Jihočeská univerzita v Českých Budějovicích
Přírodovědecká fakulta

Quadrocopter - stabilizace polohy pro autonomní let

Bakalářská práce

Aleš Mrázek

Školitel: PhDr. Milan Novák, Ph.D.

České Budějovice 2017

Bibliografické údaje

Mrázek, A., 2017: Quadrocopter - stabilizace polohy pro autonomní let. [Quadrocopter - stabilizing the position for autonomous flight. Bc.. Thesis, in Czech.] - 85 p. Faculty of Science, The University of South Bohemia, České Budějovice, Czech Republic.

Anotace

Bakalářská práce se zabývá obecnou problematikou autonomního letu multikoptér v prostoru. V práci je problematika rozdělena do několika dílčích částí, které jsou následně popsány. Pro demonstraci možného řešení je použita vývojová platforma Crazyflie a senzor Kinect spolu s knihovnou počítačového vidění OpenCV.

Klíčová slova

Multikoptéry, Crazyflie, Autonomní let, Navigace, Počítačové vidění, OpenCV. Kinect

Annotation

Bachelor thesis deals with general issues of multicopter autonomous space flight. Thesis is divided into several smaller parts, which are subsequently described. Possible solution is demonstrated on the Crazyflie development platform and Kinect sensor along with a computer vision library OpenCV.

Keywords

Multicopter, Crazyflie, Autonomous flight, Navigation, Computer vision, OpenCV, Kinect

Prohlašuji, že svoji bakalářskou práci jsem vypracoval/a 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 ..... Podpis autora .....

Poděkování

Rád bych touto cestou poděkoval PhDr. Milanu Novákovi, Ph.D. za jeho cenné rady a trpělivost při vedení mé bakalářské práce. Rovněž bych chtěl poděkovat své rodině za podporu v průběhu studia i při vypracování bakalářské práce.

Obsah

Úvod

V důsledku výrazného technologického pokroku, především v narůstání výpočetního výkonu, redukování fyzické velikosti výpočetních systémů a rozšíření open-source technologií dochází ke kladení stále vyšších nároků na automatizaci a samostatnost systémů. Jedná se o eliminaci potřeby vnějšího zásahu člověka do uzavřeného systému, bez kterého by systém přestal pracovat nebo zkolaboval, takový systém se nazývá autonomní. To se týká především oborů jako je robotika nebo internet věcí (IoT – Internet of Things). Tím pádem vzrostl také zájem o výzkum v oblasti bezpilotních vzdušných zařízení UAV (Unmanned Aerial Vehicle) spadajících do oboru robotiky. Jednou ze součástí UAV jsou vícevrtulová zařízení MAV (Mikrokopter Aerial Vehicle) a Mini UAV, jinak také nazývané jako multikoptéry. Bezpilotní prostředky se již dlouhou dobu využívají k vojenským účelům, kde měli nahradit klasické letecké prostředky v oblastech vysokého rizika nebezpečí pro piloty. Až později si tyto prostředky našly své místo v civilních sférách. Tato zařízení, především multikoptéry se v několika posledních letech těší velkému zájmu, především díky flexibilitě, široké využitelnosti a dostupnosti těchto zařízení.

Základním předpokladem pro tvorbu autonomních multikoptér a UAV prostředků je úspěšná realizace autonomního letu. Jde o doplnění takzvaného „autopilota“ o další systém, který nahrazuje lidskou obsluhu ve vztahu k řízení prostředku. Autopilot na základě vyhodnocení údajů o poloze a orientaci pomocí programových algoritmů a pravidel stabilizuje a řídí multikoptéru. Díky autopilotu doplněného o systém pro autonomní řízení je multikoptéra schopna samostatného pohybu, který je důležitý k dalším automatizovaným činnostem.

Cíle práce

Hlavním cílem práce je navržení systému umožňující UAV prostředku Crazyflie 2.0 autonomní let s použitím sensoru Kinect. Pro dosažení tohoto cíle je problematika rozdělena do několika realizačních bodů.

- Definování obecného modelu systému pro autonomní let UAV.
- Zapojení a zprovoznění jednotlivých hardwarových a softwarových prvků systému.
- Zajištění komunikace mezi jednotlivými prvky systému.
- Definování souřadného systému pomocí senzorů.
- Umožnění detekce pozice multikoptéry v souřadném systému.
- Vytvoření navigačního algoritmu pro řízení Crazyflie 2.0 uvnitř souřadného systému.

Metodika práce

Práce vychází z analýzy dostupných řešení a jejich kombinací spolu s všeobecně známými pohybovými, fyzikálními a matematickými znalostmi a principy. Na základě této analýzy je problematika v teoretické části zdokumentována, rozvedena a popsána. Na tomto základu je postavena část praktická se zaměřením na konkrétní technologie a konkrétní možné řešení.

1. Analýza dostupné literatury a projektů spojených s problematikou této práce.
2. Navržení obecného modelu problematiky autonomního letu multikoptér.
3. Rešerše možností využití technologií.
4. Navržení a popsání možného řešení.
5. Nastavení jednotlivých prvků a instalace potřebného softwarového a hardwarového vybavení.
6. Realizace navržených algoritmů, řešení problémů a testování.

1 Úvod do problematiky

1.1 Klasifikace UAV

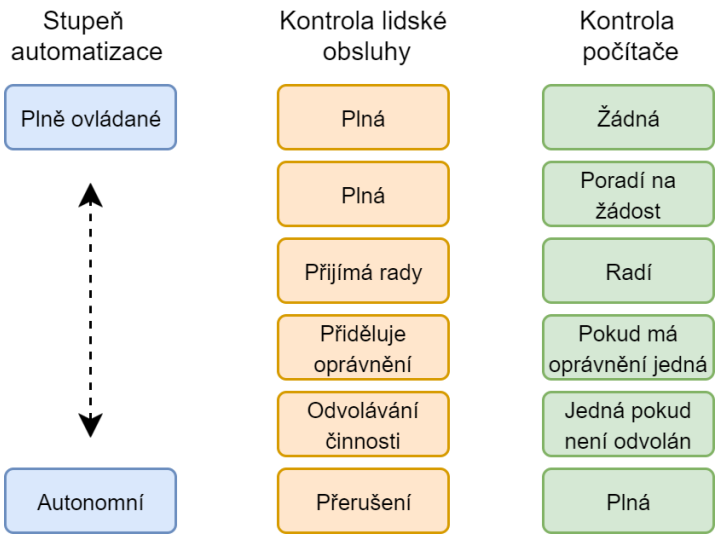
UAV jsou velmi rozsáhlou skupinou zařízení, v současné době neexistuje žádná jejich všeobecně přijímaná klasifikace. Na tato zařízení lze nahlížet z různých perspektiv, zejména v závislostech na jejich parametrech a specializaci. Následující příklady klasifikace jsou popsány v knize [1].

Velikosti UAV:

- Very small UAV – velmi malá UAV
    - Micro nebo Nano UAV – pod 30 cm
    - Mini UAV – 30-50 cm
- Small UAV – malá UAV, nad 50 cm
- Medium UAV – střední UAV, 5-10 m
- Large UAV – velká UAV, nad 10 m

Stupně autonomie:

- **Dálkově ovládané** – UAV jsou pod přímou a stálou kontrolou lidského operátora. Ten zadává příkazy prostřednictvím stanice, která udržuje konstantní datové spojení s UAV.
- **Autonomní** –UAV jednají zcela nezávisle, podle toho, jakým programem jsou vybaveny. Jsou vysílány na takzvané mise, ve kterých nevyžadují žádný další lidský zásah a jsou plně samostatné. UAV analyzuje stav svého okolí za pomoci senzorů a rozhoduje se podle předem naprogramovaných pravidel.



Obrázek 1-1:Autonomie UAV

Klasifikace vytvořená USAF (Letectvo Spojených států amerických)

- Tier N/A: Small/Micro UAV – nejmenší velikost a nejvíce přenositelné bezpilotní prostředky
- Tier I: Low altitude, long endurance – nízká hladina letu, dlouhá výdrž
- Tier II: Medium altitude, long endurance (MALE) – střední hladina letu, dlouhá výdrž
- Tier II+: High altitude, long endurance conventional (HALE) UAV – vysoká hladina letu, dlouhá výdrž
- Tier III-: High altitude, long endurance low-observable UAV – vysoká hladina letu, dlouhá výdrž, nízká míra zpozorování (stealth)

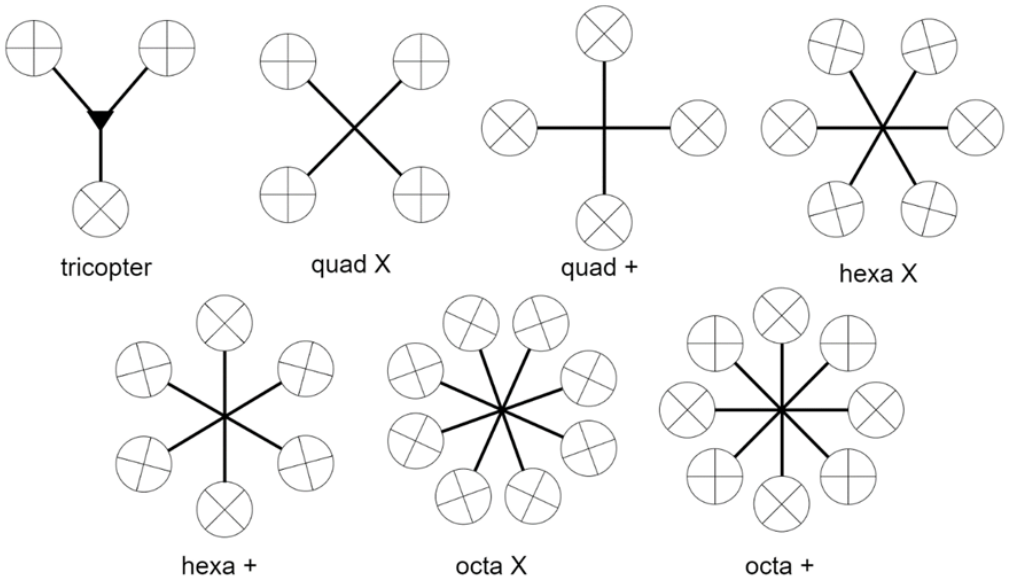
Možná využití UAV prostředků

- **Snímání** – shromažďování dat, podle toho, jakými senzory jsou vybaveny. Senzorů je celá řada v závislosti na konkrétní aplikaci, od biologických, chemických, elektromagnetických atd.
- **Monitoring** – mohou být vybaveny kamerou nebo dalšími foto zařízeními s vysokým rozlišením, které slouží k monitorování velkých oblastí, kde by bylo nepraktické instalovat statické kamery, např. v lesích, polích atd.
- **Průzkum** – vzhledem k eliminaci rizika vystavovat lidský život přímému nebezpečí, jsou velmi užitečné pro zkoumání nebezpečných nebo nehostinných oblastí, jako jsou místa katastrof, bojové zóny apod.

- **Doprava** – bezpilotní prostředky mohou poskytovat dostatečný výkon pro přenos dalších objektů. To má sice viditelný vliv na spotřebu energie, ale je to stále praktický způsob, jak rychle dopravit např. zásoby do izolovaných oblastí.
- **Armádní** – vzhledem k tomu, že bezpilotní letouny byly původně určeny pro armádní účely, je tento způsob využití UAV stále rozvíjen.
- **Vyhledávání a záchranářství** – tento směr zaměření je charakteristický pro oblasti postižené přírodní katastrofou, kde je omezen přístup a vysokou roli hraje čas. Pokud by byl bezpilotní prostředek vybaven například termokamerou, mohlo by mnohem rychleji docházet k nalézání přeživších.

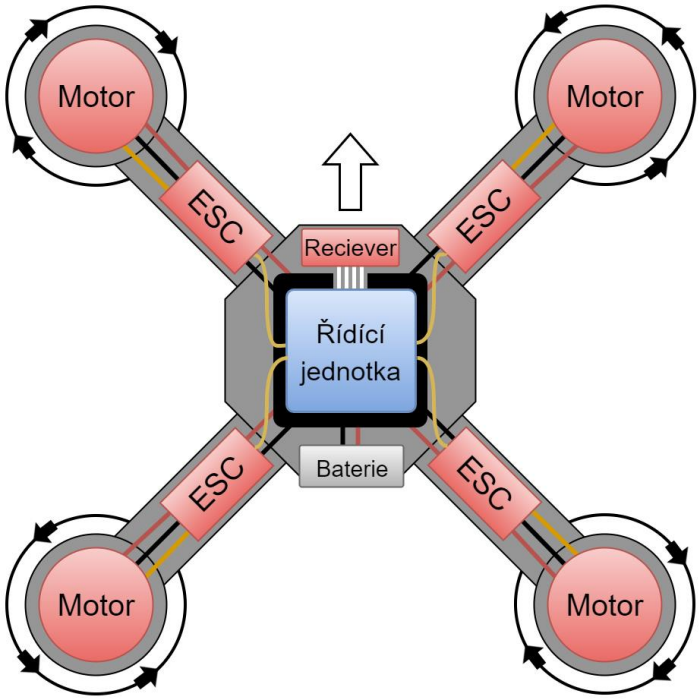
1.2 Multikoptéry

Multikoptéry jsou vícevrtulové bezpilotní létající stroje, které se často klasifikují do MAV (Mikrokopter Aerial Vehicle) a Mini UAV podkategorií bezpilotních letadel UAV. Základem je rám, na kterém jsou ve stejné vzdálenosti od těžiště umístěny motory s vrtulemi. Rámy mohou mít různé tvary, podle počtu potřebných vrtulí a vybavení, které má multikoptéra nést. Nejčastěji používaným rámem je takzvaný „quad X“, což je rám pro 4 motory ve tvaru písmene X. Ovládání pohybu je prováděno převážně změnou tahů jednotlivých motorů s vrtulemi.



Obrázek 1-2: Typy rámů multikoptér

Klasické multikoptéry mají 3-8 střídavých elektromotorů (ECM-Electronically Commutated Motors) s vrtulemi. Ty pracují na principu elektromagnetické indukce složeny ze dvou hlavních částí, statoru a rotoru. Každý motor je připojen na kontrolér rychlosti (ESC-Electronic Speed Controller), který udává směr rotace a rychlost motoru podle instrukcí z řídící jednotky. U malých (nano, mini) multikoptér bývá součástí motorů nebo řídící jednotky. Nejčastěji probíhá komunikace mezi multikoptérou a ovladačem prostřednictvím rádiových vln. Řídící signály jsou odesílány z ovladače pomocí rádiového vysílače (transmitter) a přijímány na multikoptéře rádiovým přijímačem (reciever). Multikoptéra je pak řízena na základě rádiových signálů a údajů ze senzorů, díky kterým je stabilizována.



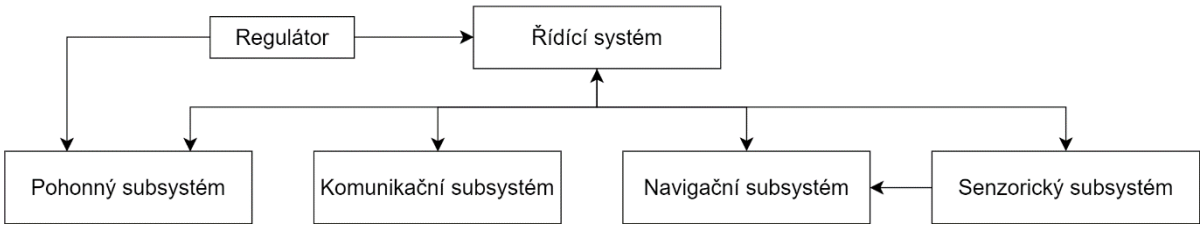
Obrázek 1-3: Jednoduché obecné schéma multikoptéry

1.2.1 Problematika autonomie multikoptér

Autonomie multikoptér závisí převážně na schopnostech „vnímání“ prostředí ve kterém se multikoptéra pohybuje. Prostředí je vnímáno pomocí senzorického vybavení, ve vztahu k autonomnímu letu se jedná o zjišťování informací o orientaci a pozici v prostoru. Na základě těchto informací je multikoptéra řízena pomocí navigačních a reakčních algoritmů. Problematika autonomního letu multikoptér není samostatně definována, lze ale vycházet z obecné problematiky týkající se mobilních robotických systémů. Obecně je problematika autonomních robotů podle literatury [2] rozdělena do několika subsystémů, ze kterých se kompletní autonomní robot skládá. Těmito subsystémy jsou:

- Řídící systém
- Pohonný subsystém
- Navigační subsystém
- Komunikační subsystém
- Senzorický subsystém
- Reakční subsystém

Následující schéma ukazuje uspořádání subsystémů a jejich interakci.



Obrázek 1-4: Schéma subsystémů autonomního robota

Práce se zaměřuje pouze na některé tyto subsystémy, nebo jejich části. Nezabývá se kompletním sestrojením, ale pouze částmi, které se přímo týkají autonomního systému a problematiky autonomního letu. Jednotlivé části jsou detailněji popsány v následující kapitole.

2 Teoretická východiska

Obsah této kapitoly vychází z literatury [2] upravené pro potřeby multikoptér vycházející z projektů Bitcraze [3] a [4].

2.1 Řídící subsystém, platforma

Řídící systém je často postaven na jedné vývojové platformě, která je mozkem celého robota a od které se odvíjejí základní vlastnosti a možnosti robota. Tyto platformy poskytují výpočetní výkon, integrují vstupně výstupní periferie a propojují jednotlivé části robota. Základem takového systému je řídící jednotka, které se v problematice multikoptér říká letový kontrolér (flight controller). Mezi populární platformy patří například Arduino a Raspberry Pi. Dalším důležitým krokem je naprogramování a konfigurace řídícího systému. V tomto kroku záleží na zvolené platformě, od které se odvíjí možnost pouhého nakonfigurování nebo kompletního naprogramování firmwaru kontroléru.

V problematice autonomních robotů jsou dvě základní možnosti, jak řešit správu autonomního chování. Od těchto možností se odvíjí, v jakém režimu pracuje řídící subsystém robota [2].

- Režim **SLAVE**: Robot potřebuje nadřazený systém. Interní CPU je podřízený, slouží systému nadřazenému, předává mu data a přijímá jeho povely. Není potřeba vysoký výkon interního CPU
- Režim **MASTER**: Robot nepotřebuje nadřazený systém. Interní CPU spravuje autonomní chování celého robota. Je potřeba vyšší výkon interního CPU.

Jelikož bývají výpočty, které souvisí se správou autonomního chování náročnější a většina platforem neposkytuje dostatečný výpočetní výkon, bývá řídící subsystém převážně v režimu „slave“. V tomto režimu je robot plně podřízený nadřazenému systému, od kterého získává pokyny.

2.1.1 Komunikační subsystém

Komunikační zařízení je důležité pro komunikaci a kooperaci s okolním prostředím. V případě práce se jedná o komunikaci mezi multikoptérou, nadřazeným systémem nebo rádiovým ovladačem. Nejčastěji bývá komunikace zprostředkována pomocí rádiového spojení o několika kanálech, příkladem může být Wifi, Bluetooth, GSM, zigBee.

2.2 Nadřazený systém

Nadřazený systém se stará o správu operací týkajících se autonomního chování. Jedná se o stanici, která může být realizována jako jednotlivé moduly (subsystémy) propojené počítačovým programem v obyčejném PC nebo minipočítači jako je Raspberry Pi.

Nadřazený systém lze rozdělit na několik menších subsystémů:

- Komunikační subsystém – komunikace s podřízeným systémem (multikoptéra)
- Navigační subsystém – nejdůležitější část, závisí na ní kvalita celého autonomního systému.
    - Získávání pozice multikoptéry
    - Výpočet řídících parametrů
    - Regulátor
- Senzorický subsystém – poskytuje důležitá data, ze kterých je navigační subsystém schopný určit pozici multikoptéry.

2.3 IMU (Inertial measurement unit)

Orientace v prostoru je určena třemi úhly nebo prostorovým vektorem. Znalost orientace hraje důležitou roli při autonomním letu multikoptér, především pro řízení, stabilizaci a navigaci. Aby bylo možné určit orientaci jsou UAV vybaveny řadou měřících zařízení, zvané inerciální měřící jednotky (IMU). IMU jsou elektronická zařízení, získávající informace o rychlosti, orientaci a gravitačních silách, díky datům získaných z akcelerometrů, gyroskopů a magnetometrů.

2.3.1 Akcelerometr

Akcelerometry slouží pro měření zrychlení. Zrychlení je vektorová veličina, která vyjadřuje časovou změnu rychlosti hmotného bodu, nebo soustavy hmotných bodů. Ze znalosti zrychlení a hmotnosti lze zjistit sílu působící na těleso a podle povah sil je možné zrychlení rozdělit do dvou základních kategorií: [5]

- **Dynamické zrychlení:** síla vzniká změnou rychlosti pohybujícího se tělesa
- **Statické zrychlení:** síla vzniká působením gravitačního pole Země

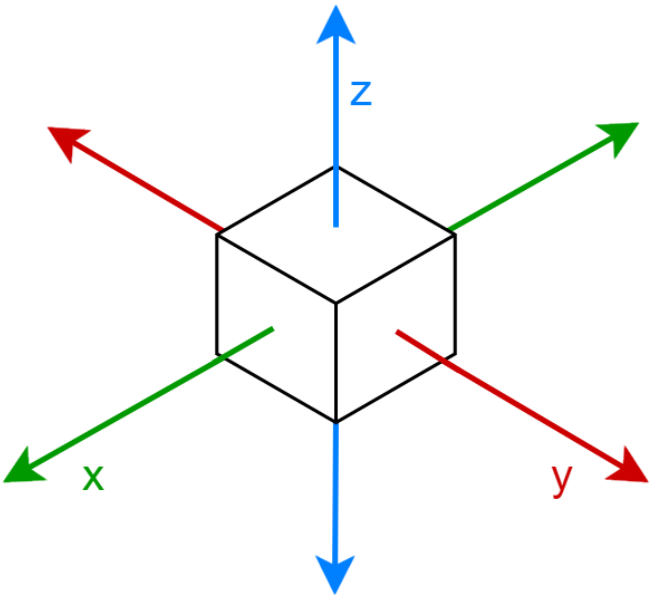
Statické zrychlení, které se naměří na jednotlivých osách, závisí na orientaci akcelerometru vůči zemi. Pokud se akcelerometr nachází vodorovně, na ose *z* se naměří přibližně 1g (gravitační zrychlení), pokud se obrátí vzhůru nohama, tak -1g a na osách *x* a *y* je zrychlení nulové. Pokud se akcelerometr překlopí na bok, na ose *z* bude zrychlení nulové a na ose *x* 1g nebo -1g. Záleží na tom, zda se překlopí na levý nebo pravý bok. Analogicky k tomu, převrácení k sobě nebo od sebe kolmo k zemi se na ose *y* naměří zrychlení 1g nebo -1g. Z naměřených hodnot na osách lze vypočítat úhly k jednotlivým osám. [7]

ρ ... *úhel svírající osa x se zemí*
φ ... *úhel svírající osa y se zemí*
θ ... *úhel svírající osa z se zemí*
x, y, z ...*naměřené hodnoty na osách*

$$\rho = arctan\left(\frac{x}{\sqrt{y^2 + z^2}}\right)$$

$$\varphi = arctan\left(\frac{y}{\sqrt{x^2 + z^2}}\right)$$

$$\theta = arctan\left(\frac{\sqrt{x^2 + y^2}}{z^2}\right)$$



Obrázek 2-1: Akcelerometr

Za předpokladu, že gravitace je jediná síla působící na snímač, poskytuje akcelerometr poměrně přesné údaje úhlové orientace. Nicméně, při pohybu a otáčení senzoru, mohou působit jiné síly a dochází ke kolísání přesnosti.

Příklady použití [6]:

- Měření otřesů a vibrací – akcelerometry mohou sloužit k měření vibrací např. strojů nebo zemětřesení. Mohou kontrolovat správný chod stroje a předpovídat možné zadrhnutí ložisek.
- Měření setrvačných sil – akcelerometry mohou sloužit k měření vzdálenosti nebo rychlosti.
- Měření náklonu – akcelerometry mohou měřit zrychlení zemské gravitace. Ze znalosti statického gravitačního zrychlení lze vypočítat náklon tělesa (roviny senzoru)

2.3.2 Gyroskop

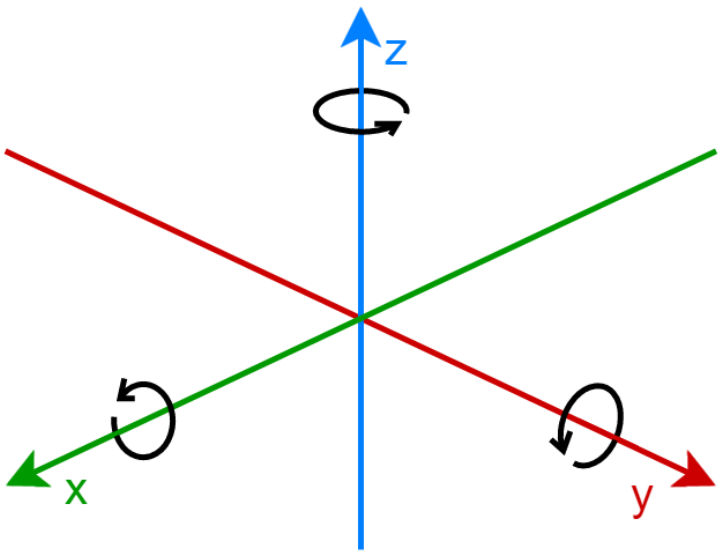
Gyroskopy jsou využívány pro měření a určování změny polohy a natočení předmětu, ke kterému jsou připevněny. Primárně měří úhlové zrychlení, údaj o tom, jak rychle se měřený předmět otáčí v jednotkách stupňů za sekundu. Rotace je měřena vzhledem k jedné z os *x*, *y*, *z*. Pro výpočet orientace se musí nejdříve inicializovat pozice se známou hodnotou, tím může být například údaj z akcelerometru. Poté se změří úhlová rychlost kolem os *x*, *y* a *z* v nějakém časovém intervalu, pro změnu úhlu pak platí: [8]

ω ... *úhlová rychlost*

dt ... *časový interval*

γ ... *změna úhlu*

$$\gamma = \omega \cdot dt$$



Obrázek 2-2: Gyroskop

Celkový úhel orientace lze zjistit přičtením změny úhlu k předchozímu úhlu orientace nebo případně k inicializační orientaci.

2.3.3 Magnetometr

Magnetometr se používá k měří magnetické indukce pole Země. Velikost magnetické indukce závisí i na aktivitách jádra Země, aktivitě Slunce a magnetických bouřích v ionosféře, na materiálu zemské kůry a podobně. Magnetická indukce je vektorová veličina, má velikost a směr. Magnetometry se obecně dělí do dvou kategorií, které se silně liší ve funkci a principu práce. [9]

- Vektorové magnetometry – měří hodnotu magnetické indukce ve zvoleném směru v prostoru. Hodnota magnetické indukce je pak vektor se směrem a velikostí.
- Skalární magnetometry – měří pouze hodnotu indukce nezávisle na směru.

Ve vztahu k řízení UAV dávají smysl pouze vektorové magnetometry, které určují vektor se směrem a velikostí. To ale neznamená, že jimi nemohou být vybaveny.

2.3.4 Odchylky měření

IMU jsou náchylné k systematickým chybám. Akcelerometr poskytuje přesné údaje v dlouhodobém horizontu, ale je nepřesný v krátkodobém měření. Gyroskop poskytuje přesné údaje o měnící se orientaci v krátkodobém horizontu, ale nezbytná integrace způsobuje, že výsledkem je drift v delších časových úsecích. Magnetometr je ovlivňován výkyvy elektromagnetického záření, které mohou být způsobeny mnoha aspekty prostředí jako jsou sluneční erupce, železobetonové konstrukce nebo měřené blízko vysokého napětí.

Řešením odchylek měření je vhodná kombinace těchto měřících jednotek, aby došlo k vyvážení chyb. Standardním způsobem kombinování těchto dvou vstupů je aplikace Kalmanova filtru, který má poměrně složitou metodiku (viz 2.5.1). Existují i jednodušší způsoby pro kombinaci dvou typů dat, jedním z nich je tzv. Komplementární filtr. Výpočet pomocí tohoto filtru je jednoduchý a poměrně přesný, proto je výhodné jej použít v případě omezeného výpočetního výkonu. Jeho hlavní nevýhodou je, že neposkytuje žádné informace o velikosti gyroskopického driftu. [35]

Rovnice komplementárního filtru:

a ... *váha pro gyroskop*

$(1-a)$ ... *váha pro akcelerometr*

dt ... *vzorkovací perioda*

$$\text{úhel}_{\text{nový}} = a \cdot \left(\text{úhel}_{\text{předch.}} + \text{úhlové}_{\text{zrychlení}_{gyr}} \cdot dt\right) + (1-a) \cdot \text{úhel}_{akc}$$

2.4 Navigace

Pro jakoukoli schopnost autonomního pohybu musí být multikoptéra vybavena navigačním systémem, ten často nebývá přímo součástí multikoptéry a je umístěn mimo v nadřazeném řídícím systému. Typickou úlohou navigačního subsystému je přesun z aktuální pozice do cílové. Tento úkol může být omezen řadami podmínek, kde základní podmínkou může být vyhnutí se kolizím s překážkami nebo požadavky na trajektorii či čas. K tomu složí různé plánovací a reakční algoritmy. Možnosti navigace se odvíjejí od senzorického vybavení. Navigační subsystém lze obvykle rozdělit na dvě úrovně. [2]

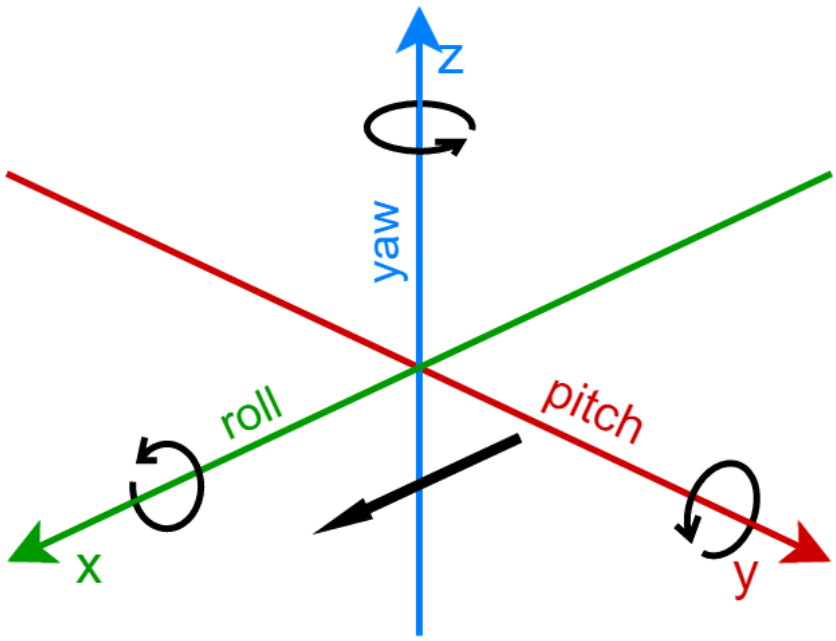
- **Globální:** Úkolem je dopravit zařízení z výchozího do cílového bodu. Vytváří globální souřadný systém daného pracovního prostoru (prostředí).
- **Lokální:** Zabraňuje kolizím s okolními objekty a je nadřazen systému globálnímu. Zpracovává informace o překážkách a možných kolizích v souřadném systému robota.

2.4.1 Pohyb multikoptér

Základním principem pohybu multikoptér je zvyšování nebo snižování otáček jednotlivých motorů s vrtulemi, které jsou určovány na základě údajů z IMU. Pro popsání směru letu se používají následující parametry (Obrázek 2-3):

- roll – naklonění, udává natočení okolo podlouhlé osy (x)
- pitch – naklopení, udává natočení okolo příčné osy (y)
- yaw – otočení, udává natočení okolo svislé osy (z)
- thrust – udává tah pro motory

Náklon (roll) je vykonáván změnou otáček motorů na pravé nebo na levé straně, v případě naklopení (pitch) předních a zadních motorů. Změna otáček způsobí, že se multikoptéra začne naklánět na stranu s motory o menších otáčkách. Otočení (yaw) okolo svislé osy je realizováno pomocí změny otáček křižných párů motorů, multikoptéra se pak začne otáčet podle směru rotorů s menšími otáčkami.



Obrázek 2-3: Pitch, roll, yaw

2.4.2 Pozice v prostoru

Získání co nejpřesnější pozice je velmi důležité pro jakýkoli autonomní pohyb. Pozice v prostoru je dána třemi souřadnicemi. Záleží na definování globálního souřadného systému, jak budou jednotlivé osy orientovány.

Absolutní navigace

Určuje současnou polohu vůči globálnímu souřadnému systému, čímž určuje takzvanou absolutní pozici. Využívá referenční body, které mají známou polohu v daném souřadném systému. Poloha robota je určena vůči těmto bodům. Tento druh navigace často využívá principů trilatence nebo triangulace viz kapitola 3.1.3. [2]

GPS (Global Positioning System), globální polohovací systém je družicový polohový systém provozovaný Ministerstvem obrany Spojených států amerických. Umožňuje na základě rádiových signálů získaných z družic určit globální polohu na celé zemi až s přesností centimetrů. K většině platforem lze jednoduše připojit samostatný GPS modul.

Vizuální navigace je založena na analýze obrazu, kde jsou v obrazu hledány a sledovány definované body. Může jít přímo o sledování robota nebo orientačních bodů.

Relativní navigace

Relativní navigace je založena na odhadu současné pozice díky měření přírůstku změny polohy vůči výchozímu bodu, nebo bobu který byl určen jako absolutní. Přírůstek je určený parametry měřitelnými na robotu. Tyto systémy mívají problémy s rostoucí chybou polohy, proto se často kombinují s absolutním systémem navigace. [2]

Odometrie spočívá v tom, že řídící systém obsahuje kinematický model robota a s jeho pomocí určuje změnu polohy v závislosti na změněn polohy akčních členů – například kol. Využívá se spíše u kolových robotů, protože je třeba zajistit stálý kontakt se zemí.

Interní navigace pracuje na principu měření zrychlení robota. Pro měření lineárních zrychlení se používají akcelerometry a pro měření úhlových zrychlení gyroskopy a magnetometry, viz 2.3.

2.5 Regulace

V technice je regulace proces udržování a usměrňování výstupních veličin systému (regulovaná veličina) na požadovanou veličinu. O automatizovanou regulaci systému se stará takzvaný regulátor, který čte stavy systému s cílem úplné eliminace odchylky. Regulace je při čtení systému v čase spojitá (analog), nebo diskrétní (digital), stejně tak zásahy regulátoru do systému. Kvalita regulace závisí na stabilitě, přesnosti a rychlosti navrženého modelu regulátoru. Modelováním regulátorů a regulovaných systémů se zabývá teorie řízení. V robotice jsou regulátory velmi důležitou součást pro stabilní a přímočarý pohyb robotů. Například naprosto identické motory mají mezi sebou drobné odchylky a každý se chová při stejných hodnotách proudu odlišně. Ve vztahu k řízení multikoptér se jedná o regulování polohy. [10]

Regulátory lze rozdělit podle druhu implementace na:

- **Softwarové –** diskrétní, jednoduchá implementace, levnější, ubírají výkon CPU, snadná modifikace
- **Hardwarové** – spojité, náročnější implementace, dražší, rychlejší, složitější modifikace

Základními typy regulací jsou:

- **P**roporcionální regulátor
- **I**ntegrační regulátor
- **D**erivační regulátor

V praxi se používají kombinace těchto základních regulací. Mezi nejpoužívanější patří PD a PID regulátor. PD regulátor je často používaný v situacích kde nedochází k rychlím změnám vstupující regulační odchylky. Naopak PID regulátor je používaný při rychlých změnách regulační odchylky, což je případem problematiky autonomního letu.

2.5.1 PID regulátor

PID regulátor si lze představit jako součet proporcionálního, integračního a derivačního regulátoru. Do regulátoru vstupuje v čase t regulační odchylka $e(t)$ a vystupuje akční veličina $u(t)$. [10] [34]

$$u(t) = r_0\, e(t) + r - 1 \int_0^t e(\tau)\, d\tau + r_1 e'(t)$$

- **P**roporcionální složka $r_0\, e(t)$
- **I**ntegrační složka $r - 1 \int_0^t e(\tau)\, d\tau$
- **D**erivační složka $r_1 e'(t)$

2.5.2 PSD regulátor

Diskrétní obdoba PID regulátoru. Algoritmus číslicového PSD regulátoru vychází z rovnice spojitého PID regulátoru tak, že je z rovnice PID regulátoru vytknuta proporcionální složka. [34]

r_0 ... *zesílení regulátorů*

T_I ... *integrační časová konstanta*

T_D ... *derivační časová konstanta*

$$u(t) = r_0 \left(e(t) + \frac{1}{T_I} \int_0^t e(t)dt + T_D \frac{de(t)}{dt} \right)$$

Jelikož číslicový regulátor nevyhodnocuje informace v čase spojitě, zavádí se převod spojitého času na diskrétní časové okamžiky. [34]

$$k \in \{0, 1,2,3, \dots\}$$

T ... *perioda vzorkování*

$$t = kT$$

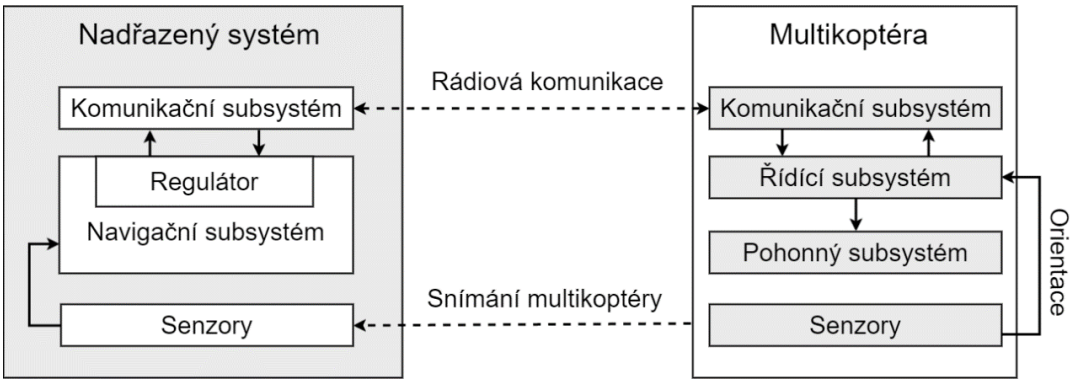
2.5.3 Kalmanův filtr

Kalmanova filtrace je speciální matematický aparát určený k filtraci signálů (dat) v časové oblasti. Ve zjednodušené formě jde o predikční-estimační[1] algoritmus, který se na základě současných a předchozích signálů (dat) snaží předpovědět průběh signálu (dat). Kalmanův filtr je například vhodný pro použití při trasování objektů. Algoritmus lze popsat jako průměrování odchylek naměřených hodnot od odhadovaných a jejich nejistot, pravděpodobností, že naměřená hodnota je správná. [11] Pokud se snažíme určit polohu a směr letu UAV prostředku, v určitých měřících okamžicích se zjišťují její okamžité polohy s nějakou přesností (chybou). Ve vztahu ke stabilizaci multikoptér se jedná o kmitání multikoptéry kolem cílového bodu. Toto kmitání je způsobeno zpožděním mezi senzory a akčními členy systému, motory multikoptéry.

[1] estimace - kvalifikované odhadování na základě zkoumání, šetření a strukturování dané problematiky

2.6 Shrnutí

Řešení konkrétních případů autonomních letů je pokaždé specifický problém. Na základě informací z dostupných projektů bylo navrženo schéma, jak by mohl obecně celý systém pro autonomní let vypadat (Obrázek 2-4). Schéma bylo vytvořeno na základě průniků literatury [2] a především problematik projektů [3] a [4].



Obrázek 2-4: Návrh schématu systému pro autonomní let

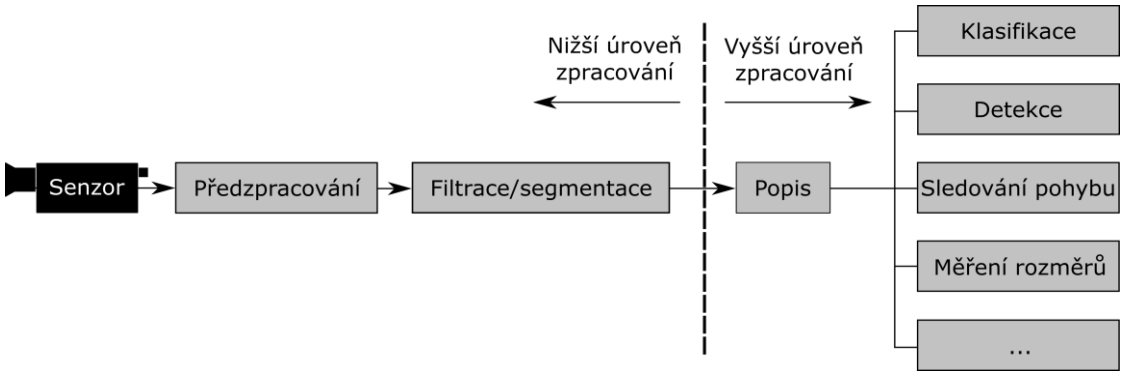
3 Technologie

3.1 Počítačové vidění, Kinect, OpenCV

Počítačové vidění (Computer Vision) je termín pro odvětví výpočetní techniky, které se zabývá vývojem zařízení a algoritmů schopných analyzovat data a získávat informace z digitálně zachyceného obrazu. Počítačové vidění spolupracuje a prolíná se s mnoha obory jako je například robotika, umělá inteligence, fyzika, optika a neurobiologie. V nich také hledá inspiraci pro řešení svých úloh. Snahou je se co nejvíce přiblížit k lidskému vidění, často i nad jeho rámec. Součástí je také usilování o co nejlepší popis objektů vyskytujících se v obraze pomocí technických prostředků. Počítačové vidění je oborem relativně novým, ale je již několik let prudce na vzestupu. První pokusy o analýzu obrazu začali v pozdních 60. letech, výpočetní výkon v té době nebyl dostupný ani dostačující, bylo obtížné pracovat i s malými sadami obrazových dat, a tak výzkum probíhal pouze na univerzitách. To dnes ale neplatí a díky tomu se každý rok posouvají možnosti počítačového vidění na novou úroveň. [13]

3.1.1 Obecný postup řešení

Organizace systému, který řeší úlohu počítačového vidění je vysoce závislá na typu problému. Některé systémy jsou plně samostatné aplikace řešící jeden daný problém a některé mohou být subsystémem systému většího a sofistikovanějšího. Dále je také důležité, zda je jeho funkce předem známa, nebo jestli je potřeba, aby byl systém učenlivý, nebo zda bude obraz zpracováván v reálném čase či nikoliv. Na Obrázek 3-1 je popsán obecný postup řešení úlohy počítačového vidění. [14]

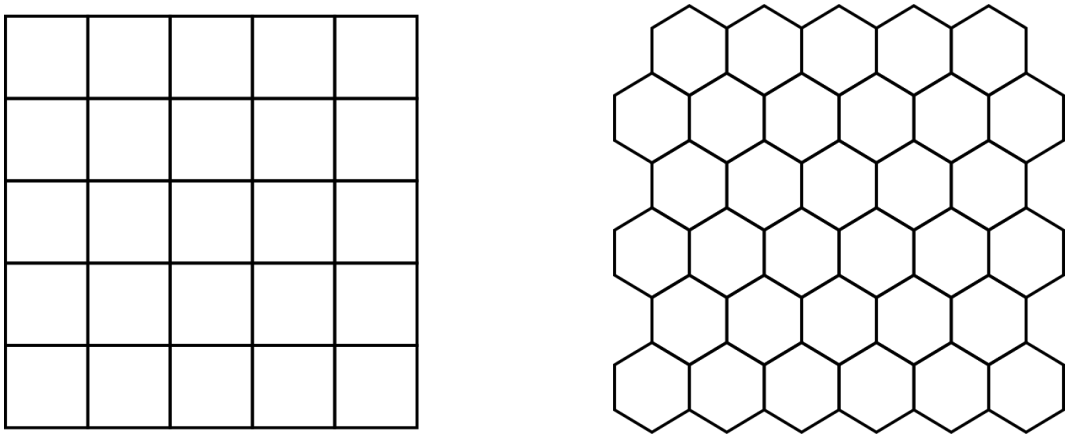


Obrázek 3-1: Schéma zpracování obrazu

Získání obrazu

Základem pro počítačové vidění je pořízení obrazových dat v co nejlepší možné kvalitě. Jde o proces zachycení 3D scény reálného světa do 2D obrazu. Může tak být učiněno pomocí kamery nebo fotoaparátu. Pro další zpracování je potřeba digitalizovaný ekvivalent zachyceného snímku. Digitalizace obrazu je proces vzorkování obrazové funkce do matice o rozměrech AxB (rozlišení) a kvantování spojité jasové úrovně každého vzorku do intervalu X (barevnost). Čím vyšší je rozlišení AxB a větší interval X, tím je původní obraz věrněji reprezentován. [14]

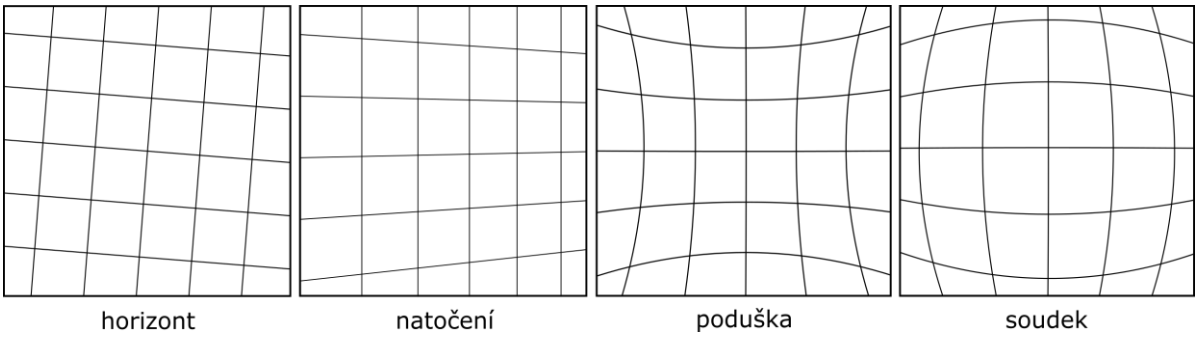
Důležitou součástí digitalizace je volba vzorkovací mřížky. Vzorkovací mřížka je uspořádání bodů při vzorkování spojitého obrazu. Používají se mřížky z pravidelných mnohoúhelníků, nejčastěji čtyřúhelníků (čtvercová) a šestiúhelníků (hexagonální). Čtvercová mřížka se používá častěji, jelikož je snadno realizovatelná. Hexagonální vzorkovací mřížka se používá u obrazů, u kterých je třeba měřit vzdálenosti nebo spojitosti objektů. [14]



Obrázek 3-2: Čtvercová a hexadecimální vzorkovací mřížka

Předzpracování obrazu

Předzpracování obrazu slouží k potlačení šumu a zkreslení vzniklého při pořízení, digitalizaci, přenosu obrazu a podmínkách při průběhu. Jako příklad lze uvést korekce horizontu, natočení, zakřivení zemského povrchu, nebo zdůraznění reprezentačního rysu obrazu. [14]



Obrázek 3-3: Příklady zkreslení

Filtrace obrazu

Cílem filtrace je zvýraznit nebo potlačit některé vlastnosti obrazu. Základní úlohou filtrace je vyhlazování šumu v obraze. Pro filtrování se využívají matematické operace jako je konvoluce, průměrování, medián nebo jejich kombinace. [14]

Segmentace obrazu

Segmentace je jedna z nejdůležitějších částí zpracování obrazu. Cílem je rozdělení obrazu na části (segmenty), které jsou nebo mohou být relevantní pro další zpracování a vyřešení dané úlohy. Výstupem segmentace by měl být soubor oblastí, které odpovídají objektům ve vstupním obraze. Jedná se o takzvanou kompletní segmentaci, ta však vyžaduje vyšší úrovně zpracování, a tak výstupem bývá pouze segmentace částečná. Ta je založena na principech homogenity obrazových vlastností (jas, barva) uvnitř segmentu nebo detekci hran. [14]

Vysokoúrovňové zpracování

U tohoto kroku bývá vstupem velmi malé množství dat, u kterých se například předpokládá výskyt hledaného objektu. Do tohoto kroku patří popis obrazu a klasifikace objektů. [14]

- **Popis obrazu** je popisem objektů z předchozího kroku segmentace. Dělí se na dva základní způsoby. Kvantitativní způsob je založen na popisu objektů pomocí souboru číselných hodnot, jako je velikost objektu nebo kompaktnost. Další možností je kvalitativní způsob, ve kterém jsou popisovány relace (vztahy) mezi objekty a jejich tvarové vlastnosti. Ve většině případů je popis obrazu vstupní informací pro klasifikaci objektů.

- **Klasifikace** objektů je závěrečným krokem, jedná se o zařazení objektů nalezených v obraze do skupin předem známých tříd.

3.1.2 Typické úlohy

Rozpoznávání

Určení, zda daná obrazová data obsahují specifický objekt je jedním z klasických úloh počítačového vidění. Vytvořené algoritmy bývají navrženy pro specifické úlohy, kdy algoritmus musí znát, nebo se musí být schopen se naučit specifické vlastnosti objektu hledaného v obrazových datech. [14] Tyto úlohy je možné dále rozdělit na:

- **Poznávání:** specifický objekt, obličeje, siluety osob, písmo, barvy
- **Identifikace:** komu konkrétnímu patří tento obličej, otisk prstu, poznávací značka
- **Detekce:** zjišťování výskytu konkrétní podmínky

Analýza pohybu

Tato úloha kombinuje úlohu rozpoznávání a následného porovnávání pozic nalezeného objektu v předchozích a následujících obrazových dat z videa. Algoritmus musí na každém jednotlivém snímku detekovat objekt a zjistit jeho polohu. Pak jsou polohy objektu porovnávány s předchozími a následujícími snímky z čehož lze vyvodit například směr nebo trasu. [14]

Rekonstrukce

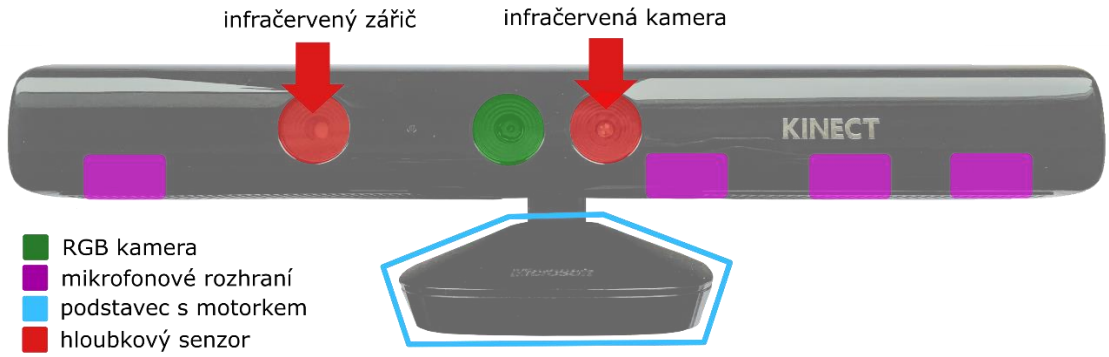
- **Rekonstrukce obrazu** slouží k odstranění nežádoucích jevů jako je šum, zrnitost a rozmazání z obrazů. Používá se i jako před proces pro zlepšení kvality obrazu před samotným zpracováním.
- **Rekonstrukce scény** se zajímá o rekonstrukci 3D modelu scény z jednoho nebo více obrazů zachycující danou scénu a to i z různých pozic.

3.1.3 Microsoft Kinect for Windows

První generace zařízení Kinect byla představena společností Microsoft v roce 2010 jako doplňující prvek ke konzoli Xbox360. Účelem tohoto zařízení bylo umožnit ovládání her pomocí gest a hlasu. Netrvalo dlouho a začaly se objevovat neoficiální ovladače, které umožňovaly připojit Kinect konstruovaný pro Xbox k PC. Microsoft vydal v roce 2011 oficiální ovladače a v roce 2012 byl uveden Kinect for Windows (Obrázek 3-4), který nebyl nic jiného než původní Kinect optimalizovaný pro PC. V roce 2013 s novou konzolí Xbox One Microsoft vydal novou verzi Kinectu, Kinect for Xbox One, taktéž podporující připojení k PC pomocí adaptéru.

Obrázek 3-4: Kinect for Windows (2012)

Základními prvky Kinectu umístěnými uvnitř pouzdra (Obrázek 3-5) jsou RGB kamera, hloubkový senzor, mikrofonové rozhraní a akcelerometr. V podstavci Kinectu je umístěn motorek, který umožnuje celému zařízení naklápění ve vertikálním směru v rozmezí ±27◦. [16]



Obrázek 3-5: Prvky Kinectu

RGB kamera u Kinectu slouží k pořizování barevných snímků a videa. Kamera Kinectu v1 defaultně natáčí v rozlišení 640x480 pixelů při 30 fps[2]. RGB kamera dovoluje také natáčení při rozlišení 1280x960 pixelů, ale pouze při 12 fps a tak se hodí spíše pro pořizování samostatných snímků než videa. Zařízení Kinect také disponuje mikrofonovým rozhraním, které tvoří čtveřice mikrofonů umístěných vedle sebe. Díky rozmístění a většímu počtu mikrofonů je možné zjistit směr a potlačit nežádoucí šum nebo ozvěnu zvuku. Kinect lze tedy použít pro ovládání pomocí hlasu a experimentování se zvukem. [17]

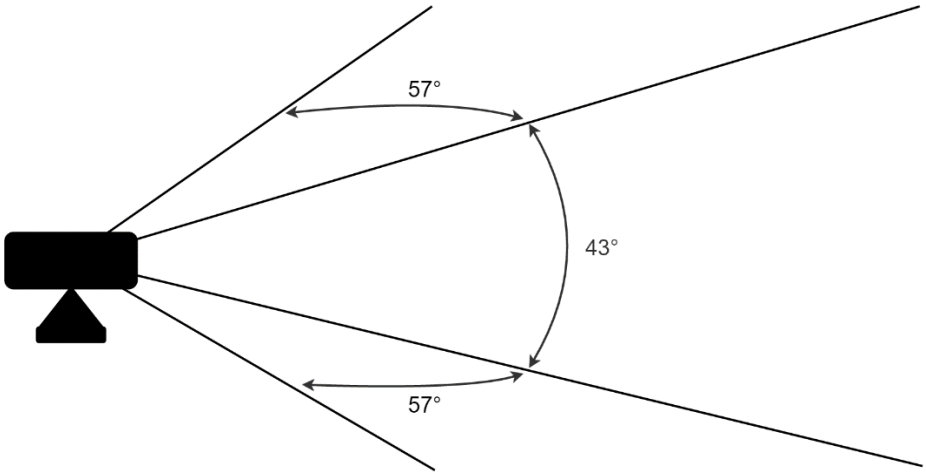
	Kinect v1	**Kinect v2**
RGB kamera	640x480 30 fps 1280x960 12 fps	1920x1080 30 fps
Hloubkový senzor	320x240 30 fps	512x424 30 fps
Maximální hloubka	~4,5 m	~4,5 m
Minimální hloubka	40 cm	50 cm
Horizontální zorné pole	57 °	70 °
Vertikální zorné pole	43 °	60 °
Motorek v podstavci	ano	ne
Skeleton - počet kloubů	20 kloubů	26 kloubů
Maximální počet sledovaných skeletonů	2	6
USB standardy	2.0	3.0
Akcelerometr	ano	ano

Tabulka 3-1: Porovnání Kinectu v1 a v2 [15]

[2] fps (Frames Per Second) - číslo udávající počet snímaných snímků za sekundu

Scéna

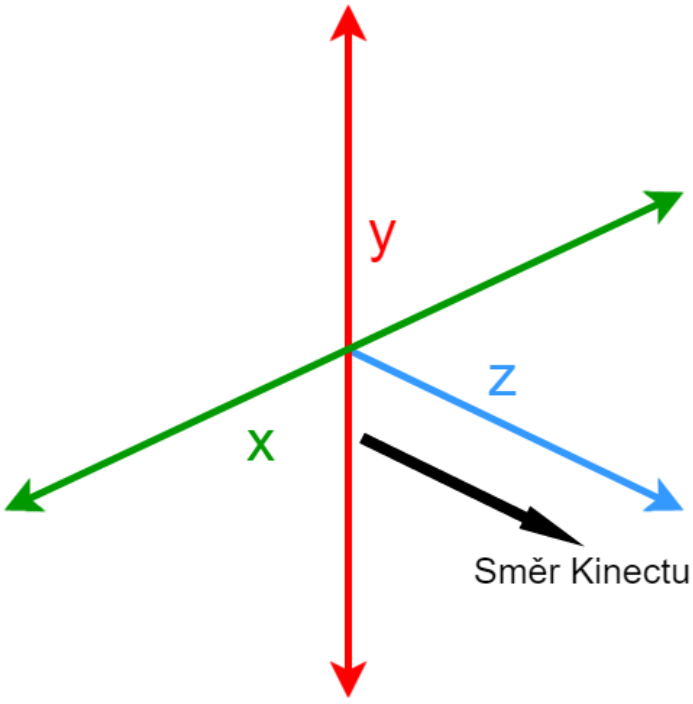
Scéna je prostor před senzorem Kinect, kde mají infračervené a barevné senzory neblokovaný výhled na vše před senzorem. Scéna Kinectu je dána zorným polem, které je 43° vertikálně a 57° horizontálně (Obrázek 3-6). [16]



Obrázek 3-6: Scéna Kinectu

Akcelerometr

Senzor Kinect obsahuje tříosý akcelerometr nakonfigurovaný pro rozsah 2g, kde g je gravitační zrychlení. To senzoru umožňuje hlásit aktuální orientaci s ohledem na gravitaci. Data z akcelerometru také mohou pomoci zjistit, zda se senzor nachází v neobvyklé poloze. Pomocí akcelerometru lze určovat orientaci až s přesností na 1 stupeň. Přesnost je mírně závislá na teplotě, která je v normálních provozních teplotách až 3 stupně. Data z akcelerometru jsou ve formě 3D vektoru (Obrázek 3-7) směřujícího ve směru gravitace. [18]



Obrázek 3-7: Souřadnicový systém akcelerometru

Hloubkový senzor a triangulace

Hloubkový senzor se skládá z infračerveného zářiče (infrared projector) a infračervené kamery (infrared camera). Kinect využívá pro zjišťování vzdáleností takzvanou aktivní triangulaci (Obrázek 3-8). Základem aktivní triangulace je, že zářič a kamera jsou od sebe v pevně dané vzdálenosti zvané triangulační báze, jejíž velikost ovlivňuje vlastnosti senzoru, jako je například minimální a maximální možná snímatelná vzdálenost. Zářič vyšle infračervený paprsek pod předem známým úhlem a ten se odrazí od překážky do infračervené kamery a změří se úhel dopadu. Společně úhel, pod kterým byl paprsek vyslán, úhel dopadu a triangulační báze tvoří triangulační trojúhelník. Pokud jsou tyto hodnoty známy, je možné pomocí jednoduchého trigonometrického výpočtu zjistit vzdálenost objektu od senzoru. [19] [20]

B ... *triangulační báze*

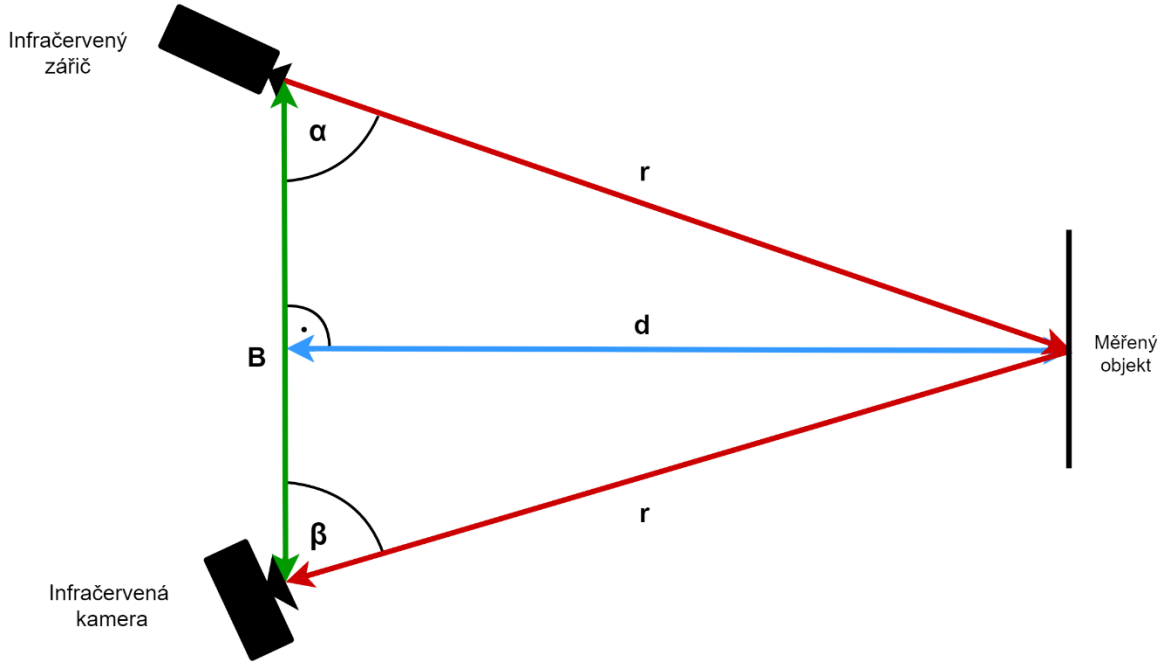
d ... *měřená vzdálenost*

r ... *infračervený paprsek*

α ... *úhel vyslání paprsku*

β ... *úhel dopadu paprsku*

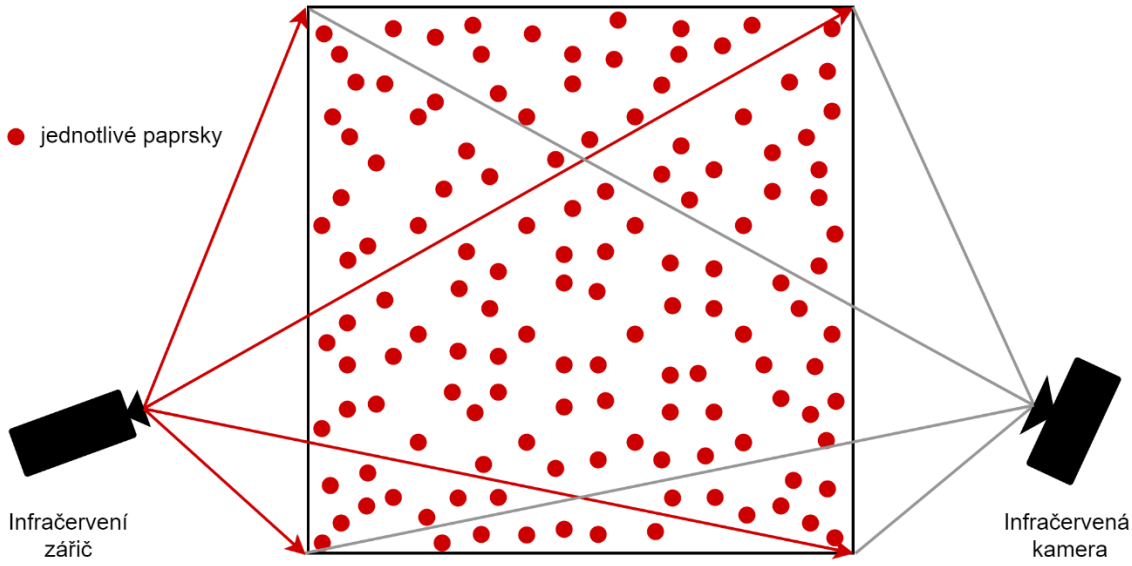
$$d = \frac{B \cdot \sin\beta \cdot \sin\alpha}{\sin(180° - (\alpha + \beta))}$$



Obrázek 3-8: Schéma aktivní triangulace

Pokud je umístěno několik paprsků do řady vedle sebe, vznikne složitější dvourozměrná (2D) triangulace. Ta snímá vzdálenosti v pásu paprsků, díky tomu je možné vytvořit dvourozměrný model. Pokud je ale možné paprsky plynule přejíždět po nehybném předmětu, nebo předmětem pod paprsky otáčet, je možné vytvořit trojrozměrný model předmětu. Této techniky se často využívá u 3D skenování.

Senzor Kinect využívá techniky trojrozměrné (3D) aktivní triangulace (Obrázek 3-9). Zářič neprojektuje pouze jeden paprsek nebo pás paprsků, ale síť paprsků s předem danou strukturou. Každý z těchto paprsků je samostatně snímán kamerou a je určena vzdálenost (Obrázek 3-8). Pokud jsou veškeré naměřené vzdálenosti umístěny do matice, vznikne hloubkový (depth) snímek scény. [19]



Obrázek 3-9: Trojrozměrná triangulace

3.1.4 OpenKinect (libfreenect)

OpenKinect je otevřená komunita lidí, kteří se zajímají o využívání Kinect hardwaru s klasickými počítači a dalšími zařízeními. Pracují na svobodných open-source knihovnách, které umožňují použít senzoru Kinect s operačními systémy Windows, Linux a Mac OS. Cílem této komunity s více než 2000 členy je vytvořit co nejlepší sadu aplikací pro Kinect. Primárním zaměřením je vývoj balíčku (knihovna a ovladač) libfreenect. Tento balíček obsahuje OpenNI2 ovladač a libfreenect knihovnu pro Kinect.

Webová stránka:	www.openkinect.org
Ke stažení:	https://github.com/OpenKinect/libfreenect
Operační systém:	cross-platform
Programovací jazyk:	C++ (nativní), C#, Python, Java, JavaScript a další
Licence:	Apache20, GPL2 (nepovinné)

3.1.5 OpenCV

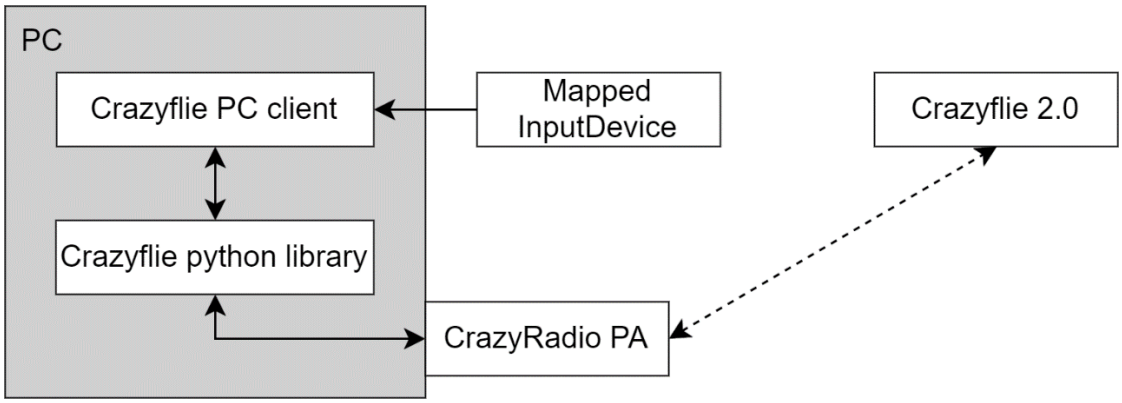
OpenCV (Open Source Computer Vision library) je open-source knihovna počítačového vidění a strojového učení vycházející pod BSD[3] licencí. Byla vytvořena s cílem zajistit společnou infrastrukturu pro aplikace počítačového vidění a urychlit využívání v komerčních produktech. Původně byla vyvíjena společností Intel. Dnes se na vývoji a využívání knihovny OpenCV podílí několik velkých společností, jako Google, Yahoo, Microsoft, Intel, IBM, Sony, Honda a Toyota.

Webová stránka:	www.opencv.org
Ke stažení:	https://github.com/opencv/opencv
Operační systém:	cross-platform
Programovací jazyk:	C/C++ (nativní), Python, Java, MATLAB
Licence:	BSD

3.2 Vývojová platforma Crazyflie

Sada Crazyflie je projektem švédské skupiny Bitcraze, kterou v roce 2011 založili Arnaud Taffanel, Marcus Eliasson a Tobias Antonsson. První myšlenky už ale sahaly do roku 2009, kdy se tito tři zakladatelé pustili ve svém volném čase do vývoje sady Crazyflie. Později se k původním třem zakladatelům přidali Björn Mauritz a Kristoffer Richardsson. Tato pětice tvoří dnešní sestavu skupiny Bitcraze. Bitcraze jsou především nadšenci do hardwaru, softwaru a moderních technologií obecně. Hlavní myšlenkou skupiny je umožnit lidem objevovat, zkoumat, inovovat a vzdělávat se, proto jsou také veškeré jejich produkty open source.

[3] BSD (Berkeley Software Distribution) – je licence pro svobodný software, umožňuje volné šíření obsahu



Obrázek 3-10: Základní zapojení platformy Crazyflie

Základními produkty sady Crazyflie jsou nano-kvadrokoptéra Crazyflie, rádio vysílač CrazyRadio, softwarový PC klient a python knihovnu (API) pro komunikaci s Crazyflie prostřednictvím CrazyRadia. Dále je možné sadu rozšířit o několik modulů jako je prototypovací deska, deska pro mikroSD karty nebo desku, která umožňuje sestrojit vlastní kvadrokoptéru s kontrolérem Crazyflie.

Webová stránka: www.bitcraze.io

Licence: BSD, open hardware, open-source firmware/software

3.2.1 Crazyflie 2.0

Crazyflie 2.0 je nástupce úspěšné kvadrokoptéry Crazyflie Nano Quadcopter (Crazyflie 1.0). Oproti staršímu předchůdci má navíc Bluetooth LE[4], který usnadňuje létání ovládané prostřednictvím mobilních zařízení. [21]

[4] Bluetooth LE (Low Energy) – bluetooth s nízkou spotřebou energie

Obrázek 3-11: Crazyflie 2.0

Specifikace

Mechanické vlastnosti:

- Váha: 27g
- Rozměry: 92x92x29mm

Rádio specifikace:

- 20 dBm rádio zesilovač, dosah až 1 km s CrazyRadio PA při LOS[5]
- Bluetooth Low Energy: podpora iOS a Android klienta (testované na iOS 7.1+ a Android 4.4+)
- Zpětně kompatibilní s Crazyflie 1.0 a CrazyRadio

Mikrokontroléry (MCU):

- STM32F405 hlavní MCU (Cortex-M4, 168MHz, 192kb SRAM, 1Mb flash)
- nRF51822 rádio a napájení MCU (Cortex-M0, 32Mhz, 16kb SRAM, 128kb flash)
- uUSB konektor
- Na desce LiPo nabíječka s 100mA, 500mA a 980mA dostupnými režimy
- Schopnost částečného USB OTG

[5] LOS (Line of sight) – přímá viditelnost

Inerciální měřící jednotky (IMU)

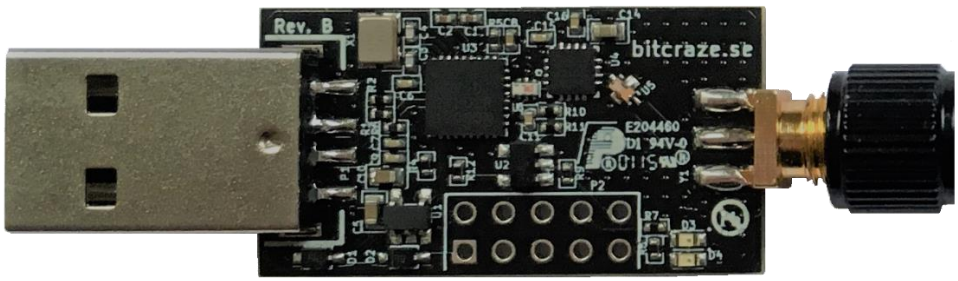
- MPU-9250
    - Tříosý gyroskop
    - Tříosý akcelerometr
    - Tříosý magnetometr
- Senzor tlaku s vysokou přesností (LPS25H)

Specifikace letu:

- Doba letu s původní baterií: 7 minut
- Doba nabíjení původní baterie: 40 minut
- Maximální doporučená nosnost: 15 g

3.2.2 CrazyRadio PA

Crazyradio PA je rádiové USB zařízení, které primárně slouží ke komunikaci s firmwarem kvadrokoptéry Crazyflie, ale může být adaptován pro mnoho dalších zařízení a aplikací, které jsou založeny na low-cost 2,4 GHz čipech od firmy Nordic Semiconductor. PA v názvu značí, že součástí vysílače je výkonový zesilovač (power amplifier). Díky tomuto zesilovači může vysílat do vzdálenosti až 1 km. Komunikace s Crazyflie probíhá prostřednictvím CRTP (Crazyflie RealTime Protocol) protokolu. Každý CRTP packet je 32 bytový, z toho je 1 byte hlavička. Tím páde je celkový payload[6] 31 bytů na packet. Hlavička obsahuje port (4 bity), kanál (2 bity) a vyhrazené 2 bity. [22]



Obrázek 3-12: Crazyradio PA

[6] payload – část odeslaných dal, které obsahují skutečně zamýšlenou zprávu

Specifikace

Čip Nordic Semiconductor nRF24LU1+

- 8051 MCU do 16MHz s 32Kb flash a 2Kb SRAM paměťmi
- 2.4GHz ISM radiové pásmo
- 125 rádio kanálů
- 2Mbps, 1Mbps a 250Kbps komunikační rychlost přenosu dat
- vysílá a přijímá datové pakety až do velikosti 32 bytů
- automaticky zpracovává adresy a pakety ack
- hardware SPI a UART
- kompatibilní s rozšířeným ShockBurst protokolem od Nordic Semiconductor

Rádio specifikace:

- 20dBm výstupní výkon (100mW)
- Low Noise Amplifier (LNA)
- konektor RP-SMA

Mechanické vlastnosti:

- váha: 6g
- rozměry: 58x16x6.5mm (včetně konektorů)
- standartní USB-A konektor

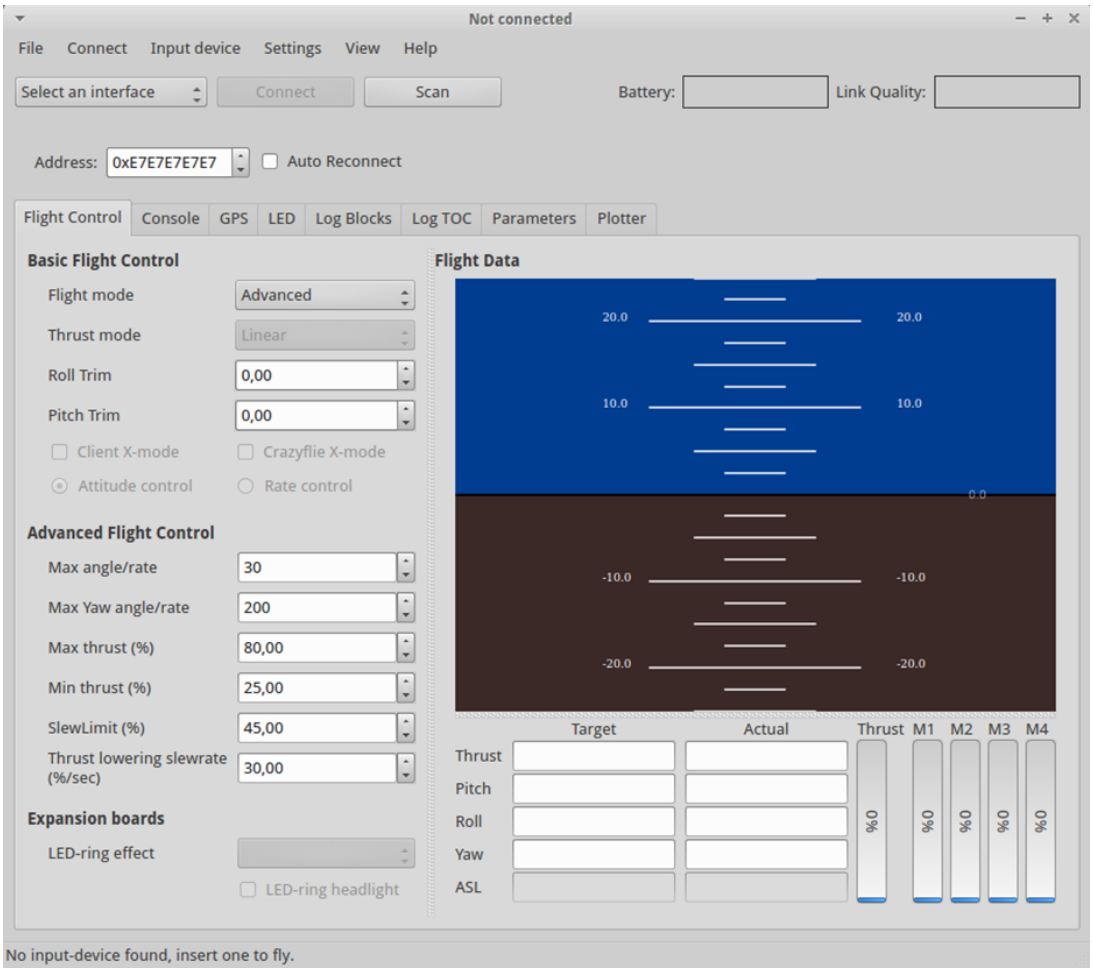
3.2.3 Crazyflie python library (cflib)

Crazyflie python library je API napsána pro používání a ovládání kvadrokoptéry Crazyflie 1.0 a 2.0. Je určena pro použití v aplikacích ke komunikaci a řízení Crazyflie kvadrokoptér. Knihovnu například využívá. Crazyflie PC client. Knihovna je založena na asynchronních a zpětných voláních pro události. Funkce jako *open_link* se okamžitě vrátí, a zpětné volání *connected* je voláno, pokud je spojení otevřeno. Knihovna neuchovává žádná vlákna nebo zámky, které udrží chod aplikace, to je ponecháno na aplikaci, která knihovnu využívá.

Webová stránka:	https://wiki.bitcraze.io/doc:crazyflie:api:python:index
Ke stažení:	www.github.com/bitcraze/crazyflie-lib-python
Operační systém:	cross-platform

3.2.4 Crazyflie PC client (cfclient)

Crazyflie PC client využívá knihovny Crazyflie pro komunikaci s kvadrokoptérami Crazyflie skrze Crazyradio (PA). Slouží k ovládání, nahrávání firmwaru, nastavování parametrů, zápisu dat, získávání dat a informuje o důležitých vlastnostech kvadrokoptér Crazyflie v reálném čase. Hlavní uživatelské prostředí je sestaveno z řady karet (Obrázek 3-13), kde jsou jednotlivé karty používány pro specifické funkce.



Obrázek 3-13: Crazyflie client UI[7]

Webová stránka:	www.wiki.bitcraze.io/doc:crazyflie:client:pycfclient:index
Ke stažení:	www.github.com/bitcraze/crazyflie-clients-python
Operační systém:	cross-platform
Programovací jazyk:	Python
Licence:	GNU GPL

[7] UI (user interface) – uživatelské rozhraní

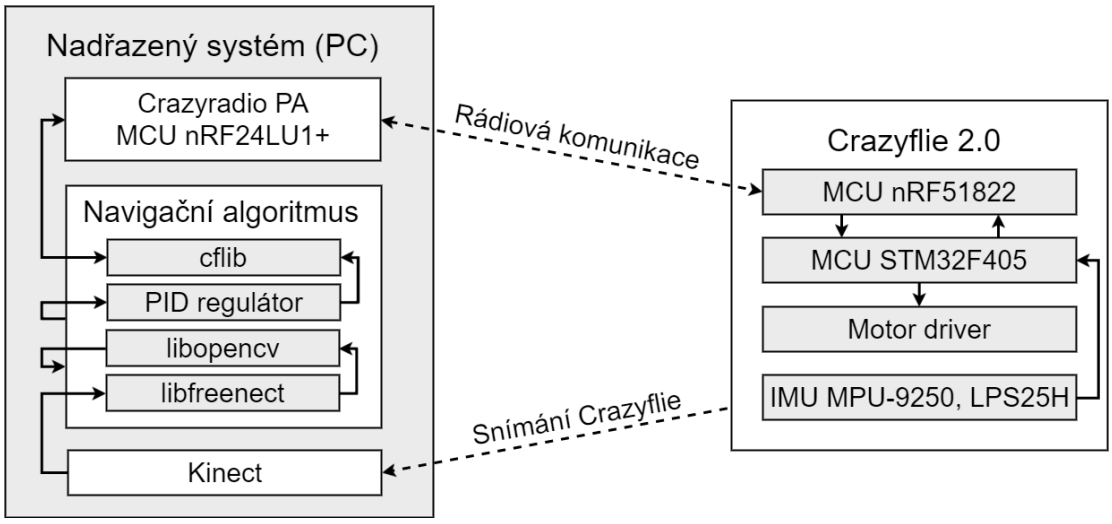
3.2.5 ZeroMQ (libzmq)

ZeroMQ (ØMQ) je open-source knihovna určená pro vysokorychlostní asynchronní komunikaci (messaging) mezi programovacími jazyky. Umožňuje rychlé a jednoduché propojení kódů napsaných v odlišných programovacích jazycích. Umožňuje použít vhodný programovací jazyk pro části aplikace, které nejsou pomocí primárního jazyku řešitelné (chybí knihovny) nebo velmi obtížně (syntaxe). První ZeroMQ proces (např. C++) odesílá zprávy skrze port, na kterém jeden nebo více dalších procesů (jiný programovací jazyk, Python, Java atd.) odposlouchává. Tyto další procesy pak můžou vysílat zprávy na jiné porty a tím si vyměňovat informace.

Webová stránka:	www.zeromq.org
Ke stažení:	www.github.com/zeromq/libzmq
Operační systém:	cross-platform
Programovací jazyk:	C (nativní), C++, C#, Python, Java, JavaScript, PHP a další
Licence:	GNU LGPL

3.3 Zapojení technologií

Následující schéma (Obrázek 3-14) ukazuje zapojení jednotlivých technologií do systému.



Obrázek 3-14: Schéma zapojení technologií do systému

4 Teoretické řešení

4.1 Teoretický návrh

4.1.1 Vývojová platforma Crazyflie

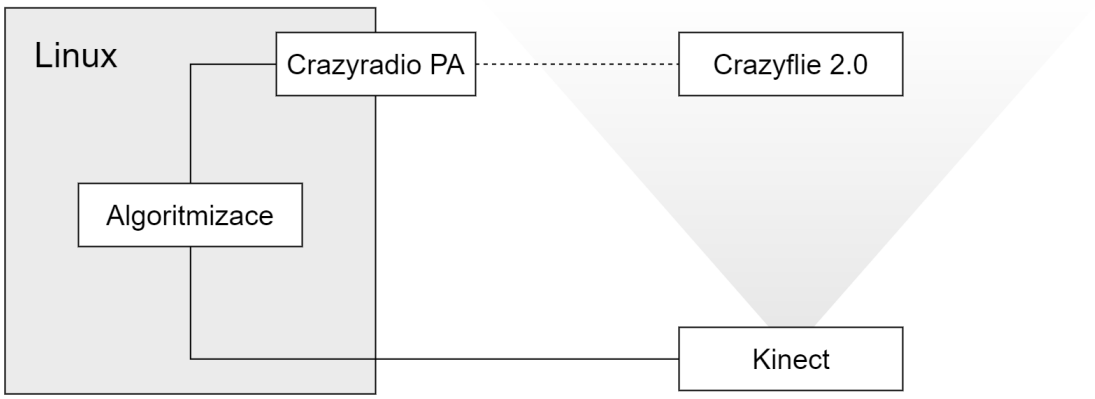
Platforma Crazyflie sama od sebe řeší řídící subsystém kvadrokopéry, který je v řežimu **SLAVE.** Crazyflie 2.0 se samo stará o vyrovnávání a stabilizaci pomocí svých senzorů (gyro, magnetometr, akcelerometr), ale ostatní pokyny (pitch, roll, yaw, thrust) přicházejí z nadřazeného systému, přicházející pomocí rádiového spojení přes Crazyradio. Takovým nadřazeným systémem je Crazyflie klient nebo jiná aplikace generující řídící parametry pro Crazyflie.

4.1.2 Nadřazený systém

Navigační a senzorický subsystém

Základem navigace je vytvoření globálního souřadného systému (x, y, z), který je realizován pomocí senzoru Kinect. Souřadnice (x, y) jsou definovány scénou RGB kamery a souřadnice (z) Depth snímkem scény. Pro detekci Crazyflie v obraze je potřeba vytvořit algoritmus, který neustále analyzuje obraz a získává pozici (x, y). Pozice (z) je v příslušném bodě (x, y) zjištěna z Depth snímku scény.

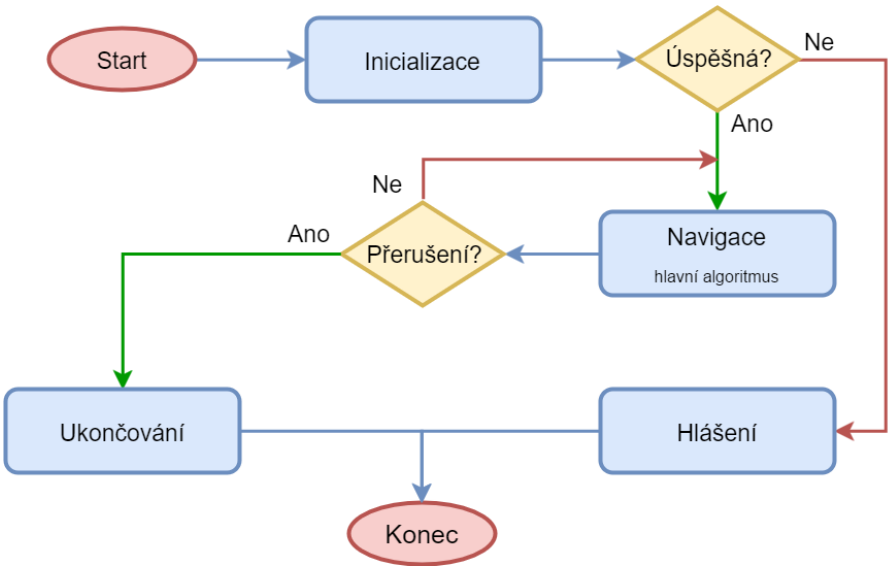
Aby bylo možné Crazyflie přemístit ze současné pozice do cílové je potřeba vytvořit algoritmus. Ten na základě současné a cílové pozice generuje řídící parametry, které jsou pomocí Crazyradia odeslány do Crazyflie.



Obrázek 4-1: Schéma zapojení prvků

4.2 Algoritmizace

Algoritmizace je nejdůležitější část této práce, závisí na ní přesnost a funkčnost celého nadřazeného systému. Při návrhu bylo vycházeno z projektů [3] a [4]. V diagramu (Obrázek 4-2) je popsán základní průběh programu, navigační algoritmus je rozebrán v následující kapitole 4.2.1.



Obrázek 4-2: Vývojový diagram – celý program

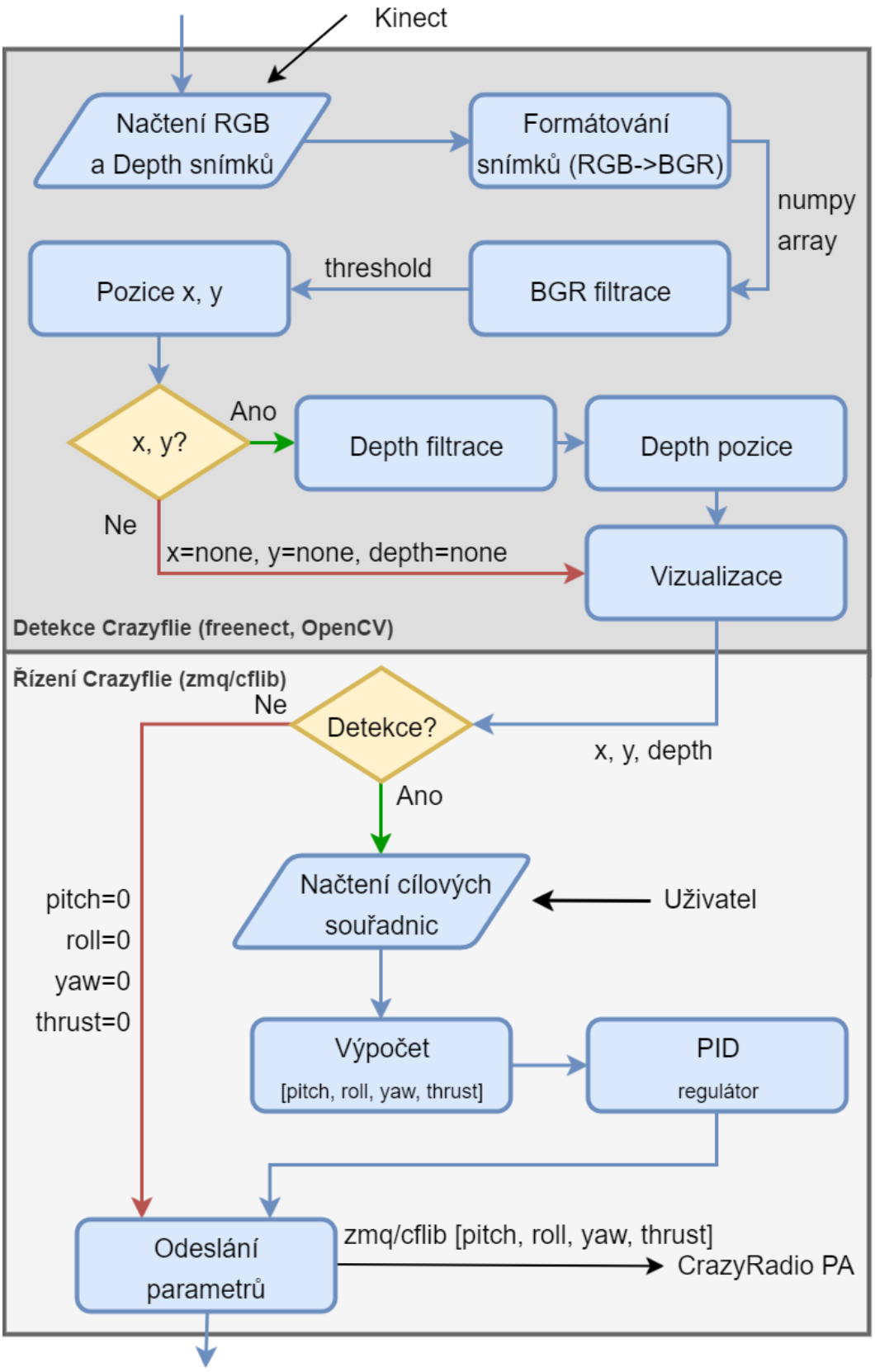
Komunikace s Crazyflie 2.0

Komunikace mezi Crazyflie a navigačním programem probíhá prostřednictvím Crazyradia PA. Pro spojení se nabízí možnost využití zmq vstup Crazyflie klienta, který je připojen ke Crazyflie a navigační program komunikuje prostřednictvím zmq packetů s klientem. Další možností je využití Crazyflie python API a navázání spojení umožnit přímo v programu. Je důležité zmínit, že možnost pomocí zmq a klienta nemá připravené rozhraní pro získávání údajů o orientaci (pitch, roll, yaw) z Crazyflie. Tento problém je řešen v následujících kapitolách.

Inicializace a hlášení

Po spuštění programu jsou inicializovány a kontrolovány potřebné moduly spolu se zavedením inicializačních proměnných a objektů. Pokud při inicializaci nastane problém, je nahlášen a následně je program ukončen. Při přerušení hlavního navigačního algoritmu jsou před ukončením celého programu uvolněny zavedené prostředky a následně je ukončen celý program.

4.2.1 Navigační algoritmus



Obrázek 4-3: Vývojový diagram - navigační algoritmus

Diagram (Obrázek 4-3) naznačuje předpokládaný průběh navigačního algoritmu, ten lze rozdělit na dvě základní části, detekci a řízení. Detekční algoritmus analyzuje obraz ze senzoru Kinect a pokud zjistí výskyt Crazyflie v obraze zjistí jeho pozici. Ta je předána řídícímu algoritmu, který na základě znalosti současné a cílové pozice Crazyflie vypočítá řídící parametry a odešle je pomocí zmq do Crazyflie klienta nebo přímo pomocí Crazyflie API.

4.2.2 Detekce Crazyflie v obraze (x, y)

Detekce probíhá na základě obecného řešení počítačového vidění (viz 3.1.1). Knihovna OpenCV umožňuje několik způsobů, jak detekovat objekt v reálném čase. Nejrychlejší variantou, jak detekovat objekt je umístění marky, například v podobě výrazně barevného míčku na konstrukci Crazyflie a následně tuto barvu hledat v obraze. Dalším možným, ale časově náročnějším na přípravu je způsob vytvoření vlastního klasifikátoru hledaného objektu.

Předzpracování/získání obrazu

OpenCV používá jako vstup BGR barevný model a od verze 3 používá NumPy knihovnu pro práci s maticemi obrazu a numerickými daty. Před samotnou analýzou obrazu je tedy důležité převést data získaná z Kinectu na BGR a NumPy formát. K tomu slouží wrapper[8] `frame_convert2.py` dostupný v knihovně freenect. Důležité je používat synchronní obrazy, to znamená, že OpenCV bude pracovat s RGB a Depth obrazy pořízenými ve stejný čas, jako příklad slouží script[9] `demo_cv2_sync.py`.

Filtrace/segmentace BGR

OpenCV nabízí mnoho možností, jak filtrovat obraz. Při potřebě detekovat určitou barvu v obraze se převádí reprezentace barev na HSV model, který umožňuje přesněji reprezentovat odstíny jedné barvy a následně je vyfiltrována hledaná barva v určitém intervalu. Výsledkem je matice obrazu, kde jsou jednotlivé složky true (1) nebo false (0), podle toho, zda se na jeho místě vyskytovala data v hledaném intervalu. Tato matice se nazývá „threshold“. Pro lepší reprezentaci ploch pro další zpracování může být snímek dále vyhlazován.

[8] wrapper – připravený kus kódu, který umožňuje kompatibilitu mezi dvěma rozhraními
[9] script – spustitelný soubor ve formě prostého textu s programem ve skriptovacím jazyce

Popis a detekce

V threshold matici je hledána největší plocha s hodnotami „true“. Po nalezení je zjištěn střed plochy, ten je reprezentován souřadnicemi (x, z). Plocha musí splňovat minimální hranici velikosti, aby nedocházelo k detekci malých ploch.

Cascade Classifier

Jak už bylo zmíněno další možností, jak detekovat Crazyflie v obraze je vytvoření vlastního klasifikátoru. Kaskádové klasifikátory jsou cvičeny několika stovkami pozitivních a několika tisícovkami negativních obrazů. Pozitivní obrazy obsahují hledaný objekt a negativní náhodné pozadí bez hledaného objektu. Vytvoření opravdu kvalitního klasifikátoru je časově náročná záležitost, kde je potřeba si dát práci s pořízením snímků a mít dostatečně výkonný výpočetní systém pro následné vytváření klasifikátoru pomocí strojového učení, kdy proces učení může trvat několik dní v kuse. I po dodržení správných postupů nemusí být výsledný klasifikátor dostačující a musí se začít od začátku. Proto se v případě práce jedná spíše o experiment, zda by se dal tento způsob využít.

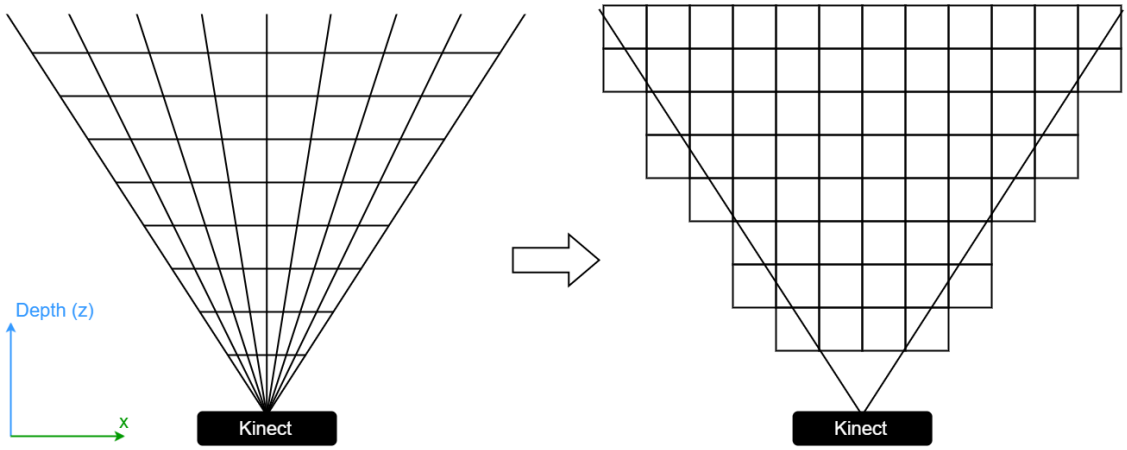
4.2.3 Depth (z) souřadnice

Na začátek je důležité poznamenat, že souřadnice z a depth jsou jedna stejná souřadnice. Poté co je pomocí analýzy obrazu zjištěna pozice (x, y). Je možné pomocí nich zjistit přibližnou polohu v depth snímku. Depth snímek je reprezentován maticí naplněnou 11bitovými číslicemi reprezentující jednotlivé vzdálenosti. Tuto reprezentaci (*raw*) lze pomocí jednoduchého vzorce přepočítat na centimetry. [27]

$$vzdálenost = \left(0.1236 * \tan\left(\frac{raw}{2842.5} + 1.1863\right)\right) * 100$$

4.2.4 Deformace souřadného systému

Díky deformaci souřadného systému není souřadnice (x, y) ve snímku pokaždé stejná pozice. Pozice x, y je ovlivněna souřadnicí (z) (Obrázek 4-4), kde dva body stále ve stejné reálné vzdálenosti budou v jednotlivých souřadnicích (z) vykazovat jiné vzájemné vzdálenosti. Tento problém je možné řešit algoritmem, který přepočte souřadnice z deformovaného do reálného souřadného systému, kde je díky relativně přesné (z) souřadnici v centimetrech možné přepočítat všechny souřadnice do stejných jednotek. Reálné souřadnice slouží pouze k výpočtu řídících parametrů, při určování cílových souřadnic a při vizualizaci se nadále používají zkreslené souřadnice. Jedná se o vlastní návrh.



Obrázek 4-4: Deformace souřadného systému

Jako první krok je nejprve třeba zjistit v jaké vzdálenosti depth (z) jsou y-nové souřadnice nezkreslené, ze kterých se následně může vycházet. Tento údaj jde zjistit ze vztahu depth (z), rozlišení obrazu a úhlu zorného pole (viz Obrázek 3-6). Dále je možné určit nové počáteční body pro lepší orientaci v souřadnicovém systému tím, že nulové souřadnice osy *x* a *y* budou uprostřed scény.

α ... *horizontální zorné pole Kinectu*

β ... *horizontální zorné pole Kinectu*

$$\alpha = 57°$$

$$\beta = 43°$$

x_{max} ... *horizontální složka rozlišení obrazu*

y_{max} ... *vertikální složka rozlišení obrazu*

$$x_{max} = 640$$
$$y_{max} = 480$$

$Depth_{0x}$ ... *hloubka nezkresleného x*

$Depth_{0y}$ ... *hloubka nezkresleného y*

$$depth_{0x} = \frac{\frac{x_{max}}{2}}{\tan\left(\frac{\alpha}{2}\right)} = \frac{320}{\tan(28{,}5)} = 589$$

$$depth_{0y} = \frac{\frac{y_{max}}{2}}{tan\left(\frac{\beta}{2}\right)} = \frac{240}{\tan(21{,}5°)} = 481$$

Velikost zkreslení je dána úhlem zkreslení a vzdáleností bodu od nezkreslených souřadnic *x* a *y* (Obrázek 4-5). Reálné souřadnice pak lze vypočítat rozdílem velikosti zkreslení a zkreslených souřadnic.

ω, φ ... *úhel zkreslení*

r_x, r_y ... *velikost zkreslení*

x_{ar}, y_{ar} ... *reálné souřanice (cm)*

Výpočet reálné souřadnice x_{ar}:

$$\omega = \tan\left(\frac{x_a - 320}{depth_{0x}}\right)$$

$$r_x = \tan^{-1}(\omega) \cdot (depth_{0x} - depth_a) = \frac{x_a - 320}{Depth_{0x}} \cdot (depth_{0x} - depth_a)$$

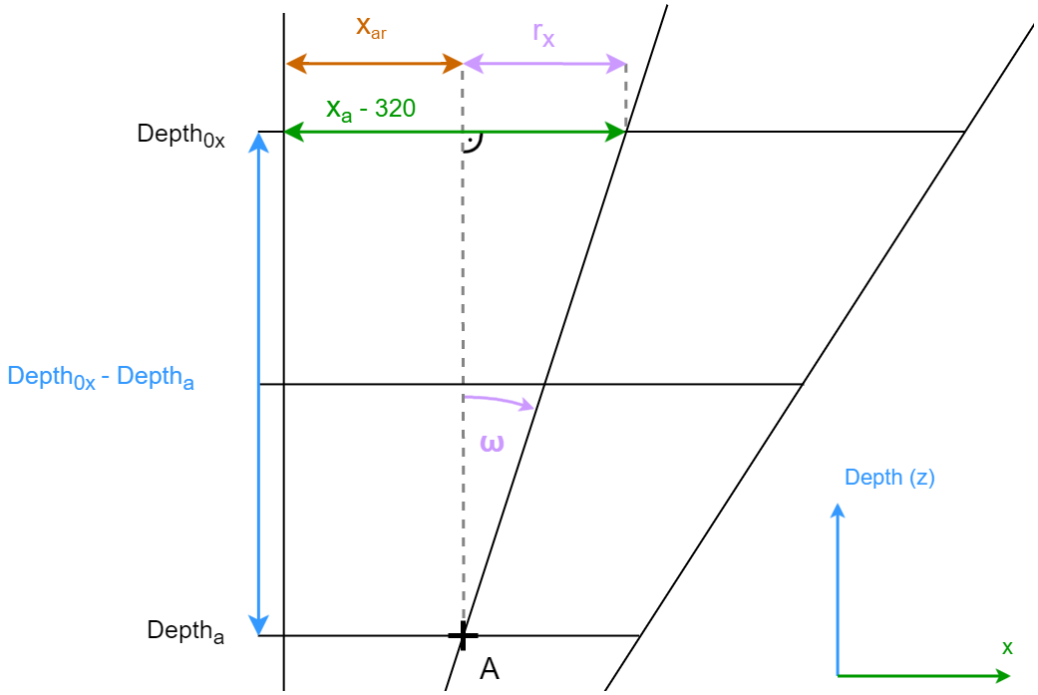
$$x_{ar} = x_{ar0} - r_x - 320$$

Výpočet reálné souřadnice y_{ar}:

$$\varphi = \tan\left(\frac{y_a - 240}{Depth_{0y}}\right)$$

$$r_y = \tan^{-1}(\varphi) \cdot \left(depth_{0y} - depth_a\right) = \frac{y_a - 240}{depth_{0y}} \cdot \left(depth_{0y} - depth_a\right)$$

$$y_{ar} = y_{ar0} - r_y - 240$$

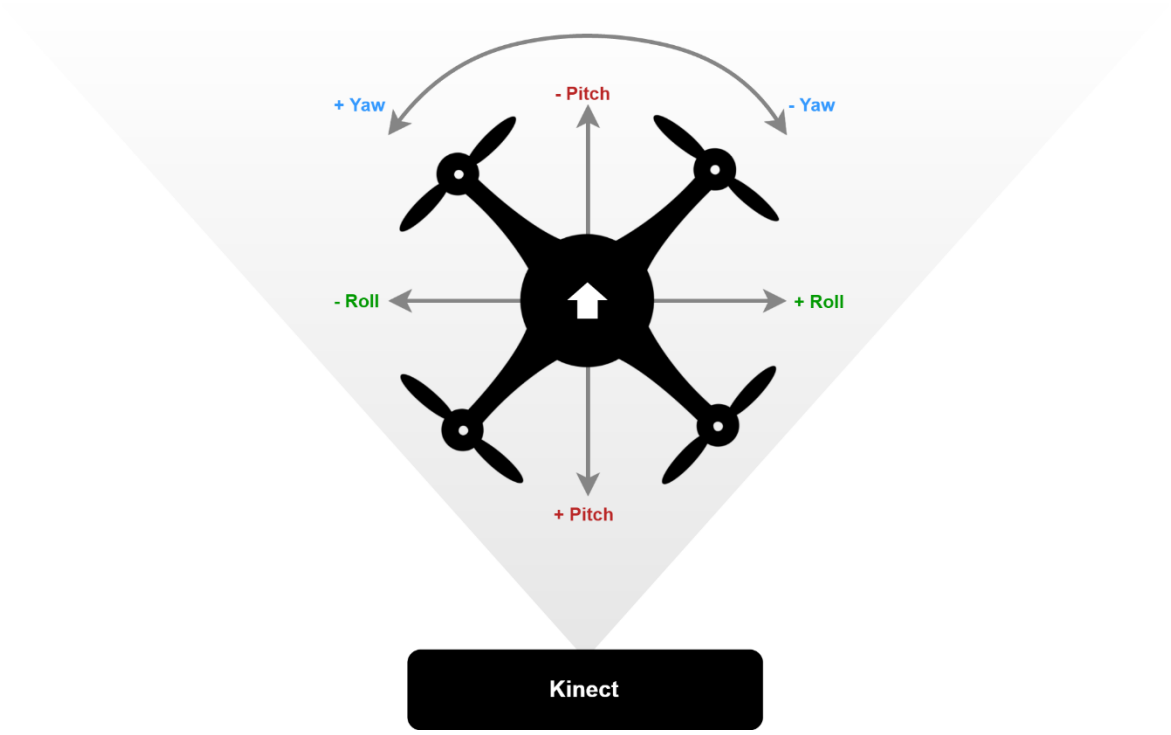


Obrázek 4-5: Princip převodu zkreslené souřadnice na nezkreslenou

Vizualizační a Souřadnice x a y jsou udávány ve zkreslené formě, reálné souřadnice slouží pouze k výpočtu řídících parametrů.

4.2.5 Ovládání a výchozí pozice

Pro vytvoření navigačního algoritmu je potřeba znát, jak na parametry reaguje Crazyflie a určit výchozí pozici ke které se budou řídící parametry přepočítávat. Na Obrázek 4-6: Ovládání a výchozí pozice Crazyflie je vidět výchozí pozice se směrem letu vpřed od sensoru Kinect.

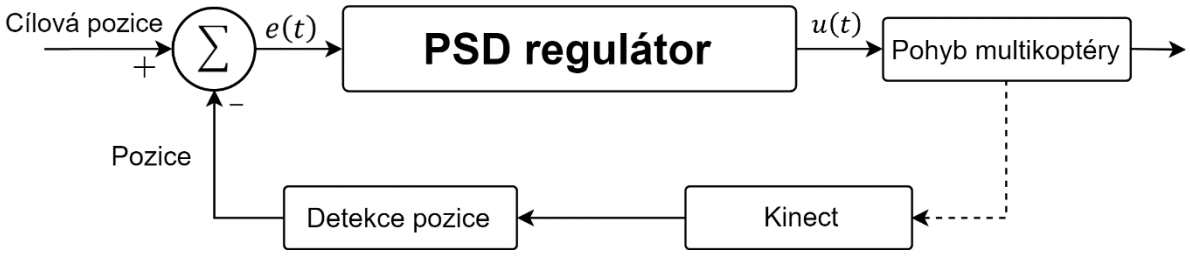


Obrázek 4-6: Ovládání a výchozí pozice Crazyflie

4.2.6 PID kontrolér

Vstupem regulátoru je regulační odchylka $e(t)$, která je výpočtem rozdílu cílové a současné pozice multikoptéry. Zde nastává problém, pokud je umožněno natočení yaw. Z pouhého odečtení cílové a současné pozice se stává mnohem komplikovanější výpočet, který je popsán v kapitole 4.2.7. Uvnitř regulátoru jsou spočteny a následně sečteny výsledné hodnoty jednotlivých regulátorů.

Výstupem jsou akční veličiny $u(t)$ reprezentující jeden z řídících parametrů pitch, roll, yaw a thrust určující pohyb multikoptéry. Pro každý řídící parametr je potřeba vytvořit a nastavit vlastní regulátor, to znamená, že každá jednotlivá souřadnice pozice ovlivňuje jiný řídící parametr. Na obrázku Obrázek 4-7 je naznačen průběh regulace těchto parametrů.



Obrázek 4-7: Model PSD kontroléru pro autonomní let multikoptéry

Další možností, jak zpřesnit pohyb je implementace Kalmanova filtru, který by byl umístěn mezi detekcí pozice a výpočtem regulační odchylky. Kalmanův filtr by eliminoval na základě predikce budoucí pozice časové zpoždění mezi zjištěním pozice a vykonaným pohybem multikoptéry.

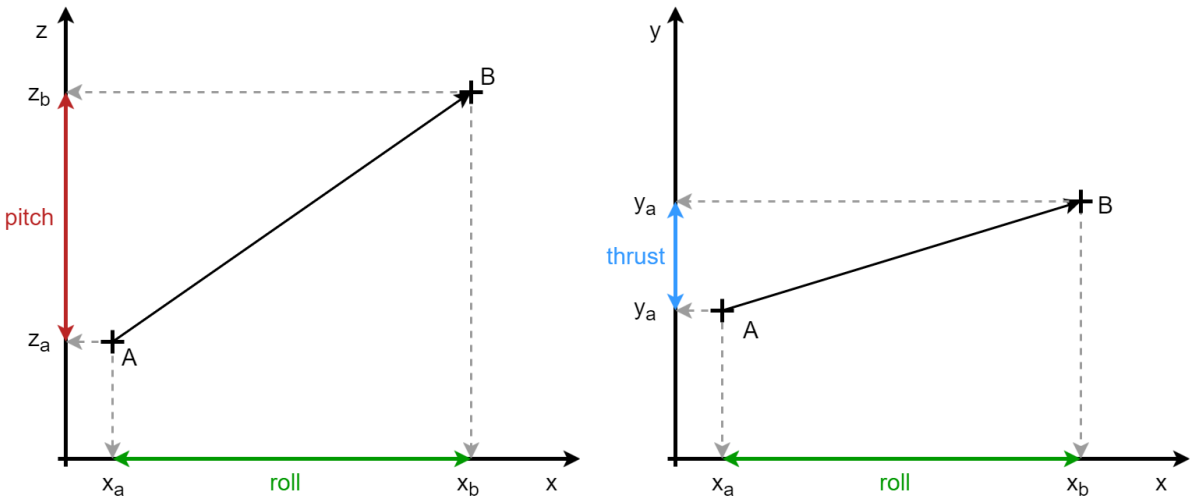
4.2.7 Výpočet řídících parametrů

Pitch, roll, thrust

Pokud se Crazyflie nebude otáčet kolem své osy (yaw, Obrázek 2-3) je možné pitch, roll a thrust zjistit ze složek směrového vektoru směřujícího ze současné pozice A do cílové pozice B v rovině x, z. Směrový vektor se počítá odečtením jednotlivých souřadnic A od B. Roll lze vypočítat pomocí rozdílu x-ových souřadnic, thrust pomocí rozdílu y-nových souřadnic a pitch rozdílem z-tových souřadnic.

$$A(x_a, y_a, z_a)$$
$$B(x_b, y_b, z_b)$$

$$roll = x_b - x_a$$
$$pitch = z_b - z_a$$
$$thrust = y_b - y_a$$



Obrázek 4-8: Výpočet parametrů - pitch, roll, thrust

Pitch, roll, thrust, yaw

Pomocí IMU uvnitř Crazyflie je možné určit momentální natočení Crazyflie yaw_{cf}. Aby bylo možné určovat a regulovat yaw musí být také definováno natočení scény yaw_0. Pomocí těchto hodnot lze určit natočení Crazyflie vůči scéně δ. Možnost určovat natočení ale komplikuje určování řídících hodnot pitch a roll, které se mění v průběhu natáčení Crazyflie.

δ ... *rozdíl úhelu natočení a natočení scény*

yaw_{cf} ... *momentální natočení Crazyflie*

yaw_0 ... *natočení scény*

$$\delta = yaw_{cf} - yaw_0$$

Následující výpočty ukazují, jak lze vypočítat hodnoty pitch a roll při natočení Crazyflie δ.

$\vec{v}$ ... *směrový vektor z bodu A do bodu B v rovině x, z*

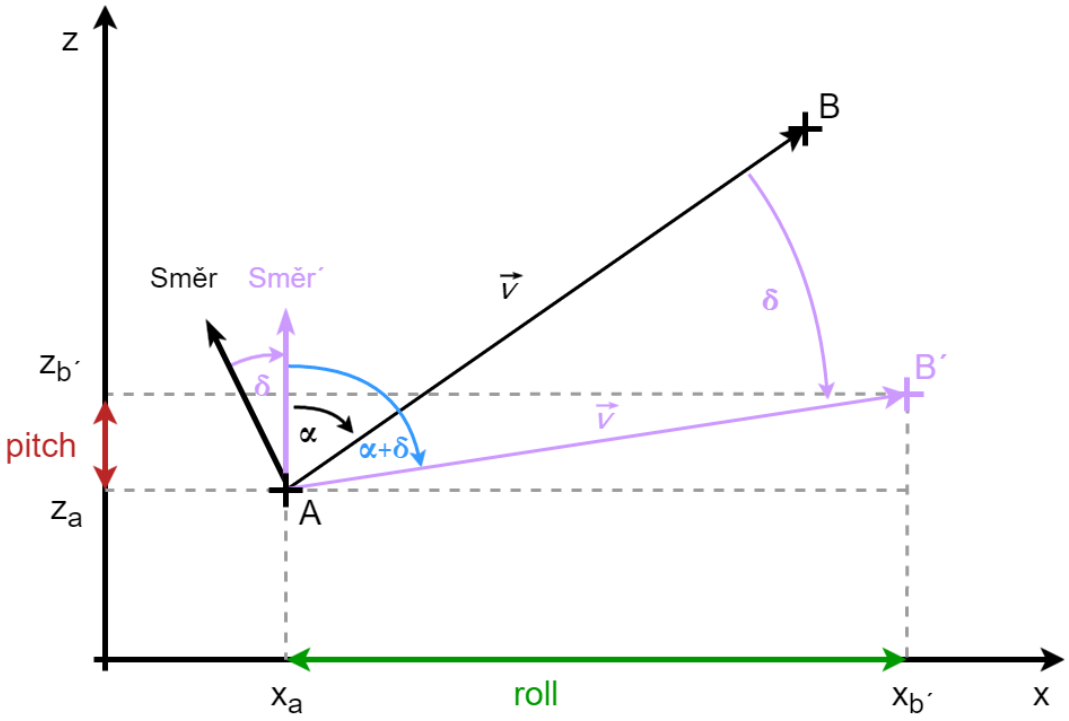
α ... *pomocný úhel*

$$|\vec{v}| = \sqrt{(x_b - x_a)^2 + (z_b - z_a)^2}$$

$$\alpha = \tan^{-1}\left(\frac{z_b - z_a}{x_b - x_a}\right)$$

$$pitch = cos(\alpha + \delta) \cdot |\vec{v}|$$

$$roll = sin(\alpha + \delta) \cdot |\vec{v}|$$



Obrázek 4-9: Výpočet parametrů při natočení yaw

Základním principem je pootočení celé scény tak, aby se Crazyflie nacházelo ve výchozí pozici a je vypočítáno nové umístění bodu B, B´

Výpočet yaw

Výpočet závisí na znalosti současného natočení a cílového natočení Crazyflie. Pokud jsou obě proměnné k dispozici jejich odečtením se zjistí o kolik stupňů je třeba Crazyflie otočit.

yaw_{cf} ... *současné natočení Crazyflie*

yaw_b ... *cílové natočení Crazyflie*

yaw ... *úhlová vzdálenost mezi* yaw_{cf} *a* yaw_b

$$yaw = yaw_b - yaw_{cf}$$

Parametr yaw je do Crazyflie odesílán ve stupních za sekundu. Odhadnout, jak dlouho poletí Crazyflie z bodu A do bodu B lze přibližně z velikosti směrového vektoru z bodu A do bodu B.

$\vec{w}$ ... *směrový vektor z bodu A do bodu B*

$$|\vec{w}| = \sqrt{(x_b - x_a)^2 + (y_b - y_a)^2 + (z_b - z_a)^2}$$

Regulátory

Pouhé vypočítání řídících parametrů nestačí, tyto hodnoty musí být regulované, aby byl pohyb v souřadném systému co neplynulejší a nejpřesnější. Pro tyto účely se nabízí implementace PID regulátoru a Kalmanova filtru

4.2.8 Řízení Crazyflie

Vstupem této části algoritmu jsou souřadnice (x, y, depth), pokud nebyly zjištěny souřadnice Crazyflie jsou automaticky poslány nulové řídící parametry.

Řídící parametry

Tato část programu obdrží souřadnice, na kterých se pohybuje Crazyflie a vypočítá hodnoty řídících parametrů (Tabulka 4-1). Pokud souřadnice nebyli zjištěny, řídící parametry se automaticky nastaví na hodnotu nula a přepošlou se do Crazyflie klienta. Tím je zabráněno, aby byli odesílány nenulové parametry, i když je Crazyflie mimo scénu.

	Rozmezí	**Jednotky**
roll	N/A	stupně
pitch	N/A	stupně
yaw	N/A	stupně za sekundu
thrust	(0-100)zmq, (10001-60000)cflib	(procenta)zmq, (int)cflib

Tabulka 4-1: Řídící parametry

Odeslání parametrů

Parametry je možné odesílat dvěma základními způsoby. Pomocí zmq packetu odeslaného do Crazyflie klienta nebo pomocí Crazyflie knihovny. Knihovna na rozdíl od zmq vyžaduje vytvoření spojení s Crazyflie multikoptérou, o které se v případě zmq stará Crazyflie klient.

5 Realizace

Pro realizaci nadřazeného systému byl zvolen OS Ubuntu verze 16.04 LTS. Veškeré postupy popsané níže jsou prováděny na této verzi Ubuntu, pro jiné verze nebo Linuxové OS nemusí být plně funkční a některé postupy se mohou lišit.

K realizaci byl využit výpočetní systém s následujícími parametry:

- Model: MSI GP60 2OD
- Operační systém: Ubuntu 16.04 LTS 64bit
- CPU: Intel Core i7-4700MQ 2.40 GHz
- GPU: Intel(R) HD Graphics 4600, Nvidia Geforce GT 740M
- RAM (operační paměť): 8192 MB
- Disk: SSD Kingston Now UV400 240 GB

5.1 Instalace ovladačů a knihoven

Repositář Ubuntu obsahuje některé balíčky knihoven, které je poté možné nainstalovat pomocí příkazu `apt-get install`. Tato instalace má ale nevýhodu, že balíček obsažený v Ubuntu repositáři nemusí být nejaktuálnější verze a může se v různých verzích OS lišit a tím přijít o některé možnosti a vylepšení v nejnovějších verzích.

Nejnovější verze knihoven při realizaci:
OpenCV – 3.1.0
libfreenect – 0.5.5
libzmq – 4.2.0

Výhodnější je nainstalovat nejnovější verze knihoven ze zdrojového kódu. Pro tuto možnost je potřeba nainstalovat několik základních balíčků. Před instalací je dobré také zkontrolovat aktualizace. Postupy jsou pro python3 pokud je potřeba z nějakého důvodu použít python2 stačí nahradit v příkazu `python3` nebo `pip3` příkazem `python` a `pip`.

```
Kontrola aktualizací
$ sudo apt-get update
$ sudo apt-get upgrade

Kontrola, zda je balíček nainstalován
$ dpkg -l <nazev_balicku>

Git
$ sudo apt-get install git

Python3
$ sudo apt-get install python3
$ sudo apt-get install python3-pip

Python2
$ sudo apt-get install python
$ sudo apt-get install python-pip

Cmake
$ sudo apt-get install cmake

Balíček build-essential
$ sudo sudo apt-get install build-essential
```

5.1.1 libopencv

Instalace z Ubuntu repositáře

```
$ sudo apt-get install libopencv-dev python-opencv
```

Instalace ze zdrojového kódu

Instalace vychází z [23] a poznatků získaných při samotné instalaci knihovny.

Nejprve je potřeba odstranit starou verzi nainstalovanou z Ubuntu repositáře.

```
$ sudo apt-get autoremove libopencv-dev python-opencv
```

Knihovnu je potřeba sestavit ze zdrojového kódu pomocí Cmake. K tomu je potřeba několik dalších balíčků.

```
Vyžadované balíčky
* GCC 4.4.x a novější
* CMake 2.6 a novější
* GTK+2.x a novější, obsahujícíc hlavičky (libgtk2.0-dev)
* pkg-config
* Python 2.6 a novější, Numpy 1.5 a novější s vývojářskými balíčky
(python-dev, python-numpy)
* ffmpeg nebo libav development packages: libavcodec-dev, libavformat-dev,
libswscale-dev

Optimální balíčky
* libtbb2 libtbb-dev
* libdc1394 2.x
* libjpeg-dev, libpng-dev, libtiff-dev, libjasper-dev, libdc1394-22-dev
```

Balíčky mohou být instalovány pomocí následujících příkazů v terminálu.

```
Vyžadované balíčky
$ sudo apt-get install cmake git libgtk2.0-dev pkg-config libavcodec-dev
libavformat-dev libswscale-dev

Optimální balíčky
$ sudo apt-get install python3-dev python3-numpy libtbb2 libtbb-dev
libjpeg-dev libpng-dev libtiff-dev libjasper-dev libdc1394-22-dev
```

Dále je potřeba získat kopii zdrojového kódu OpenCV knihovny.

```
Získání kopie zdrojového kódu z Git repositáře
$ cd <pracovni slozka>
$ git clone https://github.com/opencv/opencv

Python a doplňky
$ git clone https://github.com/opencv/opencv_contrib
```

V následujících příkazech je říkáno Cmake, že se bude kompilovat release verze a že výsledné soubory budou kopírovat do složky */usr/local*. Ve složce build by se pak měly objevit soubory podle kterých bude probíhat následná instalace.

```
$ cd opencv
$ mkdir build
$ cd build
$ cmake -D CMAKE_BUILD_TYPE=RELEASE \
        -D CMAKE_INSTALL_PREFIX=/usr/local \
        -D INSTALL_PYTHON_EXAMPLES=ON \
        -D OPENCV_EXTRA_MODULES_PATH=~/opencv_contrib/modules \
        -D BUILD_EXAMPLES=ON ..

Samotná instalace opencv knihovny
$ make
$ sudo make install
```

5.1.2 libfreenect

Knihovna freenect a ovladač OpenNI pro Kinect.

Instalace z repositáře Ubuntu

```
$ sudo apt-get install freenect
```

Instalace ze zdrojového kódu

Instalace vychází z [24] a poznatků získaných při samotné instalaci knihovny.

Nejprve je potřeba odstranit starou verzi nainstalovanou z Ubuntu repositáře.

```
$ sudo apt-get autoremove freenect
```

Pro správnou instalaci je potřeba mít nainstalováno několik balíčků.

```
Vyžadované balíčky
* libusb-1.0-0
* libusb-1.0-0-dev
* pkg-config
* freeglut3
* freeglut3-dev

Instalace balíčků
$ sudo apt-get install freeglut3-dev pkg-config build-essential libxmu-dev
libxi-dev libusb-1.0-0-dev
```

Dále je potřeba získat kopii zdrojového kódu libfreenect a nastavit Cmake. Pro možnosti nastavení stačí zadat `cmake -L`. Níže je umožněna navíc instalace audio a podpora OpenCV, které se v základu neinstalují. Pro test je třeba připojit Kinect.

```
Získání kopie zdrojového kódu z Git repositáře
$ cd <pracovni slozka>
$ git clone git://github.com/OpenKinect/libfreenect.git

$ cd libfreenect
$ mkdir build
$ cd build
$ cmake -L .. -D BUILD_CV=ON

$ make
$ sudo make install
$ sudo ldconfig /usr/local/lib64/     #/usr/local/lib/

Test
$ sudo freenect-glview
```

Python wrapper

```
Vyžadované balíčky
* Cython
* python3-dev
* python3-numpy

Instalace balíčků
$ sudo apt-get install cython python3-dev python3-numpy

Global install
$ cd libfreenect/wrappers/python
$ sudo python3 setup.py install

Local install
$ cd libfreenect/wrappers/python
$ python3 setup.py build_ext --inplace
```

Používání Kinectu je prozatím umožněno jen aplikacím, které byly spuštěny pod administrátorskými právy. Řešením je tzv. „udev pravidlo“. Ve složce `/etc/udev/rules.d` se vytvoří soubor s koncovkou `.rules` obsahující pravidla, která umožní aplikacím používání zařízení mimo administrátorský účet.

```
Přidání do skupiny
$ sudo adduser $USER video

Vytvoření pravidla pro kinect
$ cd /etc/udev/rules.d
$ sudo nano 89-kinect.rules
```

Do souboru se vloží následující pravidla:

# ATTR{product}=="Xbox NUI Motor"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02b0", MODE="0666"
# ATTR{product}=="Xbox NUI Audio"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02ad", MODE="0666"
# ATTR{product}=="Xbox NUI Camera"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02ae", MODE="0666"
# Kinect for Windows
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02bf", MODE="0666"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02c2", MODE="0666"
SUBSYSTEM=="usb", ATTR{idVendor}=="045e", ATTR{idProduct}=="02be", MODE="0666"

5.1.3 libzmq

Instalace z Ubuntu repositáře

```
$ sudo apt-get install libzmq5-dev
```

Instalace ze zdrojového kódu

Instalace vychází z [25] a poznatků získaných při samotné instalaci knihovny.

Nejprve je potřeba odstranit starou verzi nainstalovanou z Ubuntu repositáře.

```
$ sudo apt-get autoremove libzmq5-dev
```

Pro správnou instalaci knihovny je potřeba mít nainstalováno několik dalších balíčků.

```
Vyžadované balíčky
* libtool
* autoconf
* build-essential
* pkg-config
* uuid-dev

Instalace balíčků
$ sudo apt-get install libtool autoconf build-essential pkg-config uuid-
dev
```

Při samotné instalaci je u zdrojového kódu přibalen soubor `configure`, který složí ke konfiguraci instalace. Pro nápovědu stačí přidat parametr `--help` k příkazu `/configure`. Bez jakéhokoli parametru se spustí instalace s výchozím nastavením.

S výchozím nastavením se knihovna instaluje do `/usr/local/lib`.

```
Získání kopie zdrojového kódu z Git repositáře
$ cd <pracovni slozka>
$ git clone https://github.com/zeromq/libzmq.git

Instalace
$ cd libzmq
$ ./autogen.sh
$ ./configure --help
$ make
$ make chceck
$ sudo make install

Kontrola instalace
$ sudo ldconfig
```

Pyzmq

```
$ sudo pip3 install cython

Získání kopie z Git repositáře
$ cd <pracovni slozka>
$ git clone git://github.com/zeromq/pyzmq

Instalace [26]
$ $ cd pyzmq
$ python3 setup.py configure --zmq=/usr/local/include
$ sudo python3 setup.py install
```

5.1.4 Crazyradio PA

Pro Crazyradio PA nejsou potřeba na Linux a OSX instalovat žádné další ovladače.

5.1.5 Crazyflie python library (cflib)

Instalace je jednoduchá, složka zdrojového kódu obsahuje python script `setup.py`, který se spustí pomocí `python3-pip` a provede instalaci. [28]

```
Python3-pip
$ sudo apt-get install python3-pip

Získání kopie zdrojového kódu z Git repositáře
$ cd <pracovni slozka>
$ git clone https://github.com/bitcraze/crazyflie-lib-python.git

Instalace v editovatelném modu
$ cd crazyflie-lib-python
$ pip3 install -e .
```

5.1.6 Crazyflie PC client (cfclient)

Úspěšné spuštění Crazyflie clienta je závislé na následujících balíčcích, které je třeba nainstalovat. [29]

```
Vyžadované balíčky
* Python 3.4 a novější
* PyUSB
* libusb
* PyQtGraph
* ZMQ
* PyQt4
* appdirs

Instalace balíčků
$ sudo apt-get install python3 python3-pip python3-pyqt5 python3-zmq
python3-pyqtgraph
$ sudo pip3 install pyusb==1.0.0b2
$ sudo pip3 install pyqt5
$ sudo pip3 install appdirs

Získání kopie zdrojového kódu z Git repositáře
$ cd <pracovni slozka>
$ git clone https://github.com/bitcraze/crazyflie-clients-python.git

Instalace v editovatelném módu
$ cd crazyflie-clients-python
$ pip3 install -e .

Spuštění Crazyflie clienta
$ cd crazyflie-clients-python
$ python3 crazyflie-clients-python/bin/cfclient
```

Po instalaci klienta není možné používat USB zařízení mimo root účet. Následující postup umožní použití Crazyradia a připojení Crazyflie pomocí USB mimo root uživatele.

```
Přidání skupiny plugdev
$ sudo groupadd plugdev
$ sudo usermod -a -G plugdev <uzivatelske_jmeno>

Vytvoření pravidla pro radio
$ cd /etc/udev/rules.d
$ sudo nano 99-crazyradio.rules

Vytvoření pravidla pro crazyflie
$ cd /etc/udev/rules.d
$ sudo nano 99-crazyflie.rules
```

Do otevřených souboru se vloží následující pravidla:

99-crazyradio.rules:
SUBSYSTEM=="usb", ATTRS{idVendor}=="1915", ATTRS{idProduct}=="7777", MODE="0664", GROUP="plugdev"

99-crazyflie.rules:
SUBSYSTEM=="usb", ATTRS{idVendor}=="0483", ATTRS{idProduct}=="5740", MODE="0664", GROUP="plugdev"

Povolení ZMQ vstupu

Ve výchozím nastavení klienta není povoleno používat ZMQ jako vstupní zařízení. Pro povolení je potřeba upravit řádek `enable_zmq_input: false` v konfiguračním souboru clienta `config.json` na `enable_zmq_input: true.` Soubor se nachází ve složce `//home/$USER/.config/cfclient/` a vytvoří se až po prvním spuštění klienta.

```
$ cd //home/$USER/.config/cfclient/.
$ nano config.json
```

5.2 Příprava

5.2.1 Detekce

Aby mohlo být Crazyflie detekováno v obraze je třeba pro jednotlivé způsoby detekce připravit podmínky. Pro vyšší kontrast mezi pozadím a Crazyflie a tím snadnější detekci, je výhodné u obou způsobů detekce použít jako pozadí scény bílé plátno.

Barevný detekční prvek

Pro jednodušší detekci Crazyflie v obraze je na konstrukci připevněn prvek v podobě červeného míčku vyrobeného z polystyrénu (Obrázek 5-1). Polystyrén je lehký, a tak výrazně neovlivňuje letové vlastnosti Crazyflie.

Obrázek 5-1: Detekční prvek na Crazyflie 2.0

Detekce pomocí OpenCV Cascade Classifier

Detekce Crazyflie pomocí vlastního kaskádového klasifikátoru vyžaduje přípravu v podobě pořízení snímků objektu, který má být klasifikován umístěného v jednotvárném pozadí. (positive image) a mnoha fotografií náhodného pozadí (negative image), které neobsahují hledaný objekt. Pro kvalitní klasifikaci objektu je třeba pořídit několik desítek až stovek snímků objektu a pozadí. Jedná se o určitou formu strojového učení Tato technologie se například používá pro detekci obličejů osob.

Vytvoření klasifikátoru vyžaduje mít nainstalované OpenCV. Klasifikátor je vytvořen pomocí funkce `opencv_traincascade`. Zde je postup, jak bylo dosaženo vytvoření klasifikátoru. Postup vychází z [31] [32]

Nástroje pro vytvoření klasifikátoru
```
$ git clone https://github.com/mrnugget/opencv-haar-classifier-training
$ cd opencv-haar-classifier-training
```

Pro vytvoření klasifikátoru jsou potřeba tzv. „samply“ neboli vzorky. Samply se tvoří z negativních a positivních obrazů. Optimální počet je podle [31] 40 positivních a 600 negativních obrazů. Ke klasifikaci Crazyflie bylo použito 120 positivních a 3000 negativních obrazů. Positivní samply se tvoří z obrazů hledaného objektu, které jsou co nejvíce ořezány, ale musí být vidět celý hledaný objekt. Poté se vloží do složky `/positive_images` a v kořenové složce se vytvoří jejich seznam.

Vytvoření seznamu positivních obrazů
```
$ find ./positive_images -iname "*.jpg" > positives.txt
```

Pro tvorbu negativních vzorů je potřeba několik stovek až tisíců obrazů pozadích neobsahující hledaný objekt. Vloží se do složky `/negative_images` a vytvoří se jejich seznam.

Vytvoření seznamu positivních obrazů
```
$ find ./negative_images -iname "*.jpg" > negatives.txt
```

V následujícím příkazu je spuštěn script, který používá funkci `opencv_createsamples`, čímž vytváří náhodné kombinace mezi pozitivními a negativními obrazy do celkového počtu `3000`. Parametry `-w` a `-h` udávají rozlišení vytvořených samplů. Více v [33].

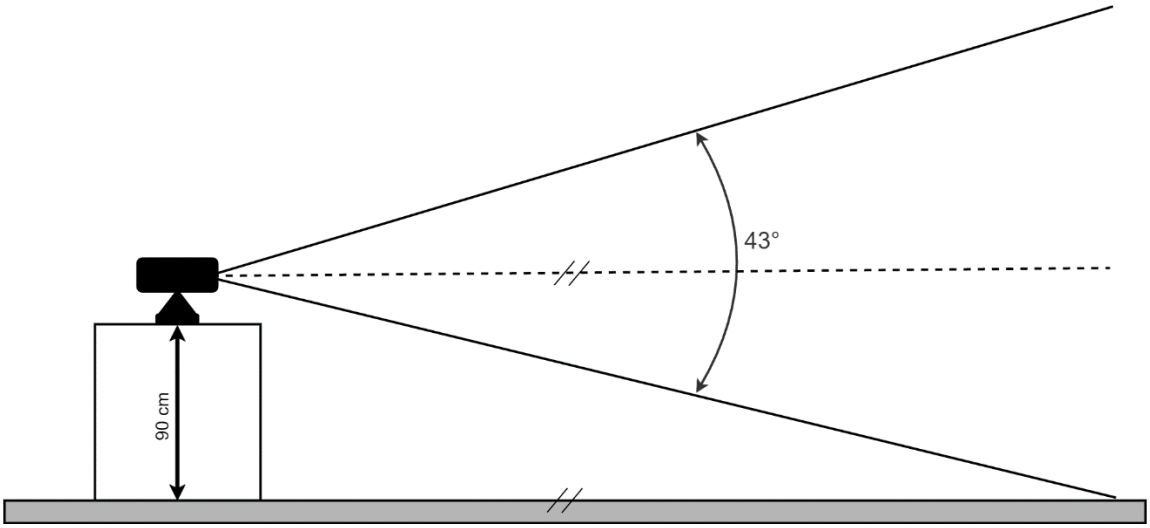
```
Vytvoření seznamu positivních samplů
$ perl bin/createsamples.pl positives.txt negatives.txt samples 3000\
   "opencv_createsamples -bgcolor 0 -bgthresh 0 -maxxangle 1.1\
   -maxyangle 1.1 maxzangle 0.5 -maxidev 40 -w 80 -h 40"

Spojení všech vytvořených vzorů do jednoho
$ python ./tools/mergevec.py -v samples/ -o samples.vec
```

Po vytvoření samplů je možné spustit učící algoritmus `opencv_traincascade`. Učení trvá několik hodin až dní podle počtu použitých samplů a jejich rozlišení. Algoritmus má několik fází, po jejichž dokončení se ukládá výsledek a je možné začít od poslední dokončené fáze, pokud je učení spuštěno se stejnými parametry [33].

```
Vytvoření seznamu positivních obrazů
$ opencv_traincascade -data data -vec samples.vec -bg negatives.txt -
numPos 1800 -numNeg 900 -numStages 10 -w 80 -h 60
```

5.2.2 Umístění senzoru Kinect

Při umístění Kinectu je důležité, aby v prostoru scény senzoru nebyly žádné přebytečné předměty. Ideální umístění je alespoň 90 cm nad zemí s kamerou namířenou rovnoběžně se zemí. Důvodem je, dodržení orientace jednotlivých os souřadného systému, osy *x* a *z* rovnoběžné se zemí a osa *y* kolmá k zemi.



Obrázek 5-2: Umístění sensoru Kinect

5.3 Algoritmizace

Programová část je rozdělena do dvou tříd (Obrázek 4-3). První třída vytváří rozhraní mezi senzorem Kinect a knihovnou OpenCV a druhá realizuje samotné navigační algoritmy a odesílá řídící parametry. V této kapitole jsou ukázány pouze některé příklady. Celý zdrojový kód programu s komentářem je umístěn na přiloženém CD, viz Přílohy.

5.3.1 Detekce

Barevný detekční prvek

Tento kód popisuje detekci barevného prvku v podobě červeného míčku umístěného k rámu Crazyflie (5.2.1Barevný detekční prvek).

Filtrace obrazu:

```
# rozmezi hledanyh barev
lower_red = np.array([120, 120, 30])
upper_red = np.array([220, 255, 255])

# prevod do HSV reprezentace barev
hsv = cv2.cvtColor(img, cv2.COLOR_BGR2HSV)
# vyhlazovani a filtrace
blurhsv = cv2.GaussianBlur(hsv, (5, 5), 0)
# filtrace podle rozmezi barev, threashold
img_th = cv2.inRange(blurhsv, lower_red, upper_red)
erode = cv2.erode(img_th, None, iterations=1)
dilated = cv2.dilate(erode, None, iterations=2)
```

Hledaní barevného detekčního prvku a jeho pozice x, y:

```
# hledani obrysu v threshold obraze
image, cnts, hierarchy = cv2.findContours(dilated.copy(),
	cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)
center_point = None

# pokud byl nalezen obrys
if len(cnts) > 0:
	# hledani nejvetsiho obrysu a hledani jeho
	# nejmensi obklopujici kruznice a stredu
	c = max(cnts, key=cv2.contourArea)
	(x, y), radius = cv2.minEnclosingCircle(c)
	center = (int(x), int(y))
	radius = int(radius)

	# filtrace malych objektu
	if radius > 8:
		center_point = center
```

OpenCV Cascade Classifier

Tento kód popisuje použití vytvořeného kaskádového klasifikátoru vytvořeného v kapilole 5.2.1.

Hledání Crazyflie v obraze pomocí vytvořeného klasifikátoru.

```
# pouziti vlastniho klasifikatoru crazyflie
crazyflie_cascade = cv2.CascadeClassifier('crazyflie_classifier.xml')

# prevedeni na cerno-bily snimek
gray = cv2.cvtColor(img, cv2.COLOR_BGR2GRAY)
# hledani crazyflie v obraze, objekty jsou vraceny jako seznam obdelniku
# (x, y, w, h) stred, vyska, sirka
find_crazyflie = crazyflie_cascade.detectMultiScale(gray)
```

Depth (z)

Předpokladem pro získání souřadnice *z* je detekování Crazyflie na pozicích *x, y*. Je důležité zmínit, že při odkazování se na místo numpy matice je potřeba prohodit souřadnice, jelikož je nejdříve odkazováno na řádek a až poté na místo v řádku.

Výpočet vzdálenosti na souřadnicích x, y.

```
# ziskani hodnoty v depth matici na pozici x, y
rawdis = raw_depth[y, x]

# vypocet vzdalenosti v cm
distance = 0.1236 * math.tan(rawdis / 2842.5 + 1.1863) * 100
```

5.3.2 Deformace souřadného systému

Tento kód je realizací navržených výpočtů v kapitole 0. Kód přepočítává souřadnice deformovaného souřadného systému na reálné.

```
# vzdalenosti kde maji x, y souradnice stejne jednotky (cm) jako depth
Depth0x =  589
Depth0y =  481

# x, y jsou souradnice detekovaneho objektu v obraze
# souradnice y je v obraze brana od shora
y = 480-y

# vypocet rozdilu na osah
r_x = ((x - 320) / Depth0x) * (depth-Depth0x )
r_y = ((y - 240) / Depth0y) * (depth-Depth0y )

# realne souradnice rx, ry
yr = y + r_y-240
xr = x + r_x-320
```

5.3.3 Výpočet řídících parametrů při natočení yaw

Výpočet je prováděn pomocí reálných souřadnic. Výsledkem jsou řídící parametry pitch a roll při jakémkoli natočení yaw Crazyflie. Kód je realizací navržených výpočtů v kapitole 4.2.6.

```
# alfa je uhel mezi osou x a vektorem v
alfa = math.atan((depth_b-depth_a)/(xb-xa))

# vzdalenost ciloveho a soucasneho bodu v rovine (x, depth(z))
v = math.sqrt(math.pow((xb - xa), 2) + math.pow((depth_b - depth_a), 2))

# posun oproti vychozi pozici, delta je natoceni Crazyflie oproti scene
# beta urcuje uhel mezi smerem Crazyflie a smerem vektoru v
if xb>xa:
    beta = -(alfa-math.radians(90))-math.radians(delta)
else:
    beta = -(alfa-math.radians(270))-math.radians(delta)

i=0
# je mozne pracovat pouze s uhlem 0-
90, i posleze znaci v jakem kvadrantu se nachazi cilovy bod
while beta > math.radians(90):
    beta = beta - math.radians(90)
    i=i+1

# velikost pitch a roll
roll = math.sin(beta) * v
pitch = math.cos(beta) * v

# urceni ridicich parametru podle toho v jakem
# kvadrantu se nachazi cilovy bod
if i==0:
    pitch = -pitch
    roll = roll
if i==1:
    temp = pitch
    pitch = roll
    roll = temp
if i==2:
     roll = -roll
if i==3:
    temp = pitch
    pitch = -roll
    roll = -temp
```

5.3.4 Implementace PID regulátoru

Vstupem pro PID regulátor jsou vzdálenosti (4.2.6) mezi současným a cílovým bodem rozložené na jednotlivých osách souřadného systému. Model diskrétního regulátoru byl převzat z projektů Bitcraze [3] a [4] (viz Přílohy).

Inicializace regulátoru:

```
# P - proporcialni slozka
# I - integracni slozka
# D - derivacni slozka
# set_point - vstup regulatoru, pri inicializaci je nulovy

roll = PID_RP(P=0, I=0, D=0, set_point=0.0)
pitch = PID_RP(P=0, I=0, D=0, set_point=0.0)
thrust = PID(P=0, I=0, D=0, set_point=0.0)
```

Použití regulátoru:

```
# r, p, t – vstupy regulatoru (odchylky roll, pitch, thrust)
roll = roll_pid.update(r)
pitch = pitch_pid.update(p)
thrust = thrust_pid.update(t)
```

5.4 Řešené problémy a testování

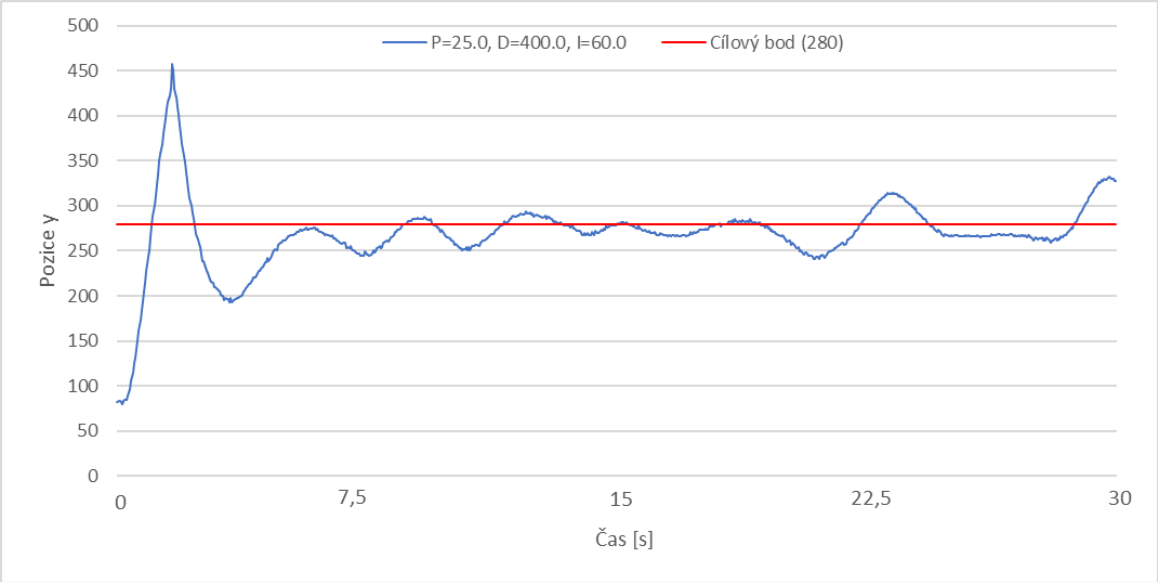
5.4.1 Složky PSD regulátoru

Program obsahuje tři (pitch, roll, thrust), popřípadě čtyři (yaw) diskrétní regulátory. Thrust nejvíce ovlivňuje přesnost pohybu multikoptéry. K dispozici byly nastavené konstanty z předchozích projektů Bitcraze [3] a [4], které byly použity pro výchozí nastavení složek regulátorů pitch, roll a yaw. Výchozí hodnoty z projektů pro nastavení konstant regulátoru thrust nebyly vhodné, jelikož po jejich aplikování byl systém nestabilní.

Konstanta	Rychlost odezvy	Stabilita odezvy
Proporcionální	Zvyšuje	Snižuje
Integrační	Snižuje	Zvyšuje
Derivační -	Zvyšuje	Snižuje

Tabulka 5-1: Vliv konstant na rychlost a stabilitu odezvy

Konstanty thrust regulátoru byly manuálně upraveny na hodnoty P=25, I=60, D=400. Prvním krokem bylo zjištění přibližné hodnoty PWM[10], reprezentující thrust při kterém Crazyflie ani neklesá ani nestoupá, ke které byla následně přičtena případně odečtena hodnota regulátoru thrust.



Graf 5-1: Průběh regulované výšky (y), regulátor thrust

Oproti pozemním robotům na multikoptéru působí mnohem více vedlejších jevů, které ovlivňují letové vlastnosti. Například, gravitace, teplota prostředí nebo jednotlivé výstupní hodnoty ostatních regulátorů. Pokud je například multikoptéra nakloněna je část energie přenesena do směru náklonu.

Existuje několik postupů, jak nastavovat složky PID regulátoru. Jedním z nejuniverzálnějších a nejpoužívanějších způsobů nastavení konstant PID regulátoru je Ziegler-Nichols. Ovšem pro co nejoptimálnější nastavení bývá potřeba regulátor doladit manuálně. Ziegler-Nichols často používá pro určení výchozího nastavení, které se případně ladí manuálně.

[10] PWM (Pulse Width Modulation) – pulzně šířková modulace, diskrétní modulace pro přenos analogového signálu

5.4.2 Převod souřadnic barevného snímku na souřadnice depth sensoru

Problémem je, že pixely na stejné pozici (souřadnice) nesnímají v RGB a deph snímcích stejné místo, z důvodu posunu kamer a rozdílných zorných polí kamer. Není tedy možné se jednoduše odkazovat na stejné místo z RGB na depth snímek a opačně. Následující tabulka a vzorce ukazují, jak bylo docíleno zpřesnění odkazování na jednotlivé pozice v depth snímku. Měření bylo docíleno vytvořením marek v depth snímku vždy 10 pixelů od krajů snímku. Na tyto marky pak byl umístěn červený míček, který byl detekován v RGB snímku a následně byla vypsána jeho pozice.

RGB souřadnice (x, y)	**Depth souřadnice (x, y, z)**	**Posun (x, y)**
15, 32	10, 10, 100	-5, -22
590, 460	630, 470, 100	40, 10
590, 32	630, 10, 100	40, -22
15, 460	10, 470, 100	-5, 10

Tabulka 5-2: Měření posunu

Z tabulky Tabulka 5-2 vyplívá, že posun na ose *x* je v rozmezí (-5, 40) a na ose y (-22, 10). Z rozmezí posunu lze zjistit posun na pixel vydělením velikosti intervalů posunu velikostí intervalů pixelů.

$$x_p \dots posun\ na\ 1\ bod\ osy\ x$$

$$y_p \dots posun\ na\ 1\ bod\ osy\ y$$

$$x_p = \frac{posunx_{max} - posunx_{min}}{pixelx_{max} - pixelx_{min}}$$

$$y_p = \frac{posuny_{max} - posuny_{min}}{pixely_{max} - pixely_{min}}$$

Poté lze ze znalosti x, y souřadnic v RGB snímku zjistit souřadnice v depth snímku. Jako pojistka je v okolí 10 pixelů vybrána nejkratší vzdálenost.

$$x_d, y_d \dots souřadnice\ v\ depth\ matici$$

$$x, y \dots souřadnice\ RGB\ matice$$

$$x_d = x - posunx_{\text{xmin}} + \left((x - posunx_{\text{min}}) \cdot x_p\right)$$
$$y_d = y - posuny_{\text{min}} + \left((y - posuny_{\text{min}}) \cdot y_p\right)$$

Je důležité si uvědomit, že díky posunu depth senzoru oproti RGB kameře se zmenšuje souřadnicový systém. Jinak řečeno, některé souřadnice RGB snímku nemají po přepočtu reálný protiklad v depth snímku a opačně. V tomto případě lze spolehlivě detekovat pozici v depth snímku přibližně v souřadnicích x(15-590) a y(32-460).

Přepočet souřadnic:

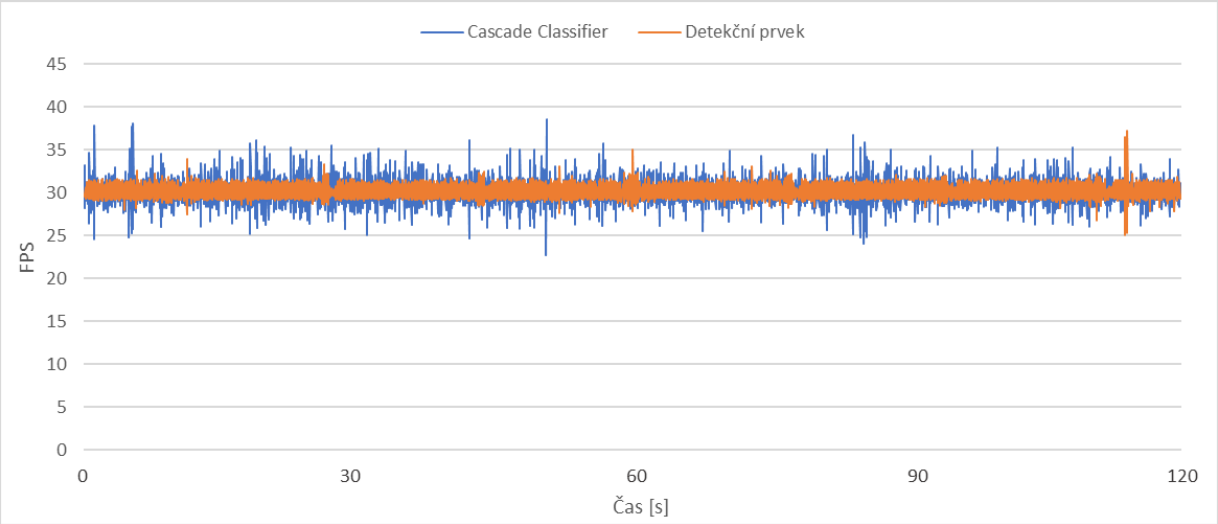
```
# posun na pixel
xp = 1/13
yp = 16/225

# prepocet souradnic
xd = round(x - 10 + (x*xp))
yd = round(y - 27 + (y*yp))

# coordinace nesmeji byt mimo rozliseni obrazu
if xd > 630:
    xd = 630
if xd < 0:
    xd = 0
if yd > 470:
    yd = 470
if yd < 0:
    yd = 0
```

Nejkratší vzdálenost v okruhu 10 pixelů:

```
# nejmensi vzdalenost v okruhu 10 pixelu
for i in range(0,9):
    for j in range(0,9):
        # pokud je hodnota na pozici mensi nez doposud
        # nalezena je ji nahrazena
        if depth_img[yd + i, xd + j] < depth:
            depth = depth_img[yd+i  ,xd+j]
```

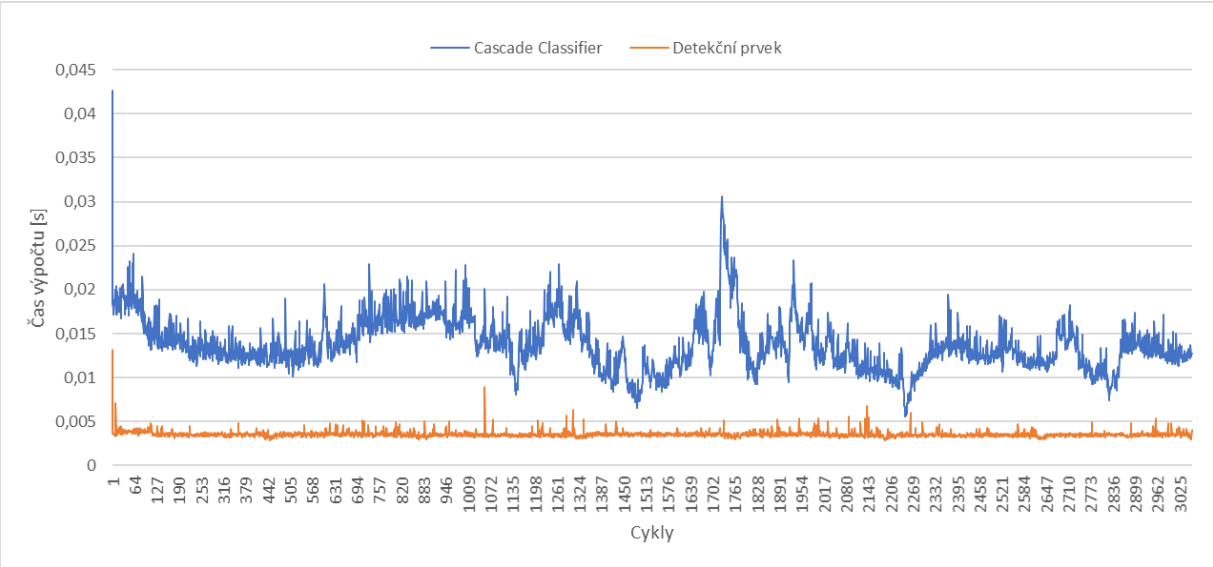
5.4.3 Dopad analýzy obrazu na výkon

Měření je prováděno na realizačním Realizace. Měření je zaměřeno na porovnání dopadu rozdílné analýzy obrazu pro detekci Crazyflie na výkon sytému. Jedná se o způsob detekce pomocí detekčního prvku a kaskádového klasifikátoru (Cascade Classifier). V grafu Graf 5-2 jsou vidět naměřené fps v průběhu dvou minut. Měření není naprosto přesné, ale i tak z grafu vyplívá, že způsob detekce objektu nemá při výkonu testovacího PC vliv na fps.



Graf 5-2: Porovnání fps při odlišném způsobu detekce

Dále byla změřena doba výpočtu od získání obrazu až po získání pozice *x*, *y* ve snímku (Graf 5-3). Z grafu je jasné, že detekce objektu pomocí vytvořeného klasifikátoru je přeci jen výpočetně náročnější než pomocí barevného detekčního prvku. Lze tedy usoudit, že u méně výkonných výpočetních systémů může mít volba způsobu detekce vliv na rychlost zpracování snímků, fps. Pomocí průměru všech naměřených hodnot je možné zjistit, že na realizačním PC výpočet pomocí klasifikátoru trvá průměrně 35,07 % času jednoho cyklu a pomocí detekčního prvku 9,08 %.



Graf 5-3: Čas výpočtu odlišných způsobů detekce

5.4.4 Úspěšnost detekce pomocí detekčního prvku v různých pozadí

Detekce barevného spektra v obraze je problematická, jelikož záleží na osvětlení a vnímání barev samotné kamery. Nastavení hledaného barevného spektra pro tentýž barevný objekt tedy vyžaduje upravení podle prostředí (pozadí) ve kterém je objekt hledán.

Testování bylo prováděno s použitím těchto rozmezí hledaných barev v obraze.

```
# sirsi rozmezi barevného spektra
lower_red = np.array([100, 120, 120])
upper_red = np.array([240, 255, 255])

# uzsi rozmezi barevneho spektra
lower_red = np.array([150, 145, 145])
upper_red = np.array([180, 255, 255])
```

Úspěšnost detekce byla měřena za denního světla z 3000 snímků, Kde byl v zorném poli náhodně přemisťován barevný detekční prvek, viz 5.2.1.

	Širší rozmezí	Detekce špatného objektu	**Užší rozmezí**	Detekce špatného objektu
Bílé pozadí	100 % (3000)	0	100 % (3000)	0
Heterogenní pozadí	96,22 % (2886)	3,78 % (114)	99,6 % (2988)	0

Tabulka 5-3: Úspěšnost detekce

Při použití bílého pozadí je možné použít širší barevné spektrum a tím zvýšit úspěšnost detekce. Při detekci v heterogenním pozadí je potřeba použít užší barevné spektrum, jinak budou detekovány i jiné barevné odstíny než na hledaném objektu. Dobrým způsobem, jak zjistit přibližný rozsah je si na určitém pixelu vypisovat barvy a přikládat k němu objekt, který má být detekován. Následně hodnoty zkoušením doladit. Testovaná pozadí jsou umístěna na přiloženém CD, viz Přílohy.

Cascade Classifier

Detekce Crazyflie pomocí vytvořeného klasifikátoru na bílém pozadí fungovala relativně v pořádku. Ovšem detekce v heterogenních pozadích provázela chyba v podobě 2 až 4 detekcí v jediném snímku. Načež bylo rozhodnuto, že k dokončení práce bude využito detekce s pomocí barevného detekčního prvku s možností pozdějšího nahrazení kvalitnějším kaskádovým klasifikátorem.

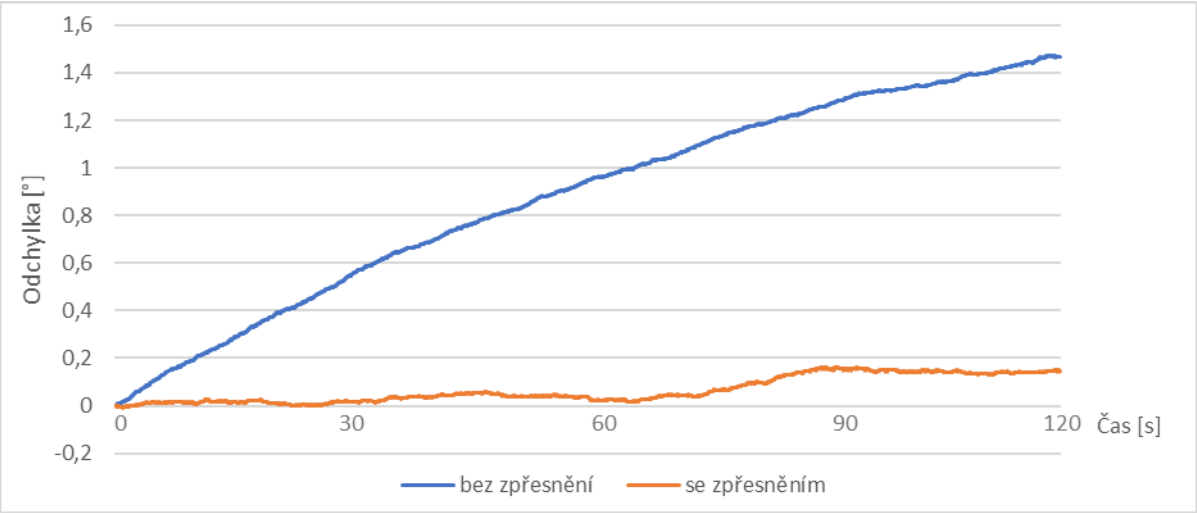
5.4.5 Nepřesné údaje yaw z Crazyflie

Údaje o poloze získané z Crazyflie se lehce mění i setrvávání v jedné poloze, což může při letu působit nepřesnosti v navigování. Toto je jednoduchý způsob, jak bylo docíleno zmenšení odchylky yaw získané z Crazyflie. Princip je takový, že při inicializaci systému je několik sekund po sobě získán z Crazyflie údaj o natočení (yaw) a tím je možné jednoduchým výpočtem určit velmi přibližnou odchylku za jednotku času.

Y ... odchylka yaw za sekundu (yaw/s)
T_t *... naměřený čas (s)*
yaw_t *... hodnoty yaw Crazyflie v čase (t)*

$$Y = \frac{yaw_{t-1} - yaw_t}{T_t - T_{t-1}}$$

Odchylku je pak možné přičíst k výchozímu natočení scény, nebo odečíst z právě získaného natočení Crazyflie. Následující graf (Graf 5-4) ukazuje, jak může být pomocí tohoto výpočtu lehce zpřesněn údaj o natočení (yaw). Při krátkém letu odchylka yaw nepůsobí problémy.



Graf 5-4: Odchylka yaw

5.5 Spuštění a ovládání programů

Spuštění programů uložených na přiloženém CD (viz Přílohy) je možné pomocí příkazů v terminálu. Na CD jsou ve složce `/software` uloženy dva příklady. Pro detekci je u obou příkladů použit jako výchozí barevný detekční prvek (5.3.1). Do zdrojového kódu je implementován i kaskádový klasifikátor.

První příklad (zmq)

Program je realizován pomocí zmq a Crazyflie klienta. Neumožňuje regulovat yaw a z pozici. Před spuštěním programu je nutné povolit zmq vstup u CF klienta a připojit se ke Crazyflie prostřednictvím klienta. Také je nutné umístit Crazyflie do výchozí pozice (4.2.5).

Druhý příklad (API)

Tento příklad je realizován pomocí Crazyflie knihovny (API), která se stará i o připojování ke Crazyflie. Tento příklad umožňuje regulovat pozici yaw a depth. Před spuštěním je nutné umístit Crazyflie do výchozí pozice (4.2.5).

```
Spuštění prvního příkladu (zmq, CF klient)
$ python3 ~/software/CrazyflieAutopilot-zmq.py

Spuštění druhého příkladu (CF API)
$ python3 ~/software/CrazyflieAutopilot-API.py
```

Programy jsou ovládány pomocí kláves a myši, následující tabulka ukazuje veškeré operace, které je v programech možné provádět.

Povel	**Rozsah**	**Ovládání**	**Program**
ukončení	-	klávesa „**q**“	zmq, API
depth +	(70, 200) mm	klávesa „**w**“	API
depth -	(70, 200) mm	klávesa „**s**“	API
yaw +	(-180, 180) stupňů	klávesa „**d**“	API
yaw -	(-180, 180) stupňů	klávesa „**a**“	API
x	(15, 590) pixelů	myš – levé tlačítko	zmq, API
y	(32, 460) pixelů	myš – levé tlačítko	zmq, API

Tabulka 5-4: Ovládání programu

Závěr

Tato práce se zabývá problematikou autonomního letu multikoptér. Cílem bylo sepsání metodiky a vytvoření postupů, kterými bylo docíleno demonstrativního autonomního letu multikoptéry Crazyflie 2.0 s použitím senzoru Kinect.

V teoretické části byly formou analýzy dostupných řešení spolu se stavbou subsystémů autonomního robota vysvětleny základy problematiky a rovněž byly popsány základní principy pohybu a stavby multikoptér. Na základě této analýzy byl vytvořen obecný model, který vedl k rozdělení systému do dvou samostatných částí. Řídící systém multikoptéry se stal podřízeným navigačnímu systému později realizovaného pomocí programu. Po vytvoření obecného modelu byly k jednotlivým částem modelu přiřazeny využívané technologie.

Poznatky z analýzy a navržený model byly výchozím bodem pro vytvoření návrhu. Značnou výhodou bylo, že platforma Crazyflie má připravené rozhraní pro komunikaci pomocí Crazyradia prostřednictvím připraveného API a klienta. Proto se praktická část zaměřovala především na návrh globálního souřadného systému pomocí senzoru Kinect a výpočtu řídících parametrů. Nevýhodou při použití senzoru Kinect lze považovat maximální limit 30 fps RGB kamery, to může být nedostatečné pro detekci rychle se pohybujícího objektu. Tento problém by mohl být řešen použitím jiné kamery. Nastává ale otázka, jak by byly zvládány výpočty analýzy obrazu při více fps. Pro detekci Crazyflie v obraze byly zkoušeny dva metody. Jako první byla zkoušena metoda použitá již v předchozích projektech Bitcraze s barevným detekčním prvkem. Druhá metoda byla pokusem o vytvoření kaskádového klasifikátoru. Detekce pomocí barev je rychlá a přesná, ale problematická při změnách světelných podmínek. Oproti tomu při použití klasifikátoru není kladen důraz na barevnost a světelné podmínky, ovšem je náročné vytvořit opravdu kvalitní klasifikátor. Vytvořený klasifikátor nebyl příliš spolehlivý na heterogenním pozadí. Vytvoření nového klasifikátoru by zabralo mnoho času a výsledek by nemusel být lepší. Proto bylo ve zbytku práce užívána detekce pomocí barevného detekčního prvku. Po získání pozice pomocí analýzy obrazu byl řešen problém s posunem mezi RGB a depth snímky. Problém byl vyřešen zjištěním posunu a nejmenší vzdálenosti v okruhu 10 pixelů. Následně bylo matematicky řešeno zkreslení souřadnic a jejich přepočítání do reálných souřadnic. Díky možnému přepočtu na souřadnice s reálnými rozměry bylo možné vytvořit návrh pro regulaci polohy s natočením yaw.

Výsledkem práce bylo vytvoření dvou příkladů systémů, na nichž byly jako poslední krok nastavovány jednotlivé složky regulátorů. První systém byl realizován s pomocí zmq a Crazyflie klienta, kde není možné regulovat yaw ani pozici depth. Druhý experimentální systém byl vytvořen na základě návrhů výpočtů reálných souřadnic s možností regulace yaw a depth pozice. Vzhledem k náročnosti seřizování regulátorů nebyla tato problematika rozebírána podrobně a byla ponechána pro případné pokračování v nadcházejících výzkumech. Dalším vylepšení pro tyto systémy by mohla být implementace Kalmanuva filtru, který byl v práci pouze zmíněn, nebo využití akcelerometru umístěného v Kinectu k vyrovnávání souřadného systému. Vzhledem k obsáhlosti tématu jsou v práci popsány převážně základy a vylepšení jednotlivých částí systému je tématem pro další výzkum. Za hlavní přínos této práce lze tedy považovat prezentování základních principů, které mohou posloužit jako výchozí bod pro další výzkum v této oblasti nebo návazností na něj. Výsledkem práce je provedení postupem při tvorbě jednotlivých částí systému pro autonomní let multikoptéry Crazyflie 2.0 s použitím senzoru Kinect. Primární cíl práce lze tedy považovat za splněný.

Seznam použité literatury

[1] FAHLSTROM, Paul Gerin a Thomas James GLEASON. *Introduction to UAV systems*. 4th ed. Chichester: John Wiley, 2012. Aerospace series. ISBN 978-1-119-97866-4.

[2] NOVÁK, Petr. *Mobilní roboty: pohony, senzory, řízení.* Praha: BEN - technická literatura, 2005. ISBN 80-730-0141-1.

[3] ELIASSON, Marcus. *Autopilot using Kinect and a PC* [online]. 2013 [cit. 2017-03-06]. Dostupné z: https://www.bitcraze.io/2013/08/autopilot-using-kinect-and-a-pc/

[4] ANTONSSON, Tobias. *Autonomous flight using Kinect2 for position control* [online]. 2015 [cit. 2017-03-17]. Dostupné z: https://www.bitcraze.io/2015/05/autonomous-flight-using-kinect2-for-position-control/

[5] Using the Accelerometer. Husstech [online]. 2014 [cit. 2017-03-23]. Dostupné z: http://husstechlabs.com/projects/atb1/using-the-accelerometer/

[6] KVĚTENSKÝ, Tomáš. Systém měření náklonu pomocí akcelerometrů. Praha, 2008. Diplomová. ČVUT. Vedoucí práce Ing. Jan Roháč Ph.D.

[7] Arduino Nano a akcelerometr. Elektronika, robotika, modely, Lego [online]. 2014 [cit. 2017-03-20]. Dostupné z: http://www.josefnav.cz/Arduino_akcelerometr.html

[8] Integrované MEMS GYROSKOPY. VOJÁČEK, Antonín. Automatizace [online]. 2009, 11.10.2009 [cit. 2017-03-20]. Dostupné z: http://automatizace.hw.cz/integrovane-mems-gyroskopy

[9] Co je a k čemu slouží magnetometr? [online]. 2008 [cit. 2017-03-29]. Dostupné z: http://www.enviweb.cz/clanek/geologie/71131/co-je-a-k-cemu-slouzi-magnetometr

[10] BLAHA, Petr a Petr VAVŘÍN. Řízení a regulace I [online]. [cit. 2017-03-06]. VUT Brno. Dostupné z: http://www.uamt.feec.vutbr.cz/~richter/vyuka/0809_BRR1/texty/brr1.pdf.

[11] VOJÁČEK, Antonín. *Co je to Kalmanova filtrace ?* [online]. 2007 [cit. 2017-04-05]. Dostupné z: http://automatizace.hw.cz/clanek/2007042901

[12] ESME, Bilgin. Kalman Filter For Dummies [online]. 2009 [cit. 2017-04-05]. Dostupné z: http://bilgin.esme.org/BitsAndBytes/KalmanFilterforDummies

[13] Computer vision. In: *Wikipedia: the free encyclopedia* [online]. San Francisco (CA): Wikimedia Foundation, 1017 [cit. 2017-03-20]. Dostupné z: https://en.wikipedia.org/wiki/Computer_vision

[14] Ing. Karel Horák, Ph.D., Ing. Ilona Kalová, Ph.D., Ing. Petr Petyovský, Ing. Miloslav Richter, Ph.D., *Počítačové vidění* [online]. VUT Brno, 2008 [cit. 2017-03-06]. Dostupné z: http://www.uamtold.feec.vutbr.cz/vision/TEACHING/MPOV/Pocitacove_videni_S.pdf.

[15] SZYMCZYK, Matthew. How *Does The Kinect 2 Compare To The Kinect 1?* [online]. In: . 2014 [cit. 2017-03-20]. Dostupné z: http://zugara.com/how-does-the-kinect-2-compare-to-the-kinect-1

[16] Kinect Sensor. *Microsoft Developer Network* [online]. 2012 [cit. 2017-03-20]. Dostupné z: https://msdn.microsoft.com/en-us/library/hh855355.aspx

[17] Kinect for Windows Sensor Components and Specifications: Kinect Sensor. *Microsoft Developer Network* [online]. 2012 [cit. 2017-03-20]. Dostupné z: https://msdn.microsoft.com/en-us/library/jj131033.aspx

[18] Accelerometer: Kinect Sensor. *Microsoft Developer Network* [online]. 2012 [cit. 2017-03-20]. Dostupné z: https://msdn.microsoft.com/en-us/library/jj663790.aspx

[19] How the Kinect Depth Sensor Works in 2 Minutes. In: *YouTube* [online]. 2013 [cit. 2017-03-30]. Dostupné z: https://www.youtube.com/watch?v=uq9SEJxZiUg

[20] Aktivní triangulace. In: *Multimediální interaktivní didaktický systém* [online]. 2013 [cit. 2017-03-30]. Dostupné z: http://midas.uamt.feec.vutbr.cz/POV/Exercise02/02_Aktivni_triangulace.pdf

[21] Crazyflie 2.0. In: *Bitcraze Wiki* [online]. [cit. 2017-03-06]. Dostupné z: https://wiki.bitcraze.io/projects:crazyflie2:index

[22] Crazyradio PA. In: *Bitcraze Wiki* [online]. [cit. 2017-03-06]. Dostupné z: https://wiki.bitcraze.io/projects:crazyradiopa:index

[23] Installation in Linux: *OpenCV* [online]. 2015 [cit. 2017-03-31]. Dostupné z: http://docs.opencv.org/3.1.0/d7/d9f/tutorial_linux_install.html

[24] Ubuntu Manual Install: *OpenKinect* [online]. 2016 [cit. 2017-03-31]. Dostupné z: https://openkinect.org/wiki/Getting_Started#Ubuntu_Manual_Install

[25] libzmq Source Git Repository: *ZeroMQ* [online]. 2010 [cit. 2017-03-31]. Dostupné z: http://zeromq.org/docs:source-git

[26] Building PyZMQ from source [online]. 2013 [cit. 2017-03-31]. Dostupné z: https://github.com/zeromq/pyzmq/wiki/Building-and-Installing-PyZMQ

[27] *Imaging Information* [online]. 2013 [cit. 2017-04-08]. Dostupné z: https://openkinect.org/wiki/Imaging_Information

[28] README.md: *Crazyflie python library* [online]. 2016 [cit. 2017-03-31]. Dostupné z: https://github.com/bitcraze/crazyflie-lib-python/blob/master/README.md

[29] README.md: *Crazyflie PC client* [online]. 2016 [cit. 2017-03-31]. Dostupné z: https://github.com/bitcraze/crazyflie-clients-python/blob/develop/README.md

[30] *PyKalma* [online]. [cit. 2017-04-09]. Dostupné z: http://pykalman.github.io/

[31] BALL, Thorsten. *TRAIN YOUR OWN OPENCV HAAR CLASSIFIER* [online]. 2013 [cit. 2017-04-06]. Dostupné z: http://coding-robin.de/2013/07/22/train-your-own-opencv-haar-classifier.html

[32] *Creating your own Haar Cascade OpenCV Python Tutorial* [online]. 2016 [cit. 2017-04-06]. Dostupné z: https://pythonprogramming.net/haar-cascade-object-detection-python-opencv-tutorial/

[33] Cascade Classifier Training: *OpenCV* [online]. [cit. 2017-04-06]. Dostupné z: http://docs.opencv.org/3.0-beta/doc/user_guide/ug_traincascade.html

[34] BLAŽENKA, Aleš. *Aplikace Ziegler-Nicholsovy metody na diskrétní regulační obvody* [online]. Brno, 2007 [cit. 2017-04-17]. Dostupné z: http://autnt.fme.vutbr.cz/szz/2007/BP_Blazenka.pdf. Vedoucí práce Ing. Olga Davidová, Ph.D.

[35] LUKÁŠ, BC. HESS. *NÁVRH DVOUKOLOVÉHO AUTONOMNÍHO ROBOTA* [online]. BRNO, 2013 [cit. 2017-04-17]. Dostupné z: http://autnt.fme.vutbr.cz/szz/2013/DP_Hess.pdf. Vedoucí práce ING. DANIEL ZUTH, PH.D.

Seznam použitých zkratek

API	Application Programming Interface
BSD	Berkeley Software Distribution
CPU	Central Processing Unit
CV	Computer Vision
ECM	Electronically Commutated Motors
ESC	Electronic Speed Controller
fps	Frames Per Second
GPS	Global Positioning System
GUI	Graphical User Interface
IMU	Inertial Measurement Unit
IoT	Internet of Things
lib	Library
LNA	Low Noise Amplifier
LTS	Long Term Support
MAV	Mikrokopter Aerial Vehicle
MCU	Microcontroller Unit
OS	Operating System
PA	Power Amplifier
PC	Personal Computer
PID	Proportional Integral Derivative
PWM	Pulse Width Modulation
RGB	Red Green Blue
UAV	Unmanned Aerial Vehicle
UI	User Interface
USB	Universal Serial Bus

Seznam obrázků, tabulek a grafů

Obrázky

Tabulky

Grafy

Přílohy

Přiložené CD obsahuje elektronickou podobu práce ve formátu PDF a zdrojové kódy jednotlivých částí aplikace. Zdrojové kódy jsou ve formě prostého textu.

Elektronická podoba bakalářské práce je umístěna v adresáři `/dokumenty`.

Soubory týkající se testování jsou umístěny ve složce `/testovani`.

Zdrojové kódy jednotlivých částí aplikace jsou uloženy v adresáři `/software`.

`/software` hlavní zdrojové kódy jednotlivých příkladů

`/software/src` vedlejší zdrojové kódy

`/software/src/samples` převzaté zdrojové kódy