STŘEDNÍ PRŮMYSLOVÁ ŠKOLA

MLADÁ BOLESLAV



**DLOUHODOBÁ PRAKTICKÁ**

**MATURITNÍ PRÁCE**

Lukáš Hruška

Mladá Boleslav 2016

STŘEDNÍ PRŮMYSLOVÁ ŠKOLA

MLADÁ BOLESLAV

Vytvoření řídícího softwaru kvadrokoptéry

**DLOUHODOBÁ PRAKTICKÁ**

**MATURITNÍ PRÁCE**

Autor: Lukáš Hruška

Studijní obor: 18-20-M/01 Informační technologie

Vedoucí práce: Radek Zvěřina

Mladá Boleslav 2016

## Prohlášení

Prohlašuji, že jsem svou dlouhodobou praktickou maturitní práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW, atd.) uvedené v přiloženém seznamu.

Nemám závažný důvod proti zpřístupňování této závěrečné práce v souladu se zákonem č. 121/2000 Sb., o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) v platném znění.

V Mladé Boleslavi dne podpis:

OBSAH

## Strana

## Zadání 6

## Všeobecné informace 7

2.1 Kvadrokoptéra 7

2.2 Využití kvadrokoptéry 7

2.3 Raspberry Pi 2 7

2.4 Arduino Uno 7

2.5 ADXL345 a L3G4200D 8

2.6 Ostatní HW 8

2.7 C++ 8

2.8 AVR C 9

2.9 Objektově orientované programování 9

2.10 Javascript a JQuery 9

2.11 WebSocket 10

2.12 I2C 10

2.13 PWM 10

2.14 paměť EEPROM 10

2.15 Komplementární filtr 11

2.16 PID algoritmus 11

2.17 NetBeans IDE 11

2.18 Arduino IDE 12

## Úvod do problému 12

3.1 Použitý HW 12

3.2 Rozvrstvení řídícího systému 12

3.3 Stabilizace 12

3.4 Detekce překážky 12

## Řešení problému 13

4.1 Postup při výběru HW 13

4.2 Rozvrstvení řídícího systému 14

4.3 Postup řešení programu 14

4.4 Finální běh řídícího systému 17

4.5 Vyskytnuté problémy a jejich řešení 21

## Závěr, vyhodnocení 22

## Zdroje 22

## Seznam příloh 22

# Zadání praktické maturitní zkoušky

## Název: Vytvoření řídícího softwaru kvadrokoptéry

## Jméno žáka: Lukáš Hruška, 4. Bi

## Úloha:

Vytvořte software pro navrženou kvadrokoptéru s programovaným RISC procesorem (vývojový kit Arduino UNO), který se bude starat o stabilitu stroje, bezpečnost, pohyb v okolí a celkové ovládání. Pro práci použijte jakékoli ovládací rozhraní (wifi, bluetooth apod.)

## Proveďte:

1. Popište použité prostředky, materiál a součásti použité pro stavbu kvadrokoptéry (technické specifikace apod.). Popište softwarové prostředky použité pro tvorbu řídícího softwaru.
2. Navrhněte a naprogramujte potřebné funkce pro udržení kvadrokoptéry ve vzduchu
3. Navrhněte a naprogramujte potřebné funkce pro kompenzaci větru a menších nárazů
4. Navrhněte a naprogramujte potřebné funkce pro udržení stability za letu s podvěšeným závažím o menší hmotnosti.
5. Navrhněte a naprogramujte potřebné funkce pro automatické zabránění nárazu do zdi a překážek i přes příkaz ovladače.
6. Vytvořte zajímavé a ergonomické ovládací rozhraní.

## Časový harmonogram:

Do 30. 10. 2015 – bod 1  
Do 30. 11. 2015 – bod 2, 3  
Do 4. 1. 2016 – bod 4  
Do 29. 2. 2016 – bod 5, 6  
Do 30. 3. 2016 – bod 7

## Literatura:

Veškeré dostupné informace na internetu.

## Zadavatel: SPŠ Mladá Boleslav

## Vedoucí práce: Radek Zvěřina

Schváleno předmětovým týmem IT dne: 2. 10. 2015

Schválil ředitel školy dne:

Ing. Martin Kubát

1. **Všeobecné informace**

## Kvadrokoptéra

Pod termínem kvadrokoptéra se v dnešní době skrývá typ bezpilotního letounu (tzv. dronu), který používá k letu 4 motory. Základní části kvadrokoptéry jsou rám se čtyřmi rameny, motory, regulátory otáček, poté jedna baterie a řídící jednotka. Řídící jednotka většinou obsahuje integrovaný gyroskop, akcelerometr, přijímač radiových vln z ovladače a předprogramovaný mikrokontrolér s řídícím programem. Většina kvadrokoptér je ovládána klasickým ovladačem pro běžné vrtulníky. V dnešní době se však vyvíjí lepší kvadrokoptéry s možností buď ovládání přes chytrá zařízení, nebo dokonce i se schopností autonomního letu.

## Využití kvadrokoptéry

Využití kvadrokoptér je opravdu široké, proto je nyní o ně takový zájem a jejich vývoj tak rychlý. Například firma Amazon používá v USA (samozřejmě s omezením) kvadrokoptéry k rozvážení menších produktů po okolí do 30 minut. Dále se kvadrokoptéry začínají používat v armádě buď jako sledovací zařízení, nebo i dokonce jako zbraň. Ale samozřejmě je zde i využití pro běžné lidi, například jako kamera, nebo k zábavě. Pro zajímavost, v tomto roce se konal v Dubaji první světový závod dronů, kde závodníci používají kamery a VR brýle pro snadnější ovládání dronů a větší přesnost.

## Raspberry Pi 2

Raspberry Pi 2 je světově známý minipočítač s ARM procesorem druhé generace. Díky Raspberry Pi se vývoj minipočítačů neuvěřitelně zvýšil a otevřel nové obzory pro vývojáře, kdy místo běžných mikrokontrolérů s menším výkonem mohou používat plnohodnotný linuxový počítač s GPIO porty podporující většinu známých sériových komunikací. Tento minipočítač však nezaujal pouze vývojáře, kteří mají tuto práci pouze jako své hobby ale i větší firmy, které používají toto zařízení jako testovací, před výrobou vlastní desky. V těle o rozměrech 8,8 x 5,6 cm se nachází čtyřjádrový procesor ARM Cortex-A7, pracující s frekvencí 900Mhz, RAM s kapacitou 1GB, grafický procesor VideoCore IV, podporujcí OpenGL ES 2.0, 1080p30 a MPEG-4, čtyři USB porty verze 2.0, 100 megabitový ethernetový adaptér s konektorem RJ-45, 40 GPIO portů, HDMI konektor, zvukový výstup v podobě 3.5 milimetrový jacku a slot pro microSD kartu.

* 1. **Arduino Uno**

*„Arduino je otevřená elektronická platforma, založená na uživatelsky jednoduchém hardware a software. Arduino je určeno pro kutily, bastlíře, umělce, designéry - zkrátka pro každého, koho zajímá vytváření interaktivních objektů nebo prostředí.“*

* czechduino.cz

Arduino Uno je nejznámější produkt společnosti Arduino. Existuje spousta jiných výrobců používající volně dostupný návrh platformy (např. Genuino, Funduino), které lze používat stejně jako oficiální Arduino Uno. Arduino Uno používá mikrokontrolér Atmega328P, nabízí 6 analogových výstupů, 6 PWM, které lze použít i jako běžný digitální výstup a dalších 8 digitálních výstupů. Frekvence mikrokontroléru je nastavena externím krystalem na 16Mhz. Paměť tohoto mikrokontroléru činí 32kB, kde 0,5kB zabírá předem vypálený Arduino bootloader, který zajišťuje základní funkce, které samotný mikrokontrolér nenabízí (např. odesílání výstupu po USB sběrnici). Výhodou tohoto zařízení je snadné nahrávání programu do mikrokontroléru pomocí USB sběrnice, snazší kód (založený na běžném AVR C, které lze také použít) a široká škála dostupných knihoven.

## ADXL345 a L3G4200D

ADXL345 je malý, nízkonapěťový tříosý akcelerometr s možností měřit v opravdu velkém přetížení (až ±16g). Jeho velkou výhodou je možnost komunikace po sběrnici I2C a schopnost detekovat pád. Jeho přesnost je 0.1 stupně

L3G4200D je nízkonapěťový tříosý gyroskop s vysokou přesností a možností filtrace nižších vibrací s maximálním škálováním 2000 stupňů za vteřinu a stejně jako akcelerometr ADXL345 je schopný komunikovat po sběrnici I2C.

## Ostatní HW

Mezi další HW, který jsem ke své maturitní práci použil, se řadí dále ultrazvukový modul HC-SR04 a kamerový modul AR0130 s infračerveným filtrem.

Ultrazvukový modul HC-SR04 je kompaktní modul o velikost 44x20 milimetrů, který obsahuje svou řídící jednotku, která pouze čeká na dotaz a vrací dobu odezvy v mikrosekundách. Používá 4 piny, z nichž 2 slouží pro napájení a další 2 pro komunikaci (echo a Trigger). Komunikace s tímto modulem funguje tak, že do pinu Trigger se pošle logická 1 po dobu 5 mikrosekund a poté se čeká na odpověď na pinu Echo. Zaznamenaná doba signálu se poté vydělí hodnotou 57.753393 (rychlost zvuku v centrimetrech za mikrosekundu vydělená dvěma).

Kamerový modul AR0130 je modul s CMOS čipem o rozlišení 1.2MP komunikující po USB sběrnici z které je i napájen. Maximální kvalita videa je 1280x720/30fps. Tento modul má výhodu v možnosti automatického přepínání infračerveného filtru. Na modulu pod čipem se nachází světelný senzor, který automatický sepne panel s infračervenými diodami a sepne infračervený filtr.

## C++

*„C++ je multiparadigmatický programovací jazyk, který vyvinul Bjarne Stroustrup a další v Bellových laboratořích AT&T rozšířením jazyka C. C++ podporuje několik programovacích stylů (paradigmat) jako je procedurální programování, objektově orientované programování a generické programování, není tedy jazykem čistě objektovým. V současné době patří C++ mezi nejrozšířenější programovací jazyky.“*

* *cs.wikipedia.org*

Dodatkové knihovny, které používám v mé dlouhodobé maturitní práci, jsou libWebSockets (webová komunikace pomocí WebSocketu), wiringPi (práce s GPIO porty u Raspberry Pi 2) a pThread (Tvorba více vláknového programu).

* 1. **AVR C**

Mikroprocesory AVR je možné programovat 2ma způsoby. Buď v assembleru, kdy je program opravdu rychlý, za to ale vývoj trvá celkem dlouho. Poté je zde možnost pomocí jazyka C, kdy se používá běžný jazyk C se základní knihovnou avr/io.h která zajišťuje celkovou komunikaci s výstupy a registry. Dále je zde spousta dalších knihoven, například pro práci s pamětí EEPROM, pro delay, …

Při programování arduina však nejsou tyto knihovny potřeba, protože jsou již obsaženy v knihovnách vydaných přímo od společnosti Arduino a usnadňují práci s ním, proto také používá i vlastní IDE (Arduino IDE). Rozdíl programu pro Arduino od běžného C programu je, že základní struktura má dvě funkce (setup() a loop()), kdy funkce setup() je volaná pouze při startu programu a loop() je běžný nekonečný cyklus.

## Objektově orientované programování

Objektově orientované programování je způsob programování (paradigma), kterým se řídí většina nových a nejčastěji používaných programovacích jazyků. Koncepce se skládá z Objektů, abstrakce, zapouzdření, skládání, delegování, dědičnosti a polymorfismu.

Objekt je základní, jedinečnou a jednoznačně identifikovatelnou entitou, která je dána parametry, které odlišují objekt od ostatních objektů a dále chováním, které objekt poskytuje svému okolí. Celý program se poté skládá ze skupin objektů, které si posíláním zpráv navzájem říkají, co je třeba udělat. Objekt, stejně jako proměnná má vlastní paměť. Ta je tvořena na základě velikosti jeho parametrů, které obsahuje. Každý objekt, má určitý typ, neboli každý objekt je instancí třídy.

Dědičnost je možnost vytvoření třídy, která je podtřídou, třídy jiné a dědí téměř všechny funkce a proměnné. Poté navíc může obsahovat vlastní proměnné a funkce.

Třída je datovým typem objektu. Třída určuje společné vlastnosti svých instancí. Třída může obsahovat funkce a proměnné. Ty se pak dále dělí do 3 typů práv. Práva máme public, private a protected . Funkce a proměnné s právy public, jsou všem přístupné a lze s nimi pracovat mimo třídu. Private je přesným opakem publicu. Funkce a proměnné s právy private nejsou přístupné mimo třídu, pouze uvnitř a ve zděděných třídách. Právo protected je právo s největším omezením. Proměnné a funkce s právy protected jsou přístupné pouze uvnitř funkce a nelze je dědit.

## Javascript a JQuery

Javascript je skriptovací multiplatformní objektově orientovaný jazyk pro webové rozhraní. Stará se o aktivní prvky fungující v reálném čase bez potřeby aktualizace stránky. Skript tedy běží na straně klienta a proto má i svá omezení. Například nemůže pracovat se soubory, z důvodu ohrožování soukromí uživatele. Dle průzkumu je na webu pouze 1,3% webů, které neobsahují Javascript.

JQuery je javascriptová knihovna, která urychluje práci s HTML kódem pomocí javascriptu a přidává javascriptu mnohem více možností. Poznat JQuery knihovnu v javasciptu je velice snadné, jelikož používá alias „$“.

Zde je příklad změny barvy pozadí na červenou v JQuery:

*$(‘body’) .css (‘background’, ‘#ccc’);*

A zde to samé v čistém javascriptu:

*Function changeBachground(color) {*

*Document.body.style.background = color;*

*}*

*Onload=”changeBackground (‘red’);*

## WebSocket

Websocket je protokol poskytující obousměrnou komunikaci (full-duplex) pomocí jednoho TCP spojení. Byl navržen pro webové prohlížeče a server, ale nyní se dá použít i v běžné aplikaci. K navázání spojení se používá URI ws a wss (např. ws://localhost:8080/) a používá se k rychlé komunikaci se serverem často posílaných dat, nejčastěji u chatu.

## I2C

I2C je sériová sběrnice používaná u většiny modulů pro vývojovou desku Arduino a také u k používání nízkorychlostních periferií (např. řízení jasu displeje). V této sběrnici je vždy pouze jedno řídící zařízení (master) a poté zbytek řízená (slave). Každé slave zařízení má svou adresu, na kterou se poté posílá dotaz, a maximální počet adres/zařízení je 128. K zapojení jsou potřeba 2 vodiče. První rozesíla hodinový signál (SCL), ten je řízen masterem, a druhý pro datový tok (SDA). K zahájení komunikace se používá podmínka START, po které následuje adresa zařízení. Poté si všechna zařízení (slave) porovnají své adresy s přijatou adresou a pokud se shoduje, zašle potvrzovací bit ACK. Poté přijímá nebo vysílá data (podle typu dotazu). Adresa zařízení se skladá ze sedmi bitů kde několik adres je již vyhrazeno pro speciální účely (např. 0000000 je broadcast adresa).

## PWM

Pulzně šířková modulace, neboli PWM, je modulace pro přenos analogového signálu pomocí dvou hodnot (logická 1 a 0). Hodnota je zakódována do opakující-se smyčky, kdy délkou logické 1 se udává zaslaná hodnota. Hodnota se tedy dá vyjádřit i v procentech, kdy 100% je stála logická 1.

## paměť EEPROM

Paměť EEPROM je trvanlivá paměť na dlouhodobé ukládání dat. Je součástí každého mikrokontroléru a velikost se pohybuje kolem 1kB. Rozdíl práce s EEPROM pamětí v AVR C jazyce a v Arduino IDE jsou pouze v jiných knihovnách. Práce s ní je naprosto stejná. Pro získáný bajtu stačí zavolat adresu v rozsahu velikost EEPROM paměti a funkce EEPROM pro získání bajtu jej vrátí, jako návratovou hodnotu.

## Komplementární filtr

Komplementární filtr je usnadněný způsob jak zkombinovat data z gyroskopu a akcelerometru do vyfiltrované hodnoty od většiny vibrací a okolního ruchu. Jeho nástupcem je Kalmanův filtr, který je dokonalejší díky tomu, že se snaží předvídat další hodnotu z předchozích, ten je ale i mnohem náročnější na výpočet a proto se ještě stále používá v některých případech komplementární filtr. Jeho principem je převést hodnotu z gyroskopu (stupně za sekundu) na aktuální úhel, čehož dosáhneme přičítáním jeho hodnoty do proměnné při každé iteraci a poté násobením dobou trvání jedné iterace. Tím získáme méně přesný úhel s lepší kompenzací okolních vibrací a nyní jej můžeme zkombinovat s přesnějším úhlem z akcelerometru, ale s vysokou náchylností na otřesy. Proto vezmeme 98% hodnoty z gyroskopu a pouze 2% z akcelerometru, čímž získáme výslednou hodnotu.

## PID algoritmus

Jedná se o řídící algoritmus, který se vždy řadí před řízenou soustavu. Dělí se na tři části. Proporcionální, integrační a derivační část. Tento algoritmus se používá u kvadrokoptér ke stabilizaci a jsou dva způsoby jak jej použít. První metoda je používání pouze hodnot z gyroskopu. Tato metoda je sice snazší ale velice nepřesná, proto se častěji používá druhá metoda. Tou je kombinace výstupní hodnoty z Kalmanova či Komplementárního filtru s hodnotou z gyroskopu, čímž získáme jak chybu v úhlu oproti nastavené hodnotě, ale ten se poté i zkompenzuje s prudkostí reakce motorů a nedojde tak k přestřelení hodnot (při správném nastavení konstant). Po získání odchylky (tzv. erroru), začnou se počítat všechny tři složky zvlášť. K jejich výpočtu je navíc vždy potřeba konstanta nastavená uživatelem (která se musí správně naladit). Proporcionální složka je tzv. zesilovač. Reguluje pouze aktuální odchylku a neřeší nic kolem. Integrační složka je složka, starající se o náhlé odchylky (náraz větru, nárůst váhy na jedné straně, …). K výpočtu je potřeba proměnná sloužící jako paměť z předchozích iterací, do které se přičítá výsledek při vynásobení odchylky s konstantou. Tato konstanta musí být rozhodně menší než konstanta P. Poslední složka (Derivační) se stará o tuhost záběru. Zabraňuje prudkým pohybům a vypočítává se rozdílem aktuální odchylky s předchozí a následným vynásobením rozdílu s konstantou D. Při provedením výpočtu všech tří složek se výsledky sečtou. Tento algoritmus se používá pro každou osu zvlášť, tudíž při každé iteraci se vypočítávají tři výsledné hodnoty PID algoritmu pomocí tří různých trojic konstant.

## NetBeans IDE

NetBeans IDE je vývojové prostředí, původně vytvořené pro jazyk Java, českými studenty Matematicko-fyzikální fakulty Univerzity Karlovi. Nyní však vlastní tento projekt firma Oracle Corporation a jedná se o multiplatformní IDE podporující Javu, PHP, Groovy, C, C++ a webové technologie (HTML, CSS, Javascript). Jeho velkou výhodou je možnost kompilace C či C++ programu na vzdáleném zařízení na cizí platformě, než na které běží samotné IDE. Oproti většině takovýmto IDE s podporou vzdálené kompilace, NetBeans podporuje i vzdálený debug a synchronizaci knihoven ze vzdáleného zařízení.

## Arduino IDE

Arduino IDE je vývojové prostředí od společnosti Arduino, pro práci s Arduino produkty. Obsahuje textový editor, textovou konzoli pro výstup, funkce pro nahrání programu, kompilaci, vypálení bootloaderu na čip, Sériový monitor pro zaznamenávání Sériových dat zasílaných z Arduina po sériovém portu (ke kterému je připojené i USB rozhraní). Arduino IDE má také možnost otevření již vytvořeného programu pro snadnější pochopení práce s různými funkcemi a výstupy. Arduino IDE je jednoduché IDE v kterém je snadné se zorientovat a je perfektní pro začátečníky i pokročilé programátory.

# Úvod do problému

Výsledkem mé závěrečné praktické maturitní práce je řídící systém pro kvadrokoptéru s pokročilou stabilizaci, detekcí překážek na nižší úrovni a zajímavým ovládacím rozhraním.

## 3.1 Použitý HW

Jelikož řídící systém je potřeba na něčem testovat, je zapotřebí vybrat určitý HW. Různých typů rámů, motorů a regulátorů je velké množství, proto je potřeba se trochu informovat o typech, jejich kvalitě, použitelnosti a možností sehnat náhradní díly. Poté je také potřeba vybrat nejprve výpočetní jednotky a poté jim vhodnou baterii, která se hodí i k vybraným regulátorům otáček a motorům. Výpočetních jednotek (miniPC, mikrokontrolérů, …) je opravdu spousta a jejich nabízený výkon je opravdu dostačující, ale dále se musí hledět i na jejich pracovní napětí, jelikož budou napájeny pouze baterií.

## 3.2 Rozvrstvení řídícího systému

Řídící systém je velice široký pojem, pod kterým se skrývá několik běžících programů, které se starají o určitou část na určité úrovni za určitým důvodem. Proto je nezbytné posoudit, jaké informace jsou nejdůležitější, které musí být uživateli skryté, které rozhodně viditelné a podle toho přizpůsobit výsledné programy.

## 3.3 Stabilizace

Jednou z nejpodstatnějších částí řídícího systému je rozhodně stabilizace. Je potřeba rozhodnout, jaká data budou používána, jakou časovou hranici by neměla překročit jedna iterace, aby stabilizace dostatečně fungovala a jaký algoritmus vlastně použít. Dále je také součástí tohoto problému i filtrace vibrací a dalších rušení, k čemu je potřeba vybrat buď Kalmanův filtr, anebo Komplementární filtr.

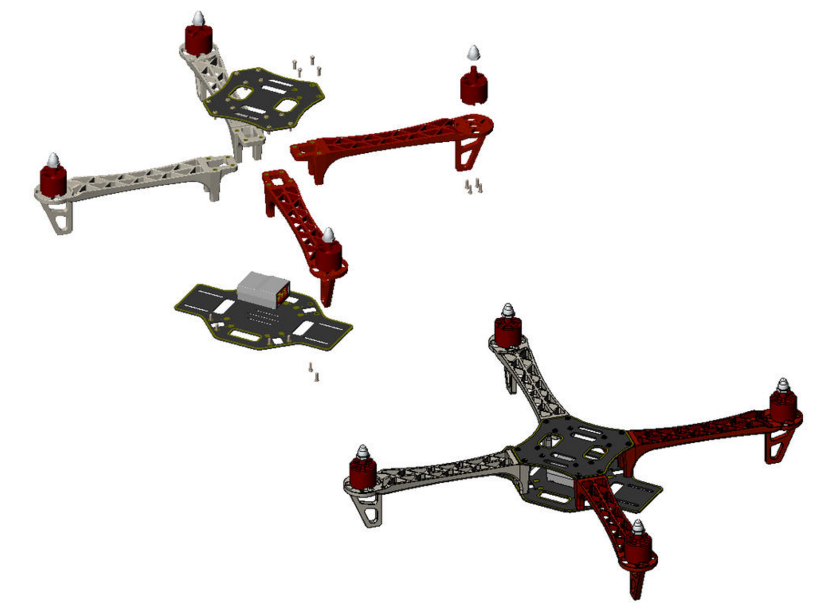
## 3.4 Detekce překážky

Možností jak detekovat překážku je opravdu široká škála. Na každý typ se používají různé moduly a liší se i požadovaný výpočetní výkon. Jedním z nejméně náročných na výpočet je detekce pomocí ultrazvukových modulů, díky kterým získáme vzdálenost od překážky vzdálené až 3 metry. Jejich nevýhodou je však problém s odrazem zvuku od zaoblených ploch a pod velkým úhlem, kdy se zvuk odrazí směrem od modulu a vzdálenost je poté nemožné změřit, tudíž modul nevidí danou překážku. Druhý způsob je pomocí dvou kamer (tzv. stereo kamer), které musí směřovat stejným směrem a poté se obraz zprůměruje a čím větší rozdíl v určitém bodě je, tím je daný objekt blíž. Tento způsob je opravdu velice přesný, ale jeho nevýhodou jsou vysoké nároky na výkon a také problém detekcí skla a zrcadla. Poslední metoda, kterou používají společnosti zabývající se drony a celkově mechatronikou, je detekce pomocí otáčejících se laserů. Jedná se o tyč, na které jsou nad sebou rozmístěné lasery (32 nebo 64), které se otáčejí kolem osy ve velké rychlosti a měří vzdálenost od překážky. Poté se vytvoří mapa vzdáleností a podle toho se pak systém rozhoduje. Tato metoda je také velice přesná a nemá ani problém s detekcí lesklých materiálů a nároky na výkon jsou nižší díky integrovanému čipu v daném modulu, který sám dopočítává vzdálenostní mapu a tu teprve přeposílá do řídící jednotky. Nevýhodou je vysoká cena modulu.

# Řešení problému

## Postup při výběru HW

Při vybírání HW jsem musel začít nejprve výběrem testovací kvadrokoptéry. Po přečtení několika recenzí různých rámů, jsem vybral rám F450 od známého výrobce DJI s již přiloženými vrtulemi DJI 1045. Jeho výhodou je snadno sehnatelné náhradní díly a střed obsahuje již připravený základní obvod pro napájení všech motorů z baterie. Dále jsem vybral motory DJI 2212, které v recenzích vedou v poměru cena/kvalita a k nim již přiložené regulátory otáček DYS Simonk s maximálním pracovním proudem 30A. Poté jsem vybíral výpočetní jednotky. Jako první výpočetní jednotku jsem podle zadání vybral Arduino Uno, a jako druhou výpočetní jednotku jsem vybral Raspberry Pi 2 z důvodu nabízeného výkonu a nízkého požadovaného napětí. Následně jsem vybral baterii vhodnou pro všechny vybrané součástky, jímž byla baterie Hacker ECO X-20C. Jedná se o tříčlánkovou baterii s kapacitou 3800mAh a výstupním napětím 11.1V. Rozdíl mezi napětím výstupním a požadovaným pro Raspberry Pi 2 jsem vyřešil převodník z 6-24V na 5V s výstupem na USB. Pro bezdrátovou komunikaci jsem použil již pořízený a spolehlivý bezdrátový USB adapter od TP-Linku TL-WN722N.



Obrázek 1CAD model kvadrokoptéry DJI F450; Autor: Alaa Elhamrawy

## 4.2 Rozvrstvení řídícího systému

Rozvrstvení celého řídícího systému byla celkem dlouhá kapitola z důvodu vytvoření dobrého základu, na kterém se řídící systém bude stavět. Systém jsem nakonec rozdělil na 3 vrstvy. O nízkoúrovňovou práci s většinou HW se stará Arduino. Má na starost ovládání regulátorů motorů, kontrolu kapacity baterie, komunikaci s gyroskopem a akcelerometrem pomocí sériové sběrnice I2C, stabilizaci a zasílání dat po sériovém výstupu do Raspberry Pi 2. Arduino Uno je tedy tzv. back-end. Dále na Raspberry Pi 2 běží program, který je jakýsi data switch a data controller, starající se přijímání dat po sériových portech, zpracovávání WebSocket dotazů z webové stránky (front-endu) a nakonec, díky výhodě multitaskingu a lepší možnosti zabránění komunikace ovladače s back-endem, zpracovává data z ultrazvukových modulů a zajišťuje odpojení ovladače od back-endu a následné převzetí kontroly a zabránění nárazu.

## Postup řešení programu

1. **Komunikace s regulátory otáček (ESC)**

Jako první co jsem vyzkoušel, byla komunikace s regulátory otáček. Arduino Uno již má knihovny pro komunikaci s motory (Wire.h), kde stačilo nadeklarovat proměnnou datového typu Servo, nastavit port pro komunikaci s regulátorem a následné odeslání PWM signálu. První funkční výsledek byl regulátor otáček ovládaný Arduinem pomocí potenciometru.

1. **I2C komunikace s akcelerometrem a gyroskopem**

Dalším krokem byla komunikace s gyroskopem a akcelerometrem. Pro pochopení celé této komunikace jsem nejprve použil výrobcem poskytovaný příklad. Poté následovalo dlouhé pročítání datasheetů každého zařízení a zkoumání co dělá který registr. Následně se mi povedlo získat první data z akcelerometru, kdy bylo potřeba získat data ze dvou registrů (2 bajty), které se poté sloučili pomocí operace *((HIGH\_BYTE<<8)|LOW\_BYTE)*. Přesněji se jedná o posun prvního bajtu o jeden bajt vlevo a poté pomocí logické operace OR sloučení s druhým bajtem. Po úspěšně navázané komunikaci s akcelerometrem se mi to samé podařilo i s gyroskopem. Následně jsem vytvořil kalibrační funkci, kdy se získá 2000 dat, které se z každé osy zvlášť nasčítají do proměnných a poté vydělí 2000, čímž jsem získal průměrnou odchylku a tu poté při dalším dotazu pro získání dat odečetl od výsledku.

1. **První sériová komunikaci s Raspberry Pi 2**

Jako třetí krok jsem se pokusil navázat první sérové spojení mezi Arduino Uno a Raspberry Pi 2. U Arduina jsem použil třídu Serial, která slouží ke komunikaci po sérovém portu a data posílal ve formátu, kdy první bajt je buď 0xFF nebo 0x00, kdy 0xFF značí, jestli se jedná o záporné číslo a 0x00 určuje kladné číslo. Druhý bajt je samotné číslo převedené do jednoho bajtu. Začátek komunikace je označen čtyřikrát za sebou zaslanou hodnotou 0xFE. Na Raspberry Pi 2 jsem použil knihovnu WiringPi.h a jeji Třídu WiringSerial, kde jsem nejprve zkontroloval spojení se samotným portem a poté získal zaslané bajty, které jsem ukládal do pole (poslední 16 bajtů). Poté pomocí cyklu jsem nalezl čtyřikrát po sobě jdoucí hodnoty 0xFE a pokud byla zbývající délka pole natolik velká, aby obsahovala zbylá zaslaná data, byla načtena, pokud ne, čekalo se na zbytek zaslaných dat. Po získání dat, byly hodnoty převedeny opět na číslo typu integer, a pokud bajt před samotnou hodnotou se rovnal 0xFF, hodnota byla vynásobena následně -1. Nakonec se data vypsala do konzole a zkontrolovala se stabilita spojení. Následně jsem navýšil frekvenci sérové komunikace z 9600bps na maximální 115200bps a opět zkontroloval stabilitu spojení.

1. **První komunikace s front-endem pomocí WebSocketu**

Čtvrtým krokem byl pokus o navázání první komunikace pomocí WebSocketu s C++ programem, používající knihovny libWebSockets. Nejprve jsem vytvořil základní webovou stránku s obyčejným formulářem s číselnými vstupy a propojení signálu při změně hodnoty ve vstupu s odesláním dat přes WebSocket. Dále jsem použil v C++ programu základní strukturu knihovny libWebSockets, kde jsem smazal odpověď na dotaz GET, a povolil pouze komunikaci přes WebSockety a následní získávání zaslaných dat. S integrací knihovny libWebSockets do vlastní třídy byl celkem problém ale po pár dnech se podařilo a vše fungovalo, jak mělo. Po testech stability se nalezlo pár chyb s mazáním dynamických proměnných a malým bufferem pro přijímaná data. Po nalezení a následném opravení chyb vše fungovalo tak, jak mělo, a mohl jsem se pustit do dalšího kroku.

1. **První použití PID algoritmu**

V tomto bodě jsem se poprvé pokusil použít PID algoritmus. Bohužel však neúspěšně, jelikož jsem používal čistě jen data z gyroskopu, která jsem pouze filtroval jednoduchou metodou, kdy jsem k 80% přechozí hodnoty přičetl 20% hodnoty aktuální. Poté jsem začal testovat, kdy jsem na kvadrokoptéru přidělal vrtule a aktivoval pouze dva protilehlé motory. Za zbylá dvě ramena jsem přivázal kvadrokoptéru k ramínku, zavěšeného k lustru a začal ladit konstanty algoritmu kromě konstant pro osu Yaw. Pokus o ladění trval přibližně 3 hodiny, dokud se baterie nevybila. Během testování jsem však narazil na problém ve složitosti ovládání pomocí formuláře a problém s chybějící možností okamžitého vynulování hodnot (zastavení motorů).

1. **Tvorba lepšího ovládacího rozhraní**

Pro lepší ovladatelnost jsem kvadrokoptéry jsem musel na pár dní ukončit ladění konstant PID algoritmu a předělat ovládací webové rozhraní. Jako nejlepší řešení jsem zvolil ovládání pomocí šipek (náklony) a kláves Shift a Ctrl (výška). Inkrementace hodnot by však měla probíhat konstantně po stejných intervalech, proto jsem nejprve musel pomocí knihoven JQuery zaznamenat stisknutí klávesy, poté porovnání, o jakou klávesu se jedná a nastavení booleanové proměnné dané klávesy na true. Při puštění klávesy se zavolá funkce dělající přesný opak, neboli mění booleanovou proměnnou na false. Během těchto operace už od začátku načtení webové stránky běží nekonečný interval, spouštěný každých 50 milisekund, hledající jaká klávesa je stisknutá pomocí booleanových proměnných. Pokud je daná proměnná rovna hodnotě true, inkrementuje/dekrementuje hodnotu pro danou operaci a odešle data pomocí WebSocketu. Poté jsem se pustil do řešení okamžitého vypnutí motorů. To jsem vyřešil čekáním na stisknutí klávesy Mezerník, kdy se však nemění žádná booleanová proměnná, ale všechny zbylé booleanové proměnné se nastaví na false (zruší se stisk všech kláves), všechny výstupní hodnoty se nastaví na 0 a data se odešlou pomocí WebSocketu. Tímto způsobem jsem dosáhl snadno použivatelného ovládacího rozhraní s možností rychlého vypnutí bez zaváhání.

1. **Návrat k použití PID algoritmu**

Prvně jsem se opět pokusil pouze vyladit konstanty ale po hodině ladění a bez jediného úspěchu jsem se rozhodl hledat chybu v datech. Poté jsem si všiml velkých odchylek v datech při vyšších otáčkách motorů. Proto jsem začal pátrat po rychlém a dobrém filtru, který by tento problém vyřešil. Nejprve jsem narazil na Kalmanův filtr, který byl velice složitý nejen na pochopení ale i na výpočet, proto jsem hledal dál a nakonec jsem nalezl Komplementární filtr, který jsem již popsal dříve (2.15). Po implementaci filtru do mého kódu jsem začal opět s testováním. První den se mi nepodařilo trefit správné konstanty pro přesnou stabilitu, ale druhý den se mi to již podařilo, ale opět jsem narazil na problémy. První problém byl, že data, která jsem posílal, a měli být původně úhly náklonu, se stali úhly uraženými za jednu iteraci, tudíž se kvadrokoptéra nevracela při puštění kláves do původní polohy, ale zůstala ve své aktuální. Druhý problém byl, že data z gyroskopu byli nepřesné a rovina se po pár vteřinách změnila v náklon deseti stupňů.

1. **Vytvoření designu pro ovládací webové rozhraní**

Po několika dnech neúspěšného testování PID algoritmu, jsem odskočil od tohoto problému a pustil jsem se do klidnější části a tou byl návrh a vytvoření designu webové stránky. K tomuto účelu jsem použil grafický vektorový editor Inkscape, kde jsem si vytvořil design letadlového akcelerometru a každou část vyexportoval, kvůli prohlížeči Edge, který má stále problémy se zpracováváním vektorových obrázků, ve formátu .png, s rozlišením 1000x1000.Poté jsem tyto obrázky přidal do webového rozhraní, nastavil barvu textu a pozadí podle návrhu a poté se pustil do předělávání C++ programu.

1. **Optimalizace a editace WebSocket serveru**

Kvůli vizualizaci akcelerometru a dat z gyroskopu bylo potřeba předělat WebSocket server. Prvně jsem editoval funkci pro příjem dat, kdy data nyní byla posílaná ve formátu csv (data oddělována středníkem), tudíž nejprve nalezne první středník a získá text, nacházející se před ním. Tyto data jsou typ komunikace. Určují, zda se jedná o zasílání hodnot motoru, nebo později například o autentizaci. Po určení typu dotazu se data zpracují, a pokud se jedná o nastavení motorů, data odešle všem ostatním připojeným klientům, kteří jsou uloženi v poli. Proto bylo potřeba vytvořit funkci pro odesílání dat klientům. Při této změně jsem si však všiml velkého zpomalení, proto jsem předělal funkci pro odesílání dat všem zbylým uživatelům, aby se data pro každého klienta posílala v jiném vláknu a po dokončení se vlákno samo ukončilo a poté smazalo. Tím se výkon opět navýšil a vše pracovalo správně. Nakonec jsem vytvořil funkci pro zasílání přijatých dat z akcelerometru a z gyroskopu všem klientům.

1. **Funkčnost vizuálního akcelerometru v ovládacím webovém rozhraní**

Po umožnění přijímat ve front-end rozhraní data z gyroskopu a akcelerometru, nezbývalo nic jiného, než nastavit funkci pro příjem dat přes WebSocket, kde jsem opět zpracoval typ dotazu a poté nastavil pro část akcelerometru css styl *transform: rotate(x deg)*, kde x je přijatá hodnota hodnota osy Y (roll). Pro druhou část akcelerometru jsem nastavil css styl *top: calc(50% - x px)*  kde x je přijatá hodnota X (pitch). Zbylá data se vypíšou pod graficky zpracovaný akcelerometr.

1. **Zabezpečení**

Z důvodu bezpečnosti, jsem také musel do webového ovládacího rozhraní zakomponovat zabezpečení heslem, kdy má možnost se přihlásit pouze první připojený uživatel. Zbylí uživatelé jsou pouze tzv. přihlížející, kteří mohou pouze sledovat ovládání kvadrokoptéry a její náklony. Pokud se přihlášený uživatel odpojí, převezme kontrolu uživatel, který se připojil hned jako druhý. Při pokusu o přihlášení od jiného uživatele se zobrazí text „BUSY“, v překladu „zaneprázdněn“, a nezmizí, dokud se opět uživatel, který ovládal kvadrokoptéru, neodpojí. Toto je ošetřeno dotazem typu „verify“, který pošle zahashované zadané heslo na WebSocket Server, který zpracuje dotaz, a porovná přijatá data s heslem zadaným v kódu. Pokud se heslo shoduje, uživateli pošle odpověď ve tvaru „verify;1“. Pokud se však heslo neshoduje, pošle uživateli odpověď ve tvaru „verify;0“. Třetí možnost může nastat, pokud je uživatel již přihlášen a snaží se přihlásit další a zadá helso také správně. V tomto případě odešle WebSocket server odpověď ve tvaru „verify:2“. A poslední možný případ je odpověď, kdy se odpojí master uživatel a je volné jeho místo.

1. **Finální verze PID algoritmu**

Po dokončení zabezpečení ovládacího webového rozhraní jsem se mohl opět pustit do ladění PID algoritmu. Po dlouhém hledání jsem však našel lepší způsob jak jej počítat. Je zde přidána jedna konstanta na víc a počítá se jak s daty z akcelerometru, tak i z gyroskopu. Po implementaci kódu opět následovalo ladění konstant. K lepším výsledkům jsem došel až po 4 dnech ladění, kdy se sice kvadrokoptéra ve vzduchu celkem kmitala a občas udělala podivné chybné náklony, ale výsledek už byl použitelný.

1. **Ladění PID konstant za běhu z ovládacího webového rozhraní**

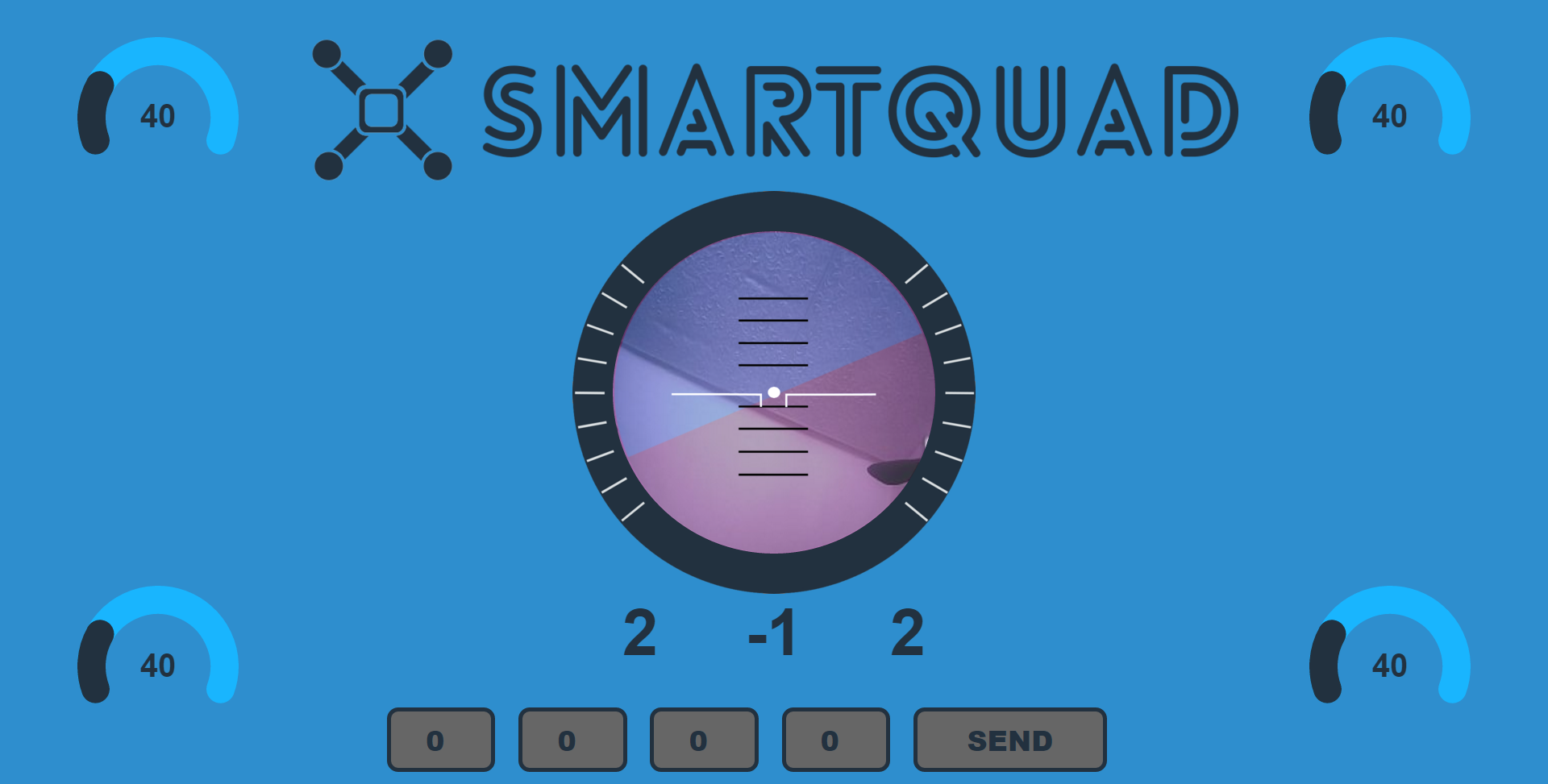
Dalším problémem, který bylo potřeba vyřešit, byla rychlejší možnost ladění PID konstant, jelikož do teď bylo potřeba přehrávat celý program, kvůli změně konstant a to velice zdržovalo, proto jsem vytvořil ve webovém ovládacím rozhraní 4 vstupní pole pro zadání všech 4 konstant pro osu roll a pitch, které se poté poslali po WebSocketu jako dotaz *PID* a ty Raspberry Pi 2 poslalo dále do Arduina. Tím jsem si velice zkrátil čas strávený laděním.

1. **Streamování videa z kamery do webového ovládacího rozhraní**

Po vyřešení problému se stabilizací, jsem se pustil do vylepšování ovládacího webového rozhraní řídícího systému, kdy jsem přidal stream z kamery s podporou infračerveného filtru. K tomu jsem použil linuxový software mjpg\_streamer, který umí pracovat jako samotný deamon a přenáší obraz z kamery na webový server pod portem 8081 jako často aktualizovaný obrázek. Odkaz k němu stačilo pouze vložit do obrázku v mém projektu, který jsem vložil doprostřed akcelerometru s 80% průsvitností a zaoblil jej podle velikosti kruhu akcelerometru.

1. **Detekce překážek pomocí ultrazvukových modulů**

K detekci překážek jsem nakonec použil čtyři ultrazvukové moduly HC-SR04, z kterých zpracovávám data pomocí Raspberry Pi 2 z důvodu možnosti běhu každého ultrazvukového modulu ve vlastním vláknu. Původně jsem používal knihovnu pthread.h jako jinde ke komunikaci s ultrazvukovými moduly, bohužel jsem však nedostal žádnou odpověď od modulu. Proto jsem použil multivláknovou funkci z knihovny wiringPi.h, který ihned fungovala tak, jak měla, a mohl jsem zapojit všechny čtyři ultrazvukové moduly. Po úspěšném zprovoznění a získávání dat díky použití mutexu, který ošetřuje, aby do jedné statické proměnné nezasahovalo najednou více vláken, jsem mohl zvětšit odesílaná data pro WebSocket server a zasílat i vzdálenost od překážek webovému ovládacímu rozhraní, kde je vytvořená css animace plynulého blikání čtvrtkruhu kolem akcelerometru, pro každý ultrazvukový modul zvlášť, kdy se nastavuje podle vzdálenosti překážky, viditelnost této signalizace. Poté jsem pouze ošetřil, že před odesláním dat back-endu se zkontroluje vzdálenost od překážky, a pokud je nalezena, přepíše hodnotu pro danou osu a zastaví před překážkou.



Obrázek 2Front-end rozhraní

1. **Finální doladění a otestování**

Ve finálním doladění jsem smazal nepoužívané a zakomentované části kódu, přidal indikaci vybité baterie a zabránění dalšímu letu do nabití baterie. Poté proběhlo testování, kde jsem vyzkoušel každou část kódu a i přes menší nedostatky, testování proběhlo úspěšně.

## Finální běh řídícího systému

**Arduino Uno**

Při startu programu se nadeklarují a nadefinují globální proměnné programu. Například konstanty pro PID algoritmus, adresy I2C zařízení, čtyři proměnné pro komunikaci s regulátory otáček, PID proměnné pro každou osu zvlášť a kalibrační proměnné. Poté se spustí funkce setup(), ve které se nejprve zahájí přenos po sériových portech s frekvenci 115200bps, zahájí se přenos po I2C sběrnici a nastaví se registry akcelerometru a gyroskopu. Poté si program načte z paměti první bajt a zkontroluje, zda je roven 0, či ne. Pokud je roven log 0, spustí se kalibrace gyroskopu a akcelerometru. Ta probíhá pomocí for cyklu, který provede 2000 iterací (2000 hodnot přičtených do kalibračních proměnných) a poté všechny proměnné vydělí hodnotou 2000. Následně kalibrační proměnné uloží do EEPROM paměti a na první bajt EEPROM paměti vloží log 1, čímž informujeme Arduino při dalším startu, že v paměti už jsou uložené kalibrační hodnoty. Pokud je tedy hodnota log 1 prvního bajtu, načtou se do kalibračních proměnných hodnoty z paměti. Jelikož se jedná o proměnné datového typu float, hodnoty mají velikost 4 bajty, proto i adresy paměti EEPROM jsou od sebe vzdáleny 4 bajty. Po načtení kalibračních hodnot se při dalším dotazu na získání hodnot z gyroskopu nebo z akcelerometru kalibrační hodnota odečte od získané hodnoty z akcelerometru/gyroskopu. Poté se nastaví piny pro komunikaci s regulátory otáček (ESC) a odešle se hodnota 0. Poté se spustí funkce loop(). V ní se nejprve zísají hodnoty z akcelerometru a gyroskopu. Poté se získají data ze sérového portu. Tato operace se provede nejprve ukládáním zaznamenaných hodnot do dočasné proměnné, dokud není počet zbývajících hodnot roven 15. Poté se čeká na bajt 0xFE. Pokud je tento bajt nalezen, přečte se další bajt. Podle tohoto bajtu se rozhodne, o jaký typ dat se jedná. Pokud je tento bajt roven hodnotě 0x00, jedná se o data pro řízení. Pokud je roven 0x01, jedná se o nastavení PID konstant. Podle typu očekávaných dat se porovná, zda je počet zbývajících dat větší nebo roven potřebné velikosti dat. Pokud ne, funkce skončí a čeká se na další iteraci. Po získání všech dat přijde na řadu Komplementární filtr. Nejprve se gyroskopová data převedou na úhel způsobem, který jsem popsal v druhé kapitole (2.15). Následovně se provedou všechny potřebné výpočty tohoto filtru a výsledek se dosadí do globální proměnné. Hned po tomto výpočtu se získá z analogového vstupu hodnota kapacity baterie a vydělí hodnotou získanou při plné kapacitě. Poté probíhá kontrola, zda se hodnota, udávající výkon otáček všech motorů nerovná log 0 a zda kapacita baterie není příliš nízka. Pokud se nerovná nule a baterie není vybitá, probíhá dosazení přijatých hodnot pro ovládání do globálních proměnných s omezením minimálního a maximálního náklonu ±20 stupňů. Poté proběhne kalkulace PID algoritmu (2.16). Po této kalkulaci se výsledné hodnoty přičtou/odečtou k danému motoru dle závislosti a výsledná hodnota se pošle do ESC. Pokud se ale hodnota výkonu otáček všech motorů rovná nule nebo stav baterie klesne pod minimální hodnotu, vynulují se všechny hodnoty PID algoritmu a odešle se do všech motorů hodnota 0. Poté proběhne odeslání získaných dat z akcelerometru a gyroskopu do Raspberry Pi 2 po sériovém portu ve formátu 0xFE, 0xFE, 0xFE, 0xFE (zahájení přenosu), 0xFF/0x00(záporné číslo/kladné číslo), hodnota akcelerometru osy pitch, 0xFF/0x00, hodnota akcelerometru osy roll, 0xFF/0x00, hodnota akcelerometru osy yaw, a dále to samé s hodnotami z gyroskopu. Poté se zahájí další iterace nekonečného cyklu.

**Raspberry Pi – C++**

Při startu tohoto programu se nejprve zavolá statické funkce mnou vytvořené třídy SerialComunication::setup(), která zahájí přenos po sériovém portu pod danou frekvencí a ověří, zda bylo možné navázat spojení. Pokud ne, funkce vrátí hodnotu false a program se ukončí. Pokud však vše proběhne správně program dále zavoláte statickou funkci třídy WebSocket::setUp(), která se postará o základní nastavení knihovny libWebSockets a nastaví hash přihlašovacího hesla. Jako poslední funkce před zahájením nekonečného cyklu se zavolá statická funkce třídy Sonar, Sonar.:setUp(). Ta se postará o vytvoření 4 vláken starajících se o získávání dat z ultrazvukových modulů a dosazování hodnoty do statických proměnných třídy SerialComunication. Před změnou hodnoty se vždy nejprve uzamkne mutex hodnota v paměti a poté zase odemkne. Po nastavení všech tříd se spustí nekonečný cyklus, který nejprve zavolá funkci SerialComunication::getSerialData(), která nejprve získá data ze sériového portu podobným způsobem jako Arduino, dosadí hodnotu do statické proměnné a poté uzamkne všechny mutexy, pro odeslání dat všem klientům WebSocket serveru, poté opět všechny mutexy odemkne a odešle získaná data z WebSocket serveru po sériovém portu. Následně po této statické funkci se zavolá statická funkce WebSocket::service(), která se stará o všechny dotazy na WebSocket server. Nejprve zjistí, o jaký typ dotazu se jedná, poté data zpracuje daným způsobem. Pokud se jedná o dotaz *verify*, porovná získaný hash s hashem pro přihlášení a pokud jsou stejné, pošle odpověď, a uloží si dané spojení do proměnné controller pro oboustrannou kontrolu, zda data posílá uživatel, který k tomu má opravnění. Pokud se jedná o dotaz *engines*, program načte všechny hodnoty a poté je dosadí do statických proměnných třídy SerialComunication. Poslední typ dotazu je *PID*, kdy program opět zpracuje odeslaná data a poté je uloží do statických proměnných třídy SerialComunication a zavolá funkci SerialComunication::sendPID() pro odeslání PID hodnot Arduinu. Poté se spustí další iterace nekonečného cyklu.

**Raspberry Pi – front-end**

Při načtení webového ovládacího rozhraní se nejprve nadeklarují globální proměnné a provede se definice některých z nich, například definice layoutu při přihlášení, při odhlášení, při spojování a při zaneprázdnění. Poté se naváže spojení přes WebSocket. Po navázání spojení se spustí funkce socket.onopen, která spustí funkci passwdListener() čekající na stisknutí tlačítka Enter pro odeslání dat na WebSocket Server. Následně se načte zbytek webu a čeká se na zadání hesla. Po zadání hesla se odešle dotaz na server se zahashovaným heslem pomocí MD5 algoritmu. Poté příjde od WebSocket Serveru odpověď, která se zpracuje ve funkci socket.onmessage. Zde se rozdělí získaná data pomocí funkce split() podle středníku a porovná se první hodnota v první indexu získaného pole s typy dotazu. Pokud se jedná o odpověď verify, na kterou web čeká, ověří hodnotu v dalším indexu pole. Ta pokud se rovná hodnotě 0, vymaže se text v inputu a změní se placeholder na napíš „Wrong password“. Pokud se rovná hodnotě 1, skryje se ztmavená vrstva, a spustí se interval vložený do proměnné keyListener volající funkci keyListenerHandler každých 50 milisekund. Tato funkce zpracovává data z booleanových hodnot, které se mění podle stisku kláves a podle toho inkrementuje hodnoty pro daný typ náklonu a poté data vizualizuje pomocí knihovny jquery.knob.js. Dále má socket ještě jednu funkci, která se volá při ukončení spojení s WebSocket serverem. Ta vypne interval keyListener, změní layout na „Connecting“ a zavolá funkci pro pokus o nové spojení každých 500 milisekund.



Obrázek 3Front-end rozhraní - detekce překážky (levá strana)

## Vyskytnuté problémy a jejich řešení

1. **Nízké napětí v USB portech Raspberry Pi 2**

První problém se ukázal již při zapojení celého HW. USB Wifi Modul nechtěl pracovat, a pokud naskočil, vypadl po dalších 5 minutách. Dlouho jsem nevěděl kde je problém, zkoušel jsem překompilovat ovladač tohoto zařízení, zvýšení napětí v USB portech v souboru /boot/config.txt, ale ani jedno řešení nepomohlo, až nakonec jsem si na internetu dohledal, že důvodem je slabý proud poskytovaný USB porty u Raspberry Pi 2. Tento problém jsem vyřešil přepájením USB kabelu (male – female) kde napájení šlo přímo z převodníku na 5V a data přes USB port. To samé jsem později musel provést ještě jednou kvůli kamerovému modulu s podporou infračerveného filtru, který také nedostával dostatečně silný proud a bez externího napájení se vůbec nepřihlásil k PC.

1. **Při nízké kapacitě baterie pád Raspberry Pi 2**

Dalším problémem byla detekce nízké kapacity baterie. Jelikož jí dlouho můj řídící systém neobsahoval, dost často se při testování stávalo, že kapacita baterie klesla na tak nízkou hodnotu, že Raspberry Pi 2 přestalo pracovat, tím pádem nebylo možné kvadrokoptéru jakkoliv ovládat, ale Arduino si pamatovalo posledně zaslaná data. Tento problém jsem vyřešil přivedením přes kondenzátory sníženého napětí z baterie do analogového portu Arduina a detekcí příliš nízkého napětí baterie s následným omezením znovu vzlétnou. Druhým ošetřením je počítání doby od posledních dat ze sériového portu, a pokud jsou data pro motory starší jak sekunda, motory vypnou.

1. **PID Stabilizace**

Opravdu velkým problémem se pro mě při vytváření maturitní dlouhodobé praktické práce stala samotná stabilizace kvadrokoptéry. Nad tímto problémem jsem strávil přes 50% stráveného času u projektu a i přes to není úplně dokonale vyřešen. Problém ale bohužel není v algoritmu, ale ve vibracích, které jdou z rámu do gyroskopu a akcelerometr. Při záznamu hodnot do grafu jsou odchylky nefiltrovaných dat v rozsahu ±32 stupňů, což je opravdu dost. Tento problém se mi povedlo alespoň částečně vyřešit a trochu odtlumit rám na přijatelnou sílu vibrací. Toho jsem dosáhl díky vyvážení vrtulí, zpevnění rámu a použití antivibrační desky pro gyroskop a akcelerometr.

# Závěr, vyhodnocení

I přes pár vyskytnutých problémů je výsledek mé dlouhodobé praktické maturitní práce téměř naprosto funkční podle zadání. Nedostatkem jsou silné výkyvy z důvodu vibrací produkovaných rámem, tudíž se nejedná o chybu řídícího systému, ale HW.   
Závěrem bych chtěl poděkovat vedení SPŠ za poskytnuté prostory pro testování mé praktické maturitní práce a panu Zvěřinovi za obětavost a dobré rady při řešení problémů se stabilizací.

# Zdroje

<https://www.raspberrypi.org/products/raspberry-pi-2-model-b/>

<https://www.arduino.cc/en/Main/ArduinoBoardUno>

<http://czechduino.cz/?co-je-to-arduino,29>

<http://owenson.me/build-your-own-quadcopter-autopilot/codeview.html>

<http://api.jquery.com/>

<http://pimylifeup.com/raspberry-pi-webcam-server/>

<https://libwebsockets.org/libwebsockets-api-doc.html>

<http://www.analog.com/media/en/technical-documentation/data-sheets/ADXL345.PDF>

<http://www.st.com/web/en/resource/technical/document/datasheet/CD00265057.pdf>

<http://www.brokking.net/ymfc-3d_qanda.html>

<https://github.com/ArduPilot/ardupilot/blob/master/libraries/PID/PID.cpp>

<http://wiringpi.com/reference/setup/>

<https://barkingbogart.wordpress.com/2014/10/07/test-drive-libwebsockets-library-a-simple-server/>

<http://www.pieter-jan.com/node/11>

Obrázek 1: <https://grabcad.com/library/quadrotor-frame-1>

# Seznam příloh

CD se zdrojovými kódy všech programů, elektronickou formou dokumentace a prezentací.