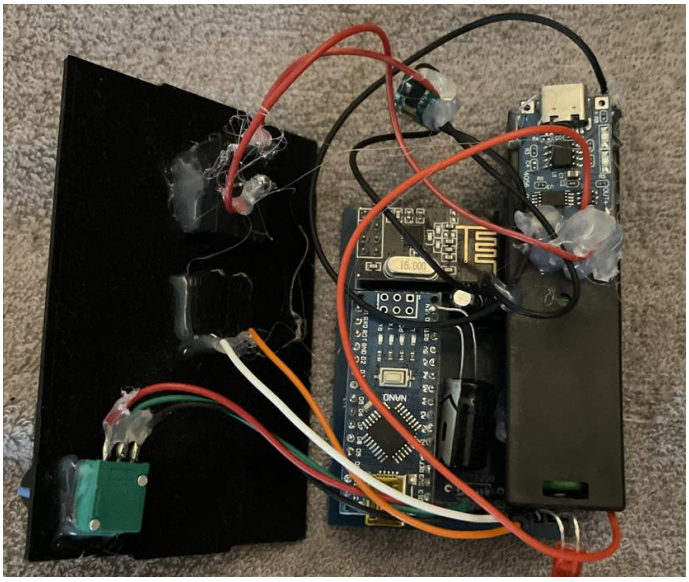
6.8 Výsledná zařízení

Takto vypadají finální zařízení bez vložení do obalu. Finální produkt je v přílohách.

Obrázek 25 přijímač

Zdroj : vlastní

Obrázek 26 vysílač



Zdroj : vlastní

7 Závěr

V této práci se podařilo vytvořit dvě separátní zařízení, které spolu navzájem bezdrátově komunikují a společně ovládají ostření u objektivů fotoaparátu. Vyvíjeno na platformě Arduino Nano pomocí vývojového prostředí Arduino IDE 2.0, kde byl napsán veškerý kód pro chod systémů. Byly zde například napsány algoritmy pro správnou funkci a odladění nevyžádaných stisků tlačítek, vyhlazení potenciometru.

Jako postup k dosažení cíle bylo použito prototypování pomocí nepájivého pole, kde se prvně sestavily oba prototypy zařízení a po testování proběhl návrh desky plošných spojů v online aplikaci easyEDA.com. Po navržení schématu byla vytvořena deska plošných spojů, výroba byla objednána na stránce JLCPCB.COM.

Pro potřebné součástky byl použit modelovací 3D open source program Blender, kde byli navrženy veškeré součástky, které byly následně pomocí 3D tisku zhotoveny s kombinací PLA a PETG filamentů. To všechno bylo zhotoveno za 1 168 Kč, cíl práce lze považovat za splněný.

I. Summary and keywords

This work focuses on designing and programming a product, that wirelessly controls the focus ring of a camera lens. The product is based on an Arduino nano programmable circuit board incorporating an ATmega328P microcontroller, that is used for all calculations, stores the whole code on a flash memory, controlling the inputs and outputs of an electronic components specifically a potentiometer, a buttons, a step motor, a motor driver and a wireless modules. The final product consists of two smaller devices, the transmitter, which is held by the user for sending information and the receiver controlling the motor. Both devices are powered by USB power banks. Programming and debugging take place in Arduino IDE. Designing and modelling parts e.g. spur gears and the case uses Blender. Final outcome aims to make cheaper, but equally efficient product as the commercial solution for this product.

Keywords: Arduino based device, wireless communication, 3D designing, microcontroller programming

II. Seznam literatury

Arduino Data Types [Online]. Retrieved July 24, 2023, from https://www.javatpoint.com/arduino-data-types

Arduino IDE 2 [Online]. (2023). Retrieved July 24, 2023, from https://docs.arduino.cc/software/ide-v2

M, L. Arduino WiFi modul NRF24L01 [Online]. *Drátek*. Retrieved from https://navody.dratek.cz/navody-k-produktum/arduino-wifi-modul-nrf24l01.html

Allegro. (2014). *A4988*. 115 Northeast Cutoff Worcester, Massachusetts 01615-0036 U.S.A. Retrieved from https://www.allegromicro.com/-/media/files/datasheets/a4988-datasheet.pdf

LEDme s. r. o. (2021). CO JE TO LED DIODA? [Online]. Retrieved from https://ledme.cz/textove-novinky/clanky/co-je-LED-dioda

Keim, R. Coupling and Decoupling [Online]. Retrieved July 25, 2023, from https://eepower.com/capacitor-guide/applications/coupling-and-decoupling/#

Debounce on a Pushbutton [Online]. (2015). Retrieved July 24, 2023, from https://docs.arduino.cc/built-in-examples/digital/Debounce

Wright, G. (2022). Debouncing [Online]. Retrieved from https://www.techtarget.com/whatis/definition/debouncing

Float [Online]. (2021). Retrieved July 24, 2023, from https://www.arduino.cc/reference/en/language/variables/data-types/float/

Language Reference [Online]. Retrieved July 26, 2023, from https://www.arduino.cc/reference/en/#variables

Monk, S. (2011). *Programming Arduino Getting Started with Sketches* [Online] (2nd ed.). New York: McGraw Hill Professional. Retrieved from http://www.sel.eesc.usp.br/jcarmo/pdfs/Arduino_SimonMonk_2011.pdf

Monk, S. (2018). *Programming Arduino Next Steps: Going Further with sketches* (2nd ed.). New York: McGraw-Hill Education.

Ralph S Bacon. (2022, November 11). #BB11 Create an Arduino Library😨 - A Real World Example (Easy) [Video file]. Retrieved from https://www.youtube.com/watch?v=mhYuiveM544

Sabarish.M. (2021, December 14). What happens in Arduino IDE when we click Compile? - SABARISH.M - Medium. Medium. Retrieved from https://medium.com

Voda, Z. (2017). *Průvodce světem Arduina* [Online] (Vydání druhé). Bučovice: Martin Stříž. Retrieved from https://www.robotikabrno.cz/docs/arduino/Pr%C5%AFvodce-sv%C4%9Btem-Arduina-CZ.pdf

Christoffersen, J. (2015). Switch Bounce and How to Deal with It [Online]. *All About Circuits*, 1. Retrieved from https://www.allaboutcircuits.com/technical-articles/switch-bounce-how-to-deal-with-it/

Horowitz, P., & Hill, W. (1989). *The art of electronics* (2nd ed). New York: Cambridge University Press.

Prada, J. (2022). What Is a Follow Focus System and How to Use It [Online]. *Artist Blog*, 1. Retrieved from https://artlist.io/blog/follow-focus/

Reichl, T. (2015). Základní typy baterií a správné nabíjení [Online]. Retrieved July 25, 2023, from https://oenergetice.cz/akumulace-energie/zakladni-typy-baterii-a-spravne-nabijeni#comments

Michalec, L. (2021). Základy digitálních potenciometrů a jejich použití [Online]. Retrieved from https://vyvoj.hw.cz/zaklady-digitalnich-potenciometru-a-jak-je-pouzivat.html

III. Seznam obrázků

IV. Seznam tabulek

V. Seznam ukázek kódu

VI. Seznam příloh

Příloha 1 Nainstalovaný systém na fotoaparátu

Zdroj : vlastní

Příloha 2 Kontakt ozubených kol zařízení a objektivu.

Zdroj : vlastní