VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
FACULTY OF MECHANICAL ENGINEERING

ÚSTAV AUTOMATIZACE A INFORMATIKY
INSTITUTE OF AUTOMATION AND COMPUTER SCIENCE

NÁVRH MODELU KVADROKOPTÉRY
DESIGN OF A QUADCOPTER MODEL

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE — **Marek Provazník**
AUTHOR

VEDOUCÍ PRÁCE — **Ing. Tomáš Hůlka**
SUPERVISOR

BRNO 2020

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA STROJNÍHO INŽENÝRSTVÍ

Zadání bakalářské práce

Ústav:	Ústav automatizace a informatiky
Student:	**Marek Provazník**
Studijní program:	Strojírenství
Studijní obor:	Základy strojního inženýrství
Vedoucí práce:	**Ing. Tomáš Hůlka**
Akademický rok:	2019/20

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Návrh modelu kvadrokoptéry

Stručná charakteristika problematiky úkolu:

Úkolem studenta bude nejprve analyzovat dynamiku pohybu kvadrokoptéry. Poté student navrhne a vytvoří simulační model kvadrokoptéry, který následně otestuje na definované úloze.

Cíle bakalářské práce:

Stručná rešerše problematiky.
Vytvoření simulačního modelu.
Otestování funkčnosti na definované úloze.

Seznam doporučené literatury:

LUUKKONEN, T. "Modelling and control of quadcopter." Independent research project in applied mathematics, Espoo 22 (2011).

ARGENTIM, L. M., et al. "PID, LQR and LQR-PID on a quadcopter platform." 2013 International Conference on Informatics, Electronics and Vision (ICIEV). IEEE, 2013.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2019/20

V Brně, dne

L. S.

doc. Ing. Radomil Matoušek, Ph.D. ředitel ústavu	doc. Ing. Jaroslav Katolický, Ph.D. děkan fakulty

ABSTRAKT

Tato práce se zabývá návrhem modelu kvadrokoptéry ve frameworku ROS (Robot Operating System) s použitím simulačního prostředí Gazebo a RViz. Obsahuje stručnou rešerši o dronech jako takových, dynamice pohybu kvadrokoptéry a základy frameworku ROS. Praktická část se zaměřuje na zdrojový kód prostředí a modelu, ovladače motorů a řízení kvadrokoptéry.

ABSTRACT

This thesis is about design of quadcopter model using framework ROS (Robot Operating System) and simulation environment Gazebo and RViz. It contains brief search about drones in general, dynamics of quadcopter and ROS framework basics. The practical part focuses on code of the world, model, actuators controllers and regulation of quadcopter.

KLÍČOVÁ SLOVA

Model kvadrokoptéry, řízení kvadrokoptéry, ROS, základy ROS, Gazebo, RViz, dron

KEYWORDS

Design of quadcopter, regulation of quadcopter, ROS, ROS basics, Gazebo, RViz, drone

BIBLIOGRAFICKÁ CITACE

PROVAZNÍK, Marek. *Návrh modelu kvadrokoptéry*, Brno, 2020. Bakalářská práce. Vysoké učení technické v Brně, Fakulta strojního inženýrství, Ústav automatizace a informatiky. Vedoucí bakalářské práce Ing. Tomáš Hůlka.

PODĚKOVÁNÍ

Tímto bych rád poděkoval vedoucímu práce Ing. Tomáši Hůlkovi za rady, pomoc a možnost psát toto téma. Dále bych chtěl poděkovat všem lektorům, se kterými jsem se za dobu svého studia setkal a od kterých jsem se mohl učit. Velké díky patří také přátelům a rodině za obrovskou podporu v mém studiu.

ČESTNÉ PROHLÁŠENÍ

Prohlašuji, že tato práce je mým původním dílem, zpracoval jsem ji samostatně pod vedením Ing. Tomáše Hůlky a s použitím literatury uvedené v seznamu literatury.

V Brně dne 25. 6. 2020 ……

Marek Provazník

OBSAH

1 ÚVOD ... 1

2 DYNAMIKA POHYBU KVADROKOPTÉRY ... 3
2.1 Základní princip ... 3
2.2 Matematický model ... 3
2.2.1 Vznášení ... 4
2.2.2 Natočení kolem osy z ... 6
2.2.3 Pohyb vpřed a vzad ... 8
2.2.4 Natočení vlevo a vpravo ... 9
2.2.5 Vertikální pohyb ... 9

3 ROS ... 11
3.1 Úvod ... 11
3.2 Packages ... 11
3.3 Workspace ... 12
3.4 Master ... 13
3.5 Nodes ... 13
3.6 Topics ... 14
3.7 Launch files ... 16

4 MODEL ... 19
4.1 Dokumentace ... 19
4.1.1 Návrh vrtule ... 19
4.1.2 Sestava ... 19
4.1.3 Rozměry ... 21
4.2 Kód ... 21
4.2.1 Rviz ... 22
4.2.2 Tělo konstrukce ... 23
4.2.3 Výpočet momentu setrvačnosti ... 25
4.2.4 Vrtule ... 28
4.2.5 Transmission ... 30
4.2.6 ROS Control ... 30
4.2.7 Lift-Drag Plugin ... 32
4.2.8 Materiály ... 33
4.2.9 Kamera ... 34
4.2.10 TF graf ... 36

5 SIMULACE ... 37
5.1 Launch ... 37
5.2 Řízení ... 38

6 ZÁVĚR ... 41

7 SEZNAM POUŽITÉ LITERATURY ... 43

8 SEZNAM ZKRATEK A SYMBOLŮ ... 45

9 SEZNAM PŘÍLOH ... 47

1 ÚVOD

Drony v dnešní době stále více pronikají do života lidí za účelem zábavy nebo dokonce zjednodušení různých pracovních činností. Využívají se jak ve vojenské, tak i v civilní sféře, jsou schopny autonomně vykonávat složité operace, a dokonce i jako letka spolupracovat na jednom úkolu.

Tato bakalářská práce má za cíl analyzovat dynamiku pohybu kvadrokoptéry a vytvořit její matematický model tak, aby podle něj bylo možné porozumět problematice jejího pohybu a použít ho jako řešení v praktické části. Dalším cílem je nastudovat a stručně sepsat rešerši o základních principech i fungování frameworku ROS a pak pomocí něj a simulačního prostředí Gazebo vytvořit model kvadrokoptéry a otestovat na vhodné úloze.

Práce se skládá ze dvou hlavních částí: první rešeršní část se zabývá jednotlivými základními pohyby kvadrokoptéry a matematicky je popisuje. Shrnuje a vysvětluje fungování „nodes," „topics," „messages" a jiných elementárních prvků ROS.

Druhá praktická část se zaměřuje na model kvadrokoptéry a svět vytvořený v jazyce XML, ovladače pro motory a menší YAML soubory, pluginy pro simulační prostředí Gazebo v jazyce C++ a v jazyce Python menší kódy pro řízení a pomocné výpočty při konstrukci.

2 DYNAMIKA POHYBU KVADROKOPTÉRY

2.1 Základní princip

Kvadrokoptéra nebo také kvadrotor je bezpilotní letoun s šesti stupni volnosti, může se pohybovat ve směru každé osy a také kolem všech rotovat [1]. Pohyb dronu je založen na elektromotorech, které uvedou čepele do rotačního pohybu, jejich listy pak vytváří vztlak. Vztlaková síla se dá určit pomocí rovnice pro výpočet vztlaku:

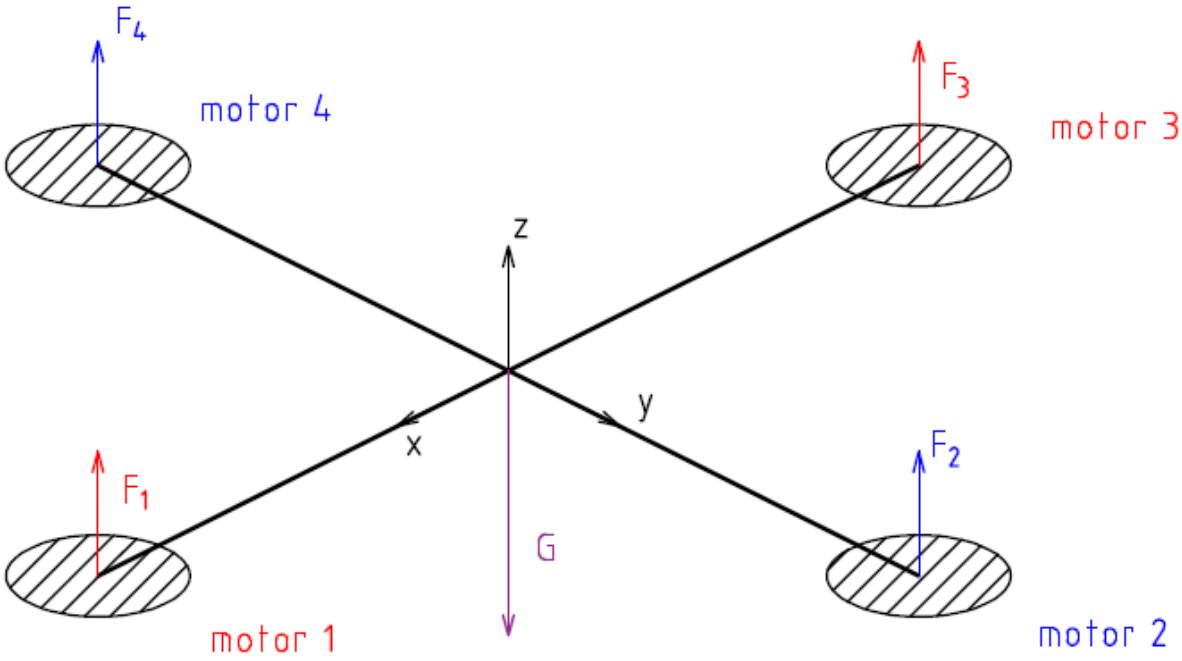
$$F_l = \frac{1}{2} \cdot \rho \cdot v^2 \cdot S \cdot c_L \tag{2.1}$$

- F_l [N] – vztlaková síla
- ρ [kg·m^{-3}] – hustota vzduchu
- v [m·s^{-1}] – rychlost
- S [m^2] – povrch listů čepele
- c_L [-] – součinitel vztlaku

Ze vzorce (2.1) je zřejmé, že můžeme ovlivnit velikost vztlakové síly pouze konstrukcí listů čepele nebo rychlostí jejich otáčení [2]. Za letu není možné zvětšovat povrch těchto listů ani úhel jejich náběhu, a proto je regulace pohybu uskutečněna pouze zvyšováním a snižováním otáček jednotlivých motorů v závislosti na vyžadované akci [3].

2.2 Matematický model

U kvadrokoptéry rozeznáváme několik základních druhů pohybu. Souřadnicový systém je zaveden podle obrázku (viz obr. 1). Vertikální pohyb (v ose z) pro vzlétání a přistávání, při pohybu v tomto směru se objekt nenatáčí kolem žádné osy. Natočení kolem osy z (anglicky yaw). Pohyb vpřed (anglicky pitch) je definován jednotkovým vektorem i_f= [1; 1] a je ho docíleno dočasným vyvedením dronu z momentové rovnováhy, které vede k natočení dronu kolem osy kolmé na vektor i_f. Na stejném principu funguje i pohyb vzad, který je definovaný vektorem v přesně opačném směru tzn. ve směru vektoru i_b= [-1; -1]. Naklopení vlevo a vpravo (anglicky roll) fungují na stejném principu vyvedení z momentové rovnováhy jako pohyb dopředu a dozadu. Směry jejich pohybu jsou určeny dvěma vektory, pro pohyb doleva i_l= [-1; 1] a vektorem i_r= [1; -1] pro pohyb doprava [1, 4].



Obr. 1: Schéma kvadrokoptéry

Model kvadrokoptéry byl vytvářen za těchto předpokladů:

- Konstrukce a vrtule jsou tuhé, neohebné
- Těžiště (působiště tíhové síly) je totožné s počátkem [0, 0, 0] modelu
- Vztlak a odpor je přímo úměrný druhé mocnině rychlosti vrtule
- Model je osově symetrický kolem osy z
- Faktory jako odpor vzduchu a vzdušné proudy jsou zanedbány

Při dodržení těchto podmínek lze popsat jednotlivé akce modelu rovnicemi v následujících kapitolách [1, 5].

2.2.1 Vznášení

Vznášení (anglicky hover) je rovnovážný stav modelu ve vzduchu, kdy se výslednice všech silových účinků (2.2), silových momentů (2.5) a (2.12) a momentů setrvačnosti rovná nule. Zároveň je nutno předpokládat nulovou počáteční rychlost ve všech směrech.

$$F_1 + F_2 + F_3 + F_4 - G = 0 \tag{2.2}$$

$$F_1 = F_2 = F_3 = F_4 \tag{2.3}$$

- G [N] – tíhová síla
- m [kg] – hmotnost kvadrokoptéry
- g [m·s^{-2}] – tíhové zrychlení (cca. 9,81 m/s^{-2})

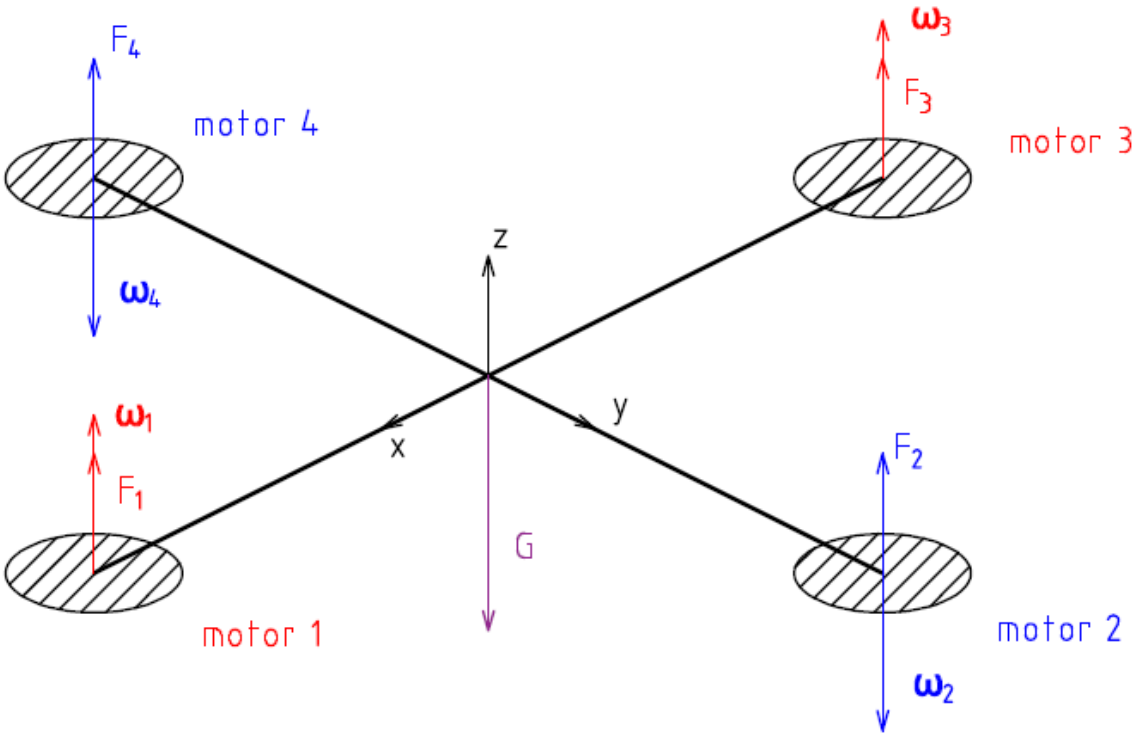
Součet všech vztlakových sil se rovná tíhové síle, která odpovídá součinu hmotnosti a tíhového zrychlení. Výsledná síla je nulová, a proto bude zrychlení v ose z také nulové. Za předpokladu, že počáteční rychlost kvadrokoptéry je v tomto směru nulová, nedojde k vertikálnímu pohybu [6].

Jediné síly vytvářející moment v ose x jsou síly F_2 a F_4, v ose y F_1 a F_3. Z rovnice (2.3) víme, že všechny síly mají stejnou velikost a z předpokladu symetrie je zřejmé, že všechna ramena, na kterých tyto síly působí jsou stejně dlouhá.

$$M_2 - M_4 = 0 \tag{2.5}$$
$$F_2 \cdot l - F_4 \cdot l = 0 \tag{2.6}$$
$$M_3 - M_1 = 0 \tag{2.7}$$
$$F_3 \cdot l - F_1 \cdot l = 0 \tag{2.8}$$

- M [N·m] – moment síly
- l [m] – vzdálenost působiště vztlakové síly motoru od těžiště modelu

Momenty sil se ve všech zmíněných osách odečtou, nevznikne tedy žádné úhlové zrychlení a model zůstává v rovnováze [6].



Obr. 2: Schéma vznášení

Obrázek 2 znázorňuje vektory úhlové rychlosti ω_1 a ω_3 v opačném směru než vektory ω_2 a ω_4, protože motor 1 a 3 se točí proti směru hodinových ručiček, zatímco motor 2 a 4 po směru hodinových ručiček. Dále si lze všimnout, že vztlakové síly od všech vrtulí působí v kladném směru osy z, i když se všechny netočí ve stejném smyslu.

Toto je způsobeno opačným profilem listů. Opačný smysl otáčení vrtulí je důležitý pro zachování rovnováhy momentů setrvačnosti [6].

$$I_1 = I_2 = I_3 = I_4 \tag{2.9}$$

$$|\omega_1| = |\omega_2| = |\omega_3| = |\omega_4| \tag{2.10}$$

$$L = I \cdot \omega \tag{2.11}$$

- I [kg·m^2] – moment setrvačnosti
- ω [rad·s^{-1}] – úhlová rychlost
- L [kg·m^2·s^{-1}] **–** moment hybnosti

Díky symetrii modelu a stejnému tvaru vrtulek jsou všechny momenty setrvačnosti stejné, jak uvádí rovnice (2.9). Z výše uvedených vztahů vyplývá, že všechny momenty hybnosti budou mít stejnou hodnotu, protože i velikosti úhlových rychlostí jsou shodné. Právě rozdílný směr vektoru úhlové rychlosti (tzn. rozdílný smysl otáčení) skupin 1, 3 (červená) a 2, 4 (modrá) zapříčiňuje, že součet momentů hybnosti bude nulový podle následující rovnice (2.12):

$$L_s = I_s \cdot \omega_s = I_1 \cdot \omega_1 - I_2 \cdot \omega_2 + I_3 \cdot \omega_3 - I_4 \cdot \omega_4 = 0 \tag{2.12}$$

Pokud se moment hybnosti soustavy rovná nule, je úhlová rychlost soustavy také nulová a soustava se nepohybuje. Soustava se nepohybuje ve směru žádné z os a zároveň se kolem ani jedné netočí, je tedy v klidu a podmínky pro tzv. vznášení jsou splněny [6].

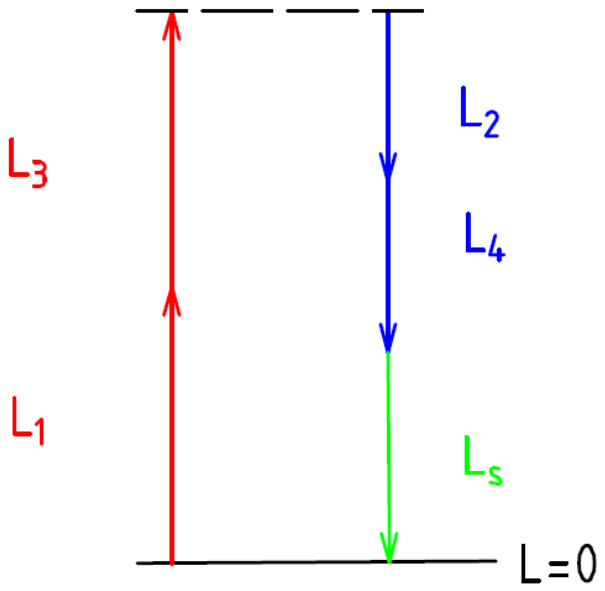
2.2.2 Natočení kolem osy z

Tato kapitola je zaměřena na natočení vzniklé v důsledku nerovnováhy momentů hybnosti. Pokud zvýšíme otáčky na motorech 1 a 3, které se točí proti směru hodinovýc h ručiček. V důsledku toho se začne model otáčet po směru hodinových ručiček. Tento jev je popsán a vysvětlen rovnicemi níže. Při zvýšení otáček n_1 a n_3 se zvýší úhlová rychlost a tím i vztlaková síla generovaná těmito vrtulemi. Je tedy zároveň potřeba snížit otáčky n_2 a n_4, aby nedošlo k silové nerovnováze ve směru osy z a tím pádem i k vertikálnímu pohybu. Pak bude dodržen stav, který vyjadřuje rovnice (2.2).

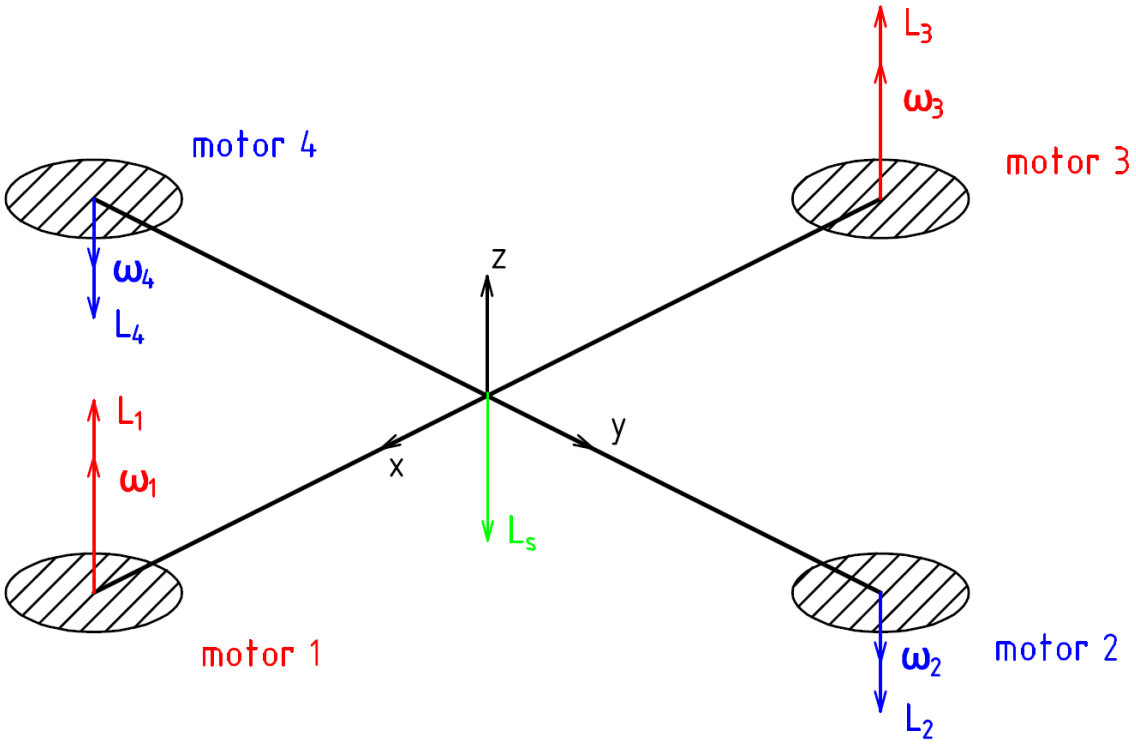
$$|\omega_1| = |\omega_3| > |\omega_2| = |\omega_4| \tag{2.13}$$

$$I_1 \cdot \omega_1 + I_3 \cdot \omega_3 - I_2 \cdot \omega_2 - I_4 \cdot \omega_4 - I_s \cdot \omega_s = 0 \tag{2.14}$$

$$\omega_s = \frac{I_1 \cdot \omega_1 + I_3 \cdot \omega_3 - I_2 \cdot \omega_2 - I_4 \cdot \omega_4}{I_s} \tag{2.15}$$



Obr. 3: Momenty hybnosti

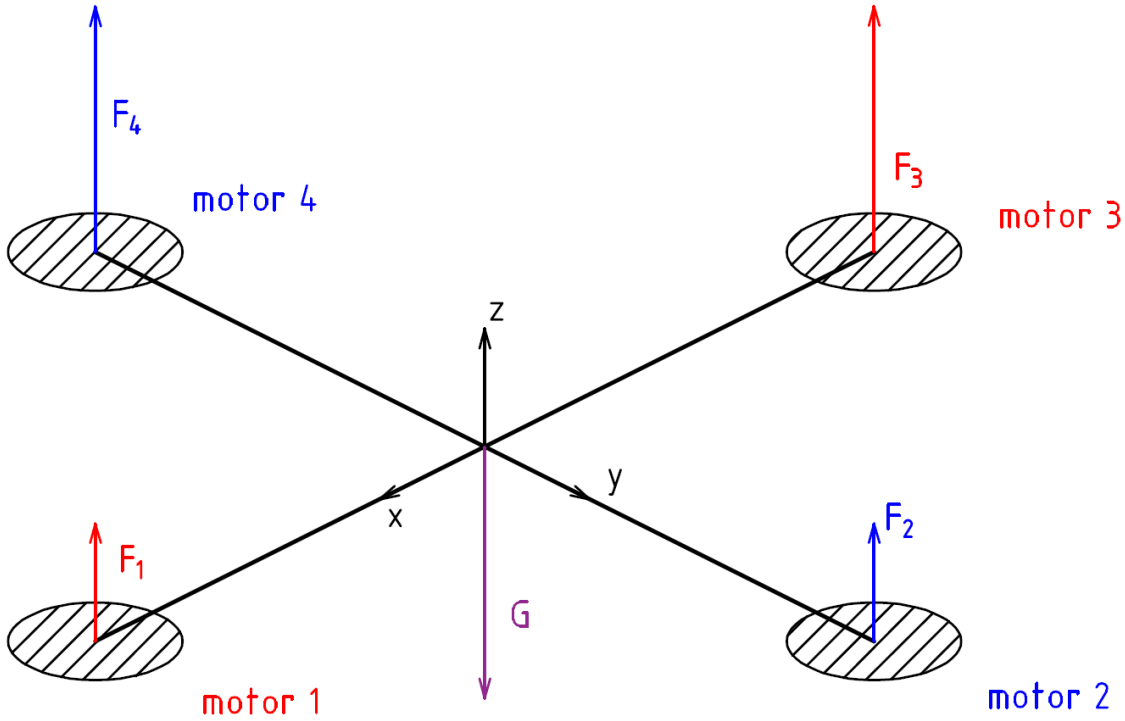


Obr. 4: Schéma natočení (yaw)

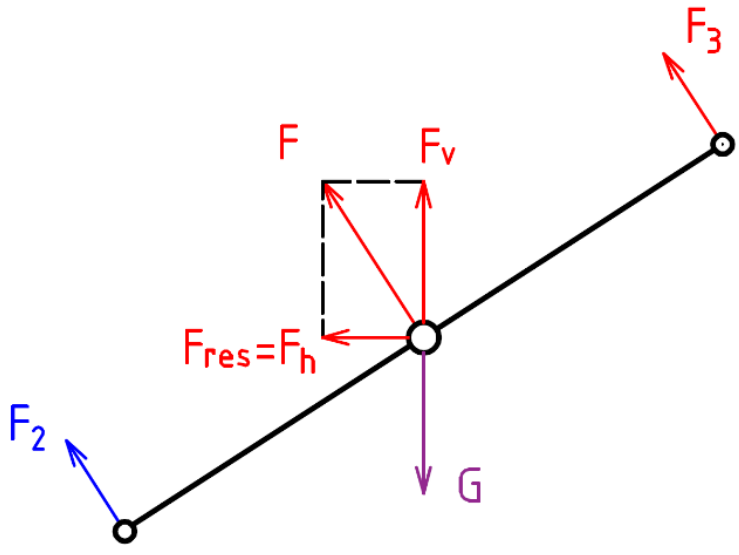
Pokud bude model v rovnovážném stavu, jak popisuje rovnice (2.12) a zvýšíme otáčky na dvou motorech, ekvivalentně se zvýší moment hybnosti celé soustavy. Ze zákona o zachování momentu hybnosti víme, že vznikne moment hybnosti soustavy L_s, který bude působit jako protiváha, jak znázorňují obrázky 3 a 4. V důsledku toho se začne celý model otáčet v opačném směru, než se točí čepele motorů se zvýšenými otáčky. Úhlová rychlost otáčení modelu kolem osy z je přímo úměrná rychlostem ω_1 a ω_3 a lze ji vypočítat pomocí vztahu (2.15), který je uveden výše [6].

2.2.3 Pohyb vpřed a vzad

Tyto akce jsou v souřadnicovém systému na obrázku 5 určeny vektory i_f= [1; 1] pro pohyb dopředu (anglicky forward) a i_b= [-1; -1] pro pohyb dozadu (anglicky backwards). Fungují na principu vyvedení z momentové rovnováhy zvýšením otáček na předních (1, 2)/zadních (3, 4) motorech. Celý dron se nakloní dopředu/dozadu. Jakmile se dostane do určitého úhlu, otáčky se zase vrátí do původního stavu. Vektor síly se rozloží do vertikální a horizontální složky. Vertikální složka síly se vyruší s tíhovou silou. Jak ukazuje obrázek 6, výsledná síla se rovná horizontální síle, a tak vzniká pohyb vpřed [6].



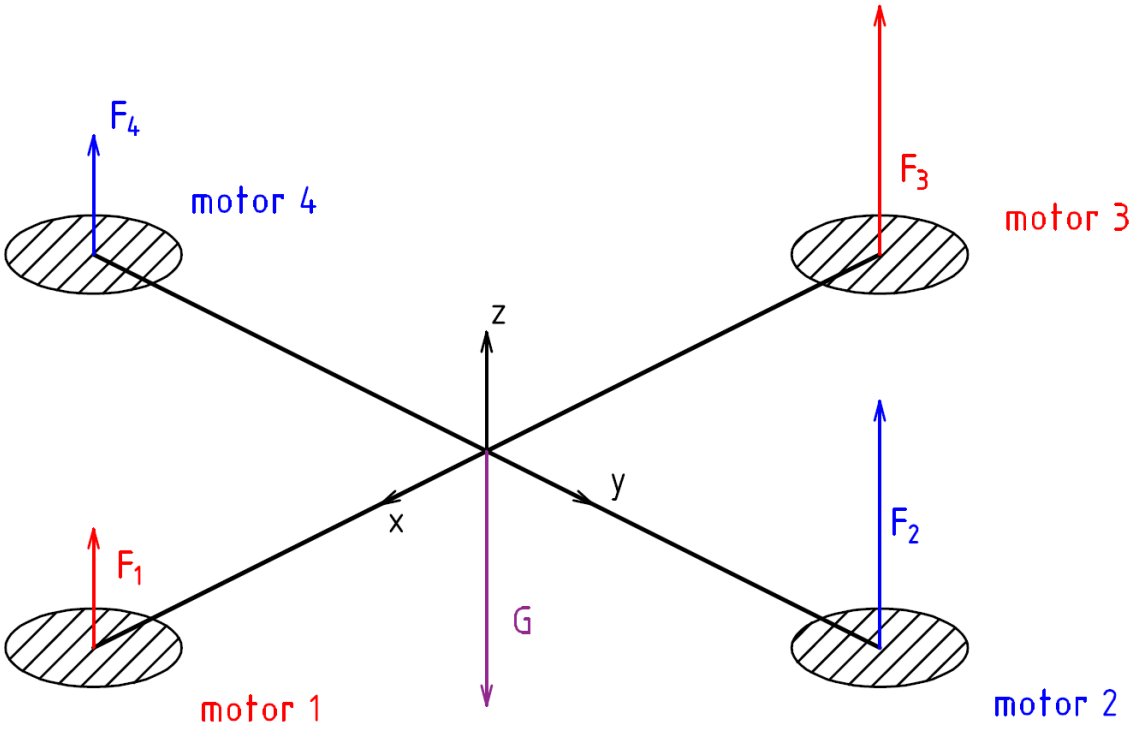
Obr. 5: Schéma dopředného pohybu



Obr. 6: Schéma výsledné síly při dopředném pohybu

2.2.4 Natočení vlevo a vpravo

Principiálně stejná operace jako pohyb vpřed a vzad. Rozdíl je ve skupinách motorů, u kterých jsou zvýšené otáčky, aby se kvadrokoptéra naklonila a začala se pohybovat do strany. Pro pohyb doleva je aktivována skupina motorů 1 a 4. Tento pohyb je definovaný jednotkovým vektorem i_l= [-1; 1]. Obrázek 7 znázorňuje uspořádání vztlakových sil od jednotlivých motorů pro pohyb doprava, který je určen jako i_r=[1; -1] [6].



Obr. 7: Schéma natočení do strany

2.2.5 Vertikální pohyb

Pohybu nahoru a dolů, tj. ve směru osy z je docíleno zvýšením/snížením otáček na každém z motorů. Rozdíl otáček musí být na každém z motorů stejný, aby byla zachována rovnováha momentů sil a momentů hybností. Tyto stavy reprezentuje obrázek 1 a blíže ho specifikují rovnice (2.16) pro stoupání a (2.17) pro klesání.

$$F_1 + F_2 + F_3 + F_4 > G \quad (2.16)$$

$$F_1 + F_2 + F_3 + F_4 < G \quad (2.17)$$

$$F_i = m \cdot a_i \quad (2.18)$$

$$a_1 = a_2 = a_3 = a_4 = a \quad (2.19)$$

$$m \cdot (a_1 + a_2 + a_3 + a_4) = m \cdot g \quad (2.20)$$

$$a_{res} = 4a - g \quad (2.21)$$

- a [m·s^{-2}] – zrychlení

Výsledné zrychlení v tomto směru je možné zjistit z (2.22). Ze vztahů (2.3) a (2.18) je možné vyvodit (2.19). Hmotnosti uvážené v (2.20) se vykrátí, při převedení zbývajících členů na jednu stranu rovnice dostaneme hodnotu celkového zrychlení. Pokud je hodnota kladná, bude soustava zrychlovat v kladném směru osy a nabere výšku. V opačném případě budou motory zpomalovat svůj pád a tím je možné kontrolovat klesání. Při dodržení předpokladů pro matematický model kvadrokoptéry je možné tvrdit, že při dosažení určité rychlosti a snížení otáček tak, aby výsledná síla byla nulová, se bude model pohybovat bez zrychlení tzn. konstantní rychlostí. Při testování na reálném modelu by toto tvrzení nebylo pravdivé, protože nepočítá s odporem vzduchu a vzdušnými proudy [6].

3 ROS

3.1 Úvod

ROS je zkratka pro Robot Operating System. Není to však operační systém [7]. Různé zdroje definují ROS různými způsoby, nejčastěji jako framework. „*Framework (pracovní rámec) je softwarový nástroj slučující dohromady návrhové vzory a objektově – orientované programování. Je to znovupoužitelný programový návrh vyjádřený množinou abstraktních tříd a vztahů mezi jejich instancemi. Vývojář může framework přizpůsobit určité aplikaci použitím a složením instancí tříd frameworku. Framework určuje architekturu aplikace. Definuje její strukturu, rozčlenění do tříd a objektů, klíčové vlastnosti mezi nimi a jejich kontrolu. Díky těmto předdefinovaným návrhovým parametrům se může vývojář soustředit na specifické rysy aplikace. Framework se zabývá obecně známými návrhovými rozhodnutími patřícími do domény rysů určité aplikace. Je zde také kladen důraz na opětovné použití naimplementovaného návrhu řešení problému* [8].“ Jiné zdroje definují ROS jako middleware. „*Middleware je programové vybavení či prostředí, které zajišťuje pro aplikace a programy některé funkce a služby nad rámec OS* [9].“ Podle [10] je ROS definován jako open-source, meta-operating systém pro roboty. Poskytuje služby jako operační systém včetně hardwarové abstrakce, nízkoúrovňového ovládání zařízení, implementace běžně používaných funkcí a předávání zpráv tzv. „messages“ mezi jednotlivými procesy a také mezi balíčky „packages“. ROS byl vytvořen pro operační systém Linux a je doporučeno ho na tomto OS používat. Nejčastěji použité programovací jazyky jsou C++ a Python [11].

3.2 Packages

Celý software je organizován do tzv. packages. ROS package je balíček souborů (složka) zahrnující spustitelné a podpůrné soubory, které slouží k určitému účelu. Je možné vytvořit vlastní package podle potřeby, software však obsahuje několik předinstalovaných balíčků. Lze jednoduše **zobrazit seznam** těchto **balíčků** následujícím příkazem v terminálu [11]:

```
rospack list
```

Každý balíček je definován souborem (manifest), který se nazývá *package.xml*. Obsahuje informace o balíčku jako je jeho jméno, verze, správce a jeho email a tzv. „dependencies“ neboli závislosti. Pro **vyhledání balíčku** s konkrétním jménem existuje příkaz:

```
rospack find „název_balíčku“
```

Název balíčku i s uvozovkami se přepíše na konkrétní název (platí i pro všechny níže uvedené příkazy). To není vždy potřeba vypsat celé. Pokud je vypsána část, která se neshoduje s žádným jiným pojmenováním, dvojstisk tabulátoru automaticky doplní celý název jediné zbývající možnosti [11].

Aby bylo možné následně **prohlédnout obsah** libovolného package, který byl, jak je popsáno výš, vyhledán, musí být zadán příkaz [12]:

```
rosls „název_balíčku“
```

Často je potřeba **změnit cestu** terminálu (tzn. ve které složce se provádí právě zadané příkazy) na náš package.

```
roscd „název_balíčku“
```

Použitím výše zmíněného příkazu se změní „adresa“, do které zadáváme příkazy [12].

Založení vlastního **balíčku** je nezbytné při programování robotů a vytváření jejich modelů. Následující řádek vytvoří nový balíček:

```
catkin_create_pkg „název_balíčku“
```

Každý package, který je vytvořen, musí být ve složce *src* příslušného workspace [11].

3.3 Workspace

Všechen software, včetně námi vytvořeného, je organizovaný v packages a ty zase ve workspace [11]. Každý workspace obsahuje tři základní složky (/build, /devel, /src), každá z nich má jiný účel. Pro vytvoření je potřeba založit novou složku s libovolným jménem. Bude použit název „catkin_ws“ podle [13].

```
mkdir -p ~/catkin_ws/src
cd ~/catkin_ws/src
catkin_init_workspace
```

První řádek založí složku „catkin_ws“ a v ní další „src“. Druhý řádek příkazem „cd“ (change direction) změní adresu terminálu. Poslední řádek inicializuje workspace.

```
cd ~/catkin_ws/
catkin_make
```

Následně se automaticky vytvoří složky /build a /devel. Aby bylo možné tento workspace používat a pracovat v něm, je nutné z něj udělat zdroj následujícím způsobem:

```
source ~/catkin_ws/devel/setup.bash
```

Tento příkaz je nutné provést vždy při změně workspace [13].

3.4 Master

ROS umožňuje spustit několik malých, na sobě navzájem nezávislých programů, které se nazývají „nodes“ a mohou být spuštěné zároveň. Aby mohly nodes takto fungovat, je nezbytná komunikace mezi nimi. **Master** se dá **spustit** takto:

```
roscore
```

Master nezpracovává žádná data, pouze uchovává informace o jednotlivých nodes, příslušné IP adresy a co je vysíláno (published)/přijímáno (subscribed). Master musí být spuštěný po celou dobu, co je ROS používán. Nodes se k Masteru připojují automaticky, pokud je spojení nějak přerušeno, nepokoušejí se dále navázat spojení. Master je možné ukončit v terminálu stisknutím CTRL+C, jakmile je master ukončen, všechny aktivní nodes přeruší spojení, které není navázáno při opětovném spuštění masteru [11].

Jiný způsob **spuštění masteru** je pomocí příkazu:

```
roslaunch „název_balíčku“ „název_souboru“
```

Dokáže otevřít několik nodes naráz a spustit master, pokud již neběží. Když ve chvíli zadání příkazu zaregistruje aktivní master, použije ho [11].

3.5 Nodes

Node je proces, který provádí výpočet. Nodes spolu navzájem komunikují pomocí topics, PRC services a Parametr Server. Řídící systém robotu zahrnuje spousty nodes, které běží zároveň. Je-li dron vybavený kamerou, motory nebo výškoměrem, pak každá z těchto věcí má svůj vlastní node, který ji řídí [14].

Používání nodes přináší ROS systému několik výhod. Pokud například nastane chyba, stane se tak pouze v rámci jednoho určitého node. Ostatní nodes, které nekomunikují a nejsou na chybné node závislé, mohou pracovat dál neovlivněny [14].

ROS nodes jsou napsány v jazycích C++ nebo Python [14].

Jak se píše v [11], spuštění node se dá uskutečnit dvěma způsoby. První a složitější je vypsat cestu přímo tak, jak se spouští kterýkoliv jiný program.

```
/opt/ros/melodic/lib/turtlesim/turtlesim_node
```

ROS má však vlastní příkaz, kde stačí vypsat dva parametry. První je package, kde se nachází náš node a druhý je název node [11]:

```
rosrun „název_balíčku“ „název_node“
```

ROS poskytuje určité informace o nodes, které zrovna běží. Pro vypsání listu všech právě aktivních nodes:

```
rosnode list
```

Ze seznamu, který se zobrazí, lze zjistit podrobnější informace o libovolném node:

```
rosnode info „název_node“
```

Pro ukončení nodes:

```
rosnode kill „název_node“
```

Na rozdíl od masteru nemá ukončení node velký vliv na ostatní nodes. Pokud si dva nodes vyměňují messages, je možné jednu z nich ukončit a znovu otevřít. Spojení bude znovu navázáno. Ekvivalent může být i stisknutí CTRL+C. Tento způsob neumožní node se odregistrovat z masteru, a proto se může stále zobrazovat v listu aktivních nodes.

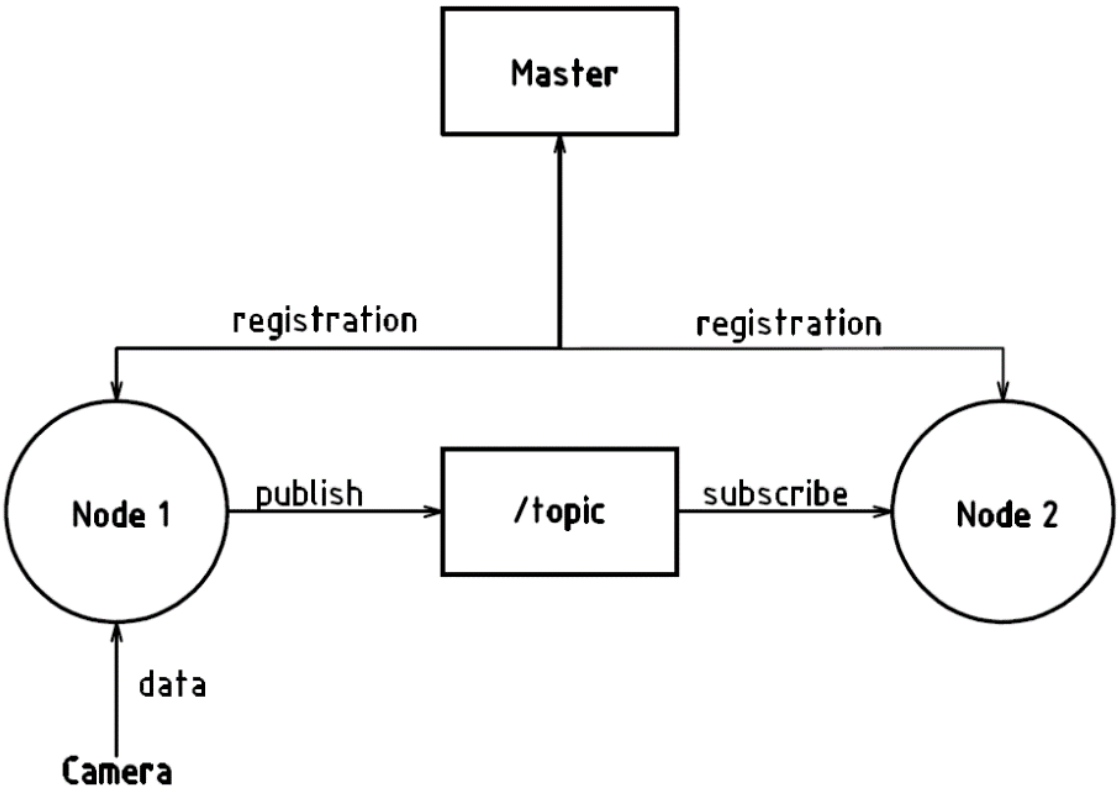
```
rosnode cleanup
```

Příkazem výše se lze zbavit všech „mrtvých“ nodes, jak popisuje [11].

3.6 Topics

Topics jsou „přepravníky“ sloužící k výměně messages (informací) mezi jednotlivými nodes. Topics publikují/odebírají data anonymně. Produkce těchto informací je oddělena od jejich dalšího užití, takže nodes publikující data na určité topics neví, které další nodes tato data odebírají. Počet nodes odebírajících určitý topic není omezený [15].

Master se stará o spojení publisheru (node, který vysílá zprávu) a subscriberu (node, který ji přijímá). Jak naznačuje obrázek 8, master ví, že existuje jeden konkrétní node (Node 1), který publikuje nějakou message na určitý topic (/topic). Druhý (Node 2), který chce tuto message přijmout, je také zaregistrovaný. Dá subscriberu (Node 2) vědět, že je tu publisher (Node 1), který vysílá zprávu na požadovaný topic a spojí je. Další komunikace už probíhá peer-to-peer mezi jednotlivými nodes (přímo od publisheru k subscriberu), takže master žádné z těchto informací nezpracovává [11].



Obr. 8: Schéma master, nodes, topics [16]

Užitečný nástroj, který zobrazí **graf** všech **nodes a topics**:

```
rqt_graph
```

Pro vypsání **seznamu** všech aktivních **topics**:

```
rostopic list
```

Zjistit, které nodes **publikují a odebírají** konkrétní **topic** lze příkazem:

```
rostopic info /„název_topic“
```

Do terminálu začne **tisknout data** publikovaná na určitý topic:

```
rostopic echo /„název_topic“
```

Pokud do následujícího příkazu poskytneme název topic, typ message a hodnotu, můžeme skrz terminál **publikovat** na konkrétní topic [11]:

```
rostopic pub /„název_topic“/„typ_message“ „hodnota“
```

3.7 Launch files

Launch files (česky spouštěcí soubory) slouží ke konfiguraci a spuštění více nodes najednou. Tyto soubory jsou velice užitečné při spouštění dvou a více nodes zaráz, protože při použití příkazu rosrun musíme spustit master a pak každý jeden node zvlášť, což zahrnuje otevření nového terminálu pro každou tuto akci (v příkladu uvedeném níže jeden terminál pro master a dva další pro turtle node a teleop_key node). Díky těmto souborům je možné udělat vše jedním příkazem, který ušetří spoustu času hlavně při debugování a opakovaném testování menších úprav na modelu [11].

Soubor je napsaný ve formátu XML a v tomto případě má příponu .launch. Jak ukazuje obrázek 10, je nutné, aby každý tag (např.: <launch>) byl uzavřen druhým tagem s lomítkem (např.: </launch>). Pokud toto pravidlo nebude dodrženo, bude se objevovat chyba zvaná „mismatched tag“ a soubor nebude možné spustit. Některé tagy mohou být uzavřeny jako tag node na obrázku 9, ale lze je zavřít i stejným způsobem jako tag launch, jak je uvedeno a podrobně vysvětleno v [11].

```
<node
type="název_node"
pkg="název_balíčku"
name="vlastní_název_node"
/>
```

Obr. 9: Node ve spouštěcím souboru [11]

Ke spuštění node přes launch file je nutné uvést minimálně tři argumenty, jako na obrázku 9. Podobně jak u příkazu rosrun uvádíme název balíčku (pkg-package), ve kterém se konkrétní node nachází. Název node je jméno spustitelného souboru (type). Vlastní název node (name) přepíše jméno node (může být libovolné, ale bez diakritiky), ať už se původně nazývá jakkoliv. Takto pak bude po spuštění node pojmenována v rosnode list, stejně jako na obrázku 11 [11].

catkin_ws > src > turtle > launch > t.launch

```
<launch>

<node
type="turtlesim_node"
pkg="turtlesim"
name="turtle"
/>

<node
type="turtle_teleop_key"
pkg="turtlesim"
name="teleop_key"
/>

</launch>
```

Obr. 10: Turtlesim launch file [11]

V předchozích kapitolách bylo popsáno, jak vytvořit workspace a package. Na obrázku 10 lze v horní části vidět, že byl vytvořen package „turtle“ ve workspace „catkin_ws“ a ve složce „launch“ soubor „t.launch“ spustitelný příkazem:

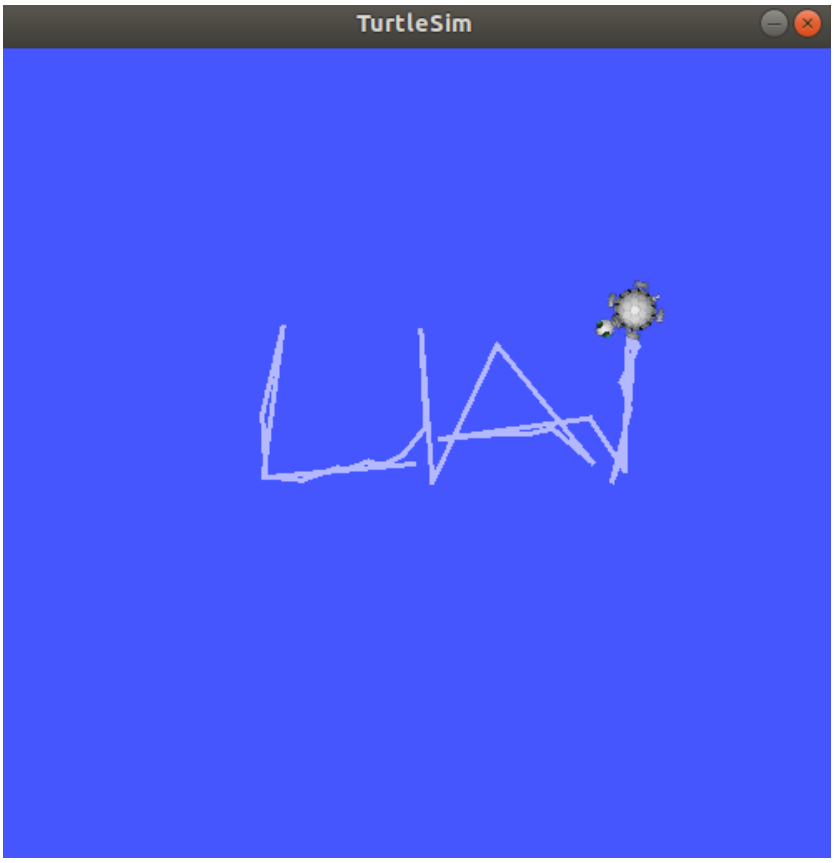
```
roslaunch „název_balíčku“ „název_souboru“
```

```
roslaunch turtle t.launch
```

Následně se spustí master a všechny nodes definované v tomto souboru. Výše uvedený kód spustí „turtle“ node, 2D svět se želvou ukazuje obrázek 12. Druhá node „teleop_key“ umožní v terminálu želvu ovládat pomocí šipek [11].

```
mary@Marvel:~/catkin_ws$ rosnode list
/rosout
/teleop_key
/turtle
```

Obr. 11: Rosnode list



Obr. 12: Turtlesim node

4 MODEL

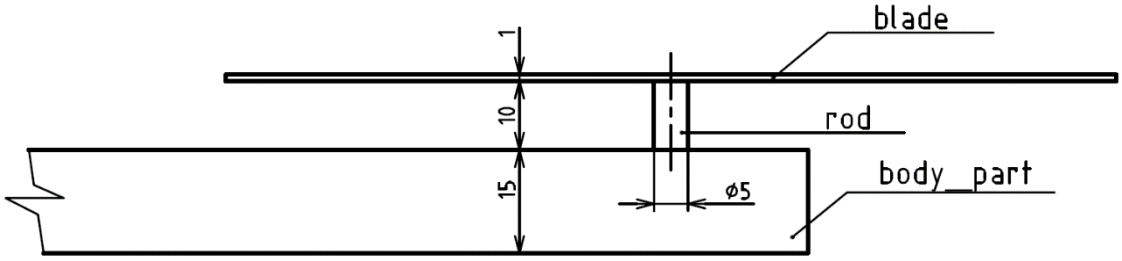
Tato kapitola je zaměřena na návrh modelu pro Gazebo. Gazebo je 3D dynamický simulátor, který dokáže simulovat roboty v komplexním prostředí. Umožňuje projektování robotů a testování jejich algoritmů s realistickými scénáři. Má k dispozici rozsáhlou knihovnu robotů a prostředí, širokou škálu senzorů a několik fyzických „enginů“ [17]. „*V počítačové terminologii označení kódu (například 3D grafiky), pod kterým je následně tvořen konkrétní software*.“ [18]. Je zde uvedena dokumentace rozměrů, kód a jeho vysvětlení.

4.1 Dokumentace

Rozměry modelu měly být navrženy tak, aby byl model co nejmenší a tím pádem měl co nejmenší hmotnost. Nakonec byl model zvětšen kvůli špatnému fungování simulace, jak je podrobněji vysvětleno v následujících kapitolách.

4.1.1 Návrh vrtule

Aby bylo možné roztočit vrtuli, je potřeba motor. Ve skutečném modelu by byl uložený někde v konstrukci (na výkrese část reprezentovaná názvem „body_part“). Tento model je zjednodušený, a proto funkci motoru plní hřídelka (rod). Jak je nakresleno na obrázku 13, hřídelka je zavazbena na těle konstrukce (body_part). Tato vazba má jeden stupeň volnosti, umožňuje rotační pohyb kolem osy hřídele. Na hřídeli je připevněná čepel, vazba mezi čepelí a hřídelkou nemá žádný stupeň volnosti, nemohou se vůči sobě hýbat (chovají se jako jedna součást). Všechny rozměry zakótované na obrázku 13 jsou uvedeny v milimetrech (platí i pro další výkresy).

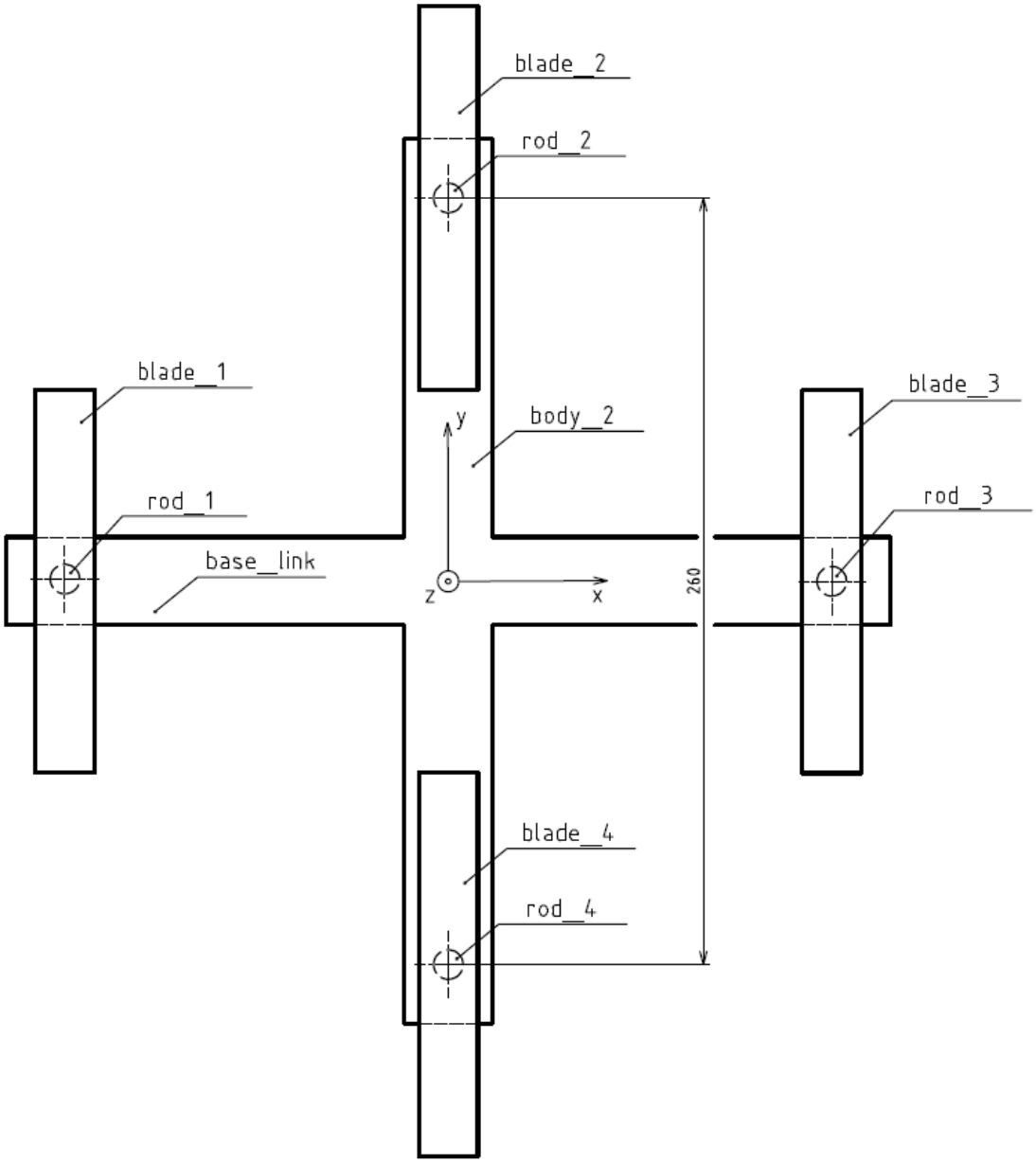


Obr. 13: Výkres vrtule

4.1.2 Sestava

Hlavním cílem práce bylo vytvořit funkční model kvadrokoptéry a odpovídajícím způsobem ho otestovat. Design modelu nebyl předmětem této práce, a proto mu nebyla věnována velká pozornost. Jelikož je model psán v jazyce XML a jeho formát je URDF (Unified Robot Description Format), bylo přistoupeno k nejjednoduššímu řešení grafické stránky modelu. Jak znázorňuje obrázek 13 a 14, kvadrokoptéra byla vytvořena za použití základních geometrických tvarů, od kterých se odvíjí simulace kolizí, hmotnost,

setrvačnost, moment setrvačnosti a další fyzikální veličiny potřebné k provedení simulace. Vzhled robotů je možné změnit na složité tvary pomocí tzv. „meshes“. Gazebo například používá formát .stl nebo collada soubory (.dae) [19]. Změní vizuální stránku modelu, ale výpočet simulace stále probíhá na základě definovaných geometrických tvarů.



Obr. 14: Výkres sestavy kvadrokoptéry

Výše uvedený obrázek 14 ukazuje, jak byl dron vymodelován. Je podle něj psán kód v kapitole 4.2. Byl sestaven z deseti základních částí (dvě součásti v konstrukci těla, čtyři hřídelky a čtyři čepele) a nehmotné kostičky umístěné kousek od počátku ve směru dopředného pohybu kvadrokoptéry. Tato kostička reprezentuje kameru a není zakreslena ve výkresu, protože slouží jen k vizualizaci a nemá žádné fyzické parametry.

4.1.3 Rozměry

Jelikož jsou všechny součásti základní geometrické tvary (konkrétně kvádry a válce) a způsob modelovaní odpovídá zadávání rozměrů ve směrech jednotlivých os kartézského souřadnicového systému, není pro každou součást zvlášť zpracován výkres. Místo toho jsou všechny rozměry uvedeny v tabulce 1 a 2.

Tab. 1: Tabulka rozměrů těl konstrukcí a čepelí

	base_link	body_2	blades
x[mm]	300	30	20
y[mm]	30	300	130
z[mm]	15	15	1

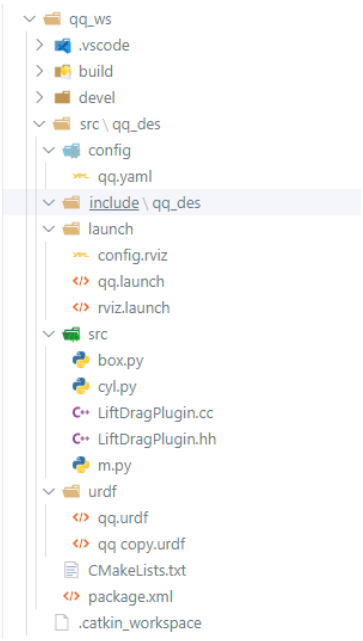
Tab. 2: Tabulka rozměrů hřídelí

	rods
r[mm]	5
h[mm]	10

Všechny rozměry v tabulkách jsou uvedeny v milimetrech. Tabulka 1 udává rozměry pro tělesa ve tvaru kvádru, tabulka 2 válce. Aby model odpovídal výkresu sestavy, musí souřadnice zapsané v kódu korespondovat s těmi v tabulce. Pokud dojde k prohození, může dojít v lepším případě k jinému uspořádání součástí, v horším k chybě a neúspěšnému spuštění modelu v simulačním prostředí Gazebo.

4.2 Kód

Prvním krokem ve vytváření našeho modelu bude založit pracovní prostředí. Na obrázku 15 je vidět, že workspace tohoto kódu nese pojmenování „qq_ws“ a je vytvořen stejným způsobem, jak popisuje kapitola 3.3. Jak již víme z kapitoly 3.2, všechny soubory ve frameworku ROS jsou umístěny v jednotlivých packages. Pro jednoduchost a možnost rychle spustit jakýkoliv soubor z tohoto package byl nazván „qq_des“ (z anglického description – česky popis). Package byl vytvořen ve složce „src“ stejným způsobem, jak je popsáno v příslušné kapitole. Dále je založena složka „urdf“, kde se nachází soubor „qq.urdf“ (formát XML).



Obr. 15: Workspace qq_ws

4.2.1 Rviz

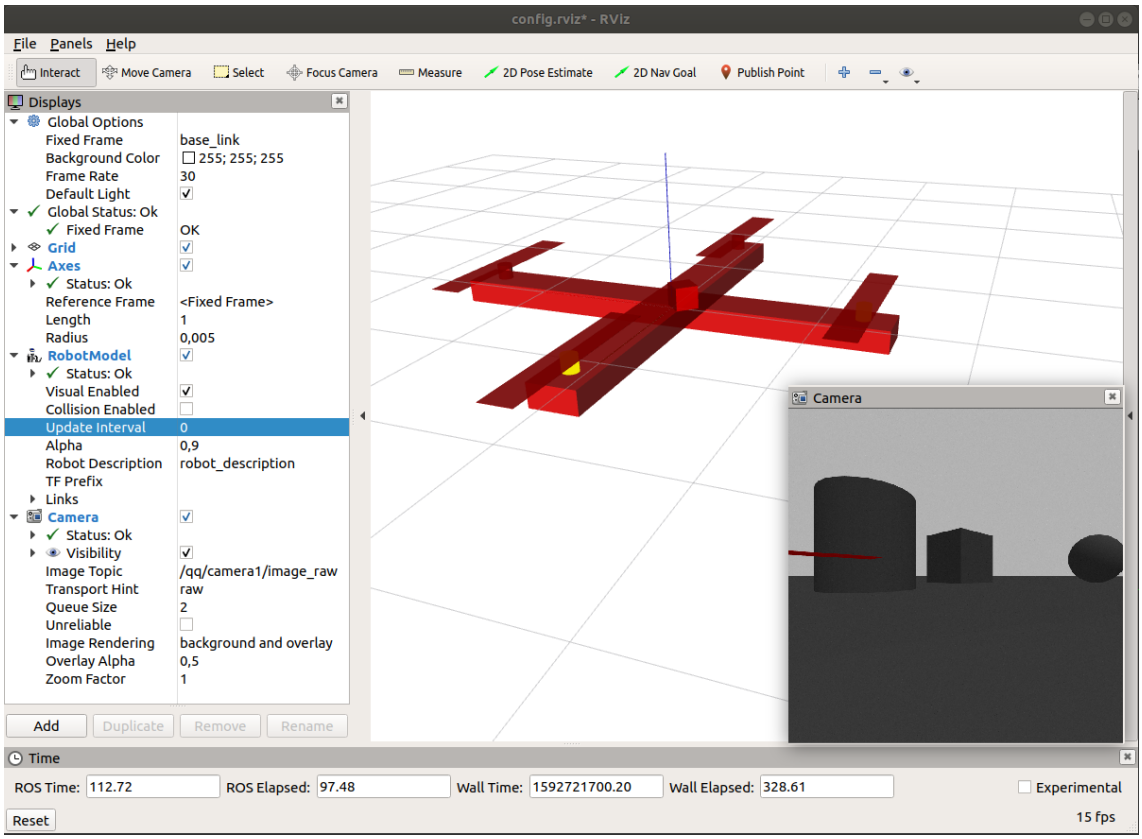
Rviz je spuštěn jako ROS node a je to nástroj pro 3D vizualizaci dat přijímaných z ostatních nodes. Tyto data mohou obsahovat například přenos obrazu z kamery nebo údaje z laseru či 3D senzoru [20]. Tento nástroj je velmi užitečný pří vyvíjení modelu, protože dokáže zobrazit počátek modelu a osy každé součásti. Jeho spuštění je rychlé (na rozdíl od Gazeba), a proto byl použit pro kontrolu každé právě přidané součásti. Při modelování je vhodné provádět kontrolu při každém přidání nové součásti, protože je pak mnohem snazší odhalit chybu.

Podle obrázku 15 založíme složku „launch“ a v ní soubor „rviz.launch“. Jak název napovídá, bude sloužit ke spuštění node Rviz. Soubor obsahuje následující kód:

Výpis 1: Ukázka souboru iniciujícího Rviz node

```
<launch>
    <param name="robot_description"
           textfile="$(find qq_des)/urdf/qq.urdf"/>

    <node name="robot_state_publisher"
          pkg="robot_state_publisher"
          type="robot_state_publisher"/>

    <node name="joint_state_publisher"
          pkg="joint_state_publisher"
          type="joint_state_publisher">
        <param name="joint_state_publisher_gui" value="True"/>
    </node>

    <node name="rviz"
          pkg="rviz"
          type="rviz"
          args="-d $(find qq_des)/launch/config.rviz"/>
</launch>
```

První část kódu (tag <param>) v svém poli „textfile“ specifikuje soubor, ze kterého má Rviz vzít informace pro svoji vizualizaci. Zbývající tagy spouští jednotlivé nodes. „Robot_state_publisher“ je node nezbytný k zobrazení jednotlivých částí modelu. Jelikož nejsou všechny vazby fixní, vrtulkou lze otáčet. Byl iniciován node „joint_state_publisher“, díky kterému je možné v prostředí Rviz otáčet vazbami (joint), které to umožňují. Jeho parametr spustí (value=“True“) joint_state_publisher_gui, který umožní uživateli otáčet příslušné jointy. Poslední node spouští samotný Rviz. Jeho argument načte předchozí uloženou konfiguraci. Nastavené parametry jsou tedy uloženy do složky „launch“ pod názvem „config.rviz“.



Obr. 16: Rviz

Pro zobrazení modelu jako na obrázku 16 je nutné přes tlačítko „Add“ ve spodní části přidat „RobotModel“. V sekci „GlobalOption“ se potřeba změnit „FixedFrame“ na base_link. Bez nodes robot_state_publisher a joint_state_publisher se jednotlivé části modelu nespojí a model nebude poskládán tak, jak je vyžadováno. V tuto chvíli sice není žádná část modelu vytvořená, ale prostředí Rviz je připraveno ke kontrole. Nastavení uložíme jak bylo zmíněno v předchozím odstavci (File > Save Config As).

4.2.2 Tělo konstrukce

Tělo konstrukce se skládá ze dvou kvádrů (link) a jedné vazby (joint). Tato vazba je pevná, nemá žádné stupně volnosti, protože základní část konstrukce musí být vytvořena jako jedna část. Všechny části kódu v této kapitole patří hned za sebe.

Výpis 2: Ukázka kódu těla konstrukce (část base_link)

```
<?xml version="1.0" ?>
<robot name="qq">
<!--Body-->
    <link name="base_link">
        <visual>
            <origin xyz="0 0 0" rpy="0 0 0"/>
            <geometry>
                <box size="3 .3 .15"/>
            </geometry>
        </visual>
```

```
    <collision>
       <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
            <box size="3 .3 .15"/>
        </geometry>
    </collision>
    <inertial>
        <mass value="1"/>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <inertia ixx="0.009374999999999998"   ixy="0"   ixz="0"
                 iyy="0.7518750000000001"     iyz="0"
                 izz="0.7575"/>
    </inertial>

</link>
```

Výpis 2 je úplným začátkem souboru „qq.urdf“. Nejprve je definována verze jazyka tagem <xml/>, poté je tagem <robot> otevřen model robotu. Tento tag je uzavřen úplně na konci souboru a neměl by za ním být žádný další tag. Vytvoříme nový link, ve kterém musíme definovat jeho vizuální vlastnosti. Část „base_link“ má svůj střed v počátku souřadného systému (xyz=“0 0 0“ – hodnoty v metrech) a není nijak natočena (rpy=“0 0 0“ – roll, pitch, yaw. Natočení v radiánech kolem příslušných os x, y, z). Tvar je definován tagem <box> jako kvádr o příslušných rozměrech (v metrech). Po uzavření tagu </visual> je soubor uložen [21]. Model je následně zkontrolován pomocí Rviz:

```
roslaunch qq_des rviz.launch
```

V části <collision> jsou uvedeny parametry pro výpočet kolizí v simulačním prostředí Gazebo. Tato sekce kódu bývá většinou totožná s obsahem tagu <visual>.

Tag <mass> definuje hodnotu hmotnosti v kilogramech a <inertial/> definuje matici momentu hybnosti. Výpočtem těchto hodnot se zabývá kapitola 4.2.3.

Výpis 3: Ukázka kódu těla konstrukce (část joint)

```
<joint name="joint" type="fixed">
    <axis xyz="0 0 1"/>
    <origin xyz="0 0 0" rpy="0 0 0"/>
        <parent link="base_link"/>
        <child link="body_2"/>
</joint>
```

Výpis 3 otevírá tagem <joint> vazbu se jménem „joint“. Tato vazba je typu „fixed“, tzn. že nemá žádný stupeň volnosti. Tag <axis> určuje vektor, v jehož směru bude osa vazby. Argument tagu <parent> určuje „mateřský“ link. Střed této součásti je výchozím bodem pro určování polohy jointu, jehož umístění v prostoru je zase počátkem

pro „dceřiný“ link. Tzn. že pokud změníme souřadnice v tagu <origin> (pro joint), posune se i střed „dceřiného“ linku.

Výpis 4: Ukázka kódu těla konstrukce (část body_2)

```
<link name="body_2">
    <inertial>
        <mass value="1"/>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <inertia ixx="0.7518750000000001"   ixy="0"   ixz="0"
                 iyy="0.009374999999999998" iyz="0"
                 izz="0.7575"/>
    </inertial>
    <collision>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
            <box size=".3 3 .15"/>
        </geometry>
    </collision>
    <visual>
        <origin xyz="0 0 0" rpy="0 0 0"/>
        <geometry>
            <box size=".3 3 .15"/>
        </geometry>
    </visual>
</link>
```

Specifikace linku „body_2“ ukazuje výpis 4. Jeho střed bude totožný s částí „base_link“, protože <origin> tag jointu i „body_2“ linku je nulový. Jediný rozdíl mezi těmito dvěma díly je v argumentu tagu <box> (prohozené hodnoty souřadnic x a y, jak uvádí tabulka 1) a z toho vyplývající změna matice v <ineartia> tagu.

4.2.3 Výpočet momentu setrvačnosti

Pro výpočet matice momentu setrvačnosti byly použity dva vztahy. Jeden pro kvádr a druhý pro válec. Moment setrvačnosti kvádru je určen rovnicemi [6]:

$$I_x = \frac{1}{12} \cdot m_k \cdot (y^2 + z^2) \tag{4.1}$$

$$I_y = \frac{1}{12} \cdot m_k \cdot (x^2 + z^2) \tag{4.2}$$

$$I_z = \frac{1}{12} \cdot m_k \cdot (x^2 + y^2) \tag{4.3}$$

- I_x, I_y, I_z – momenty hybnosti k osám x, y, z
- m_k – hmotnost kvádru
- x, y, z – rozměry kvádru v příslušných osách

Ve složce „src“ je umístěn soubor „box.py“ s kódem, který vypočítá tyto hodnoty pro kvádr.

Výpis 5: Ukázka kódu pro výpočet momentů setrvačnosti kvádru

```
def box(m, x, y, z):
    ix = (m/float(12))*((y*y)+(z*z))
    iy = (m/float(12))*((x*x)+(z*z))
    iz = (float(m)/12)*((x*x)+(y*y))
    print('ix: ', ix)
    print('iy: ', iy)
    print('iz: ', iz)

m = input('mass= ')
x = input('x dimension= ')
y = input('y dimension= ')
z = input('z dimension= ')
box(m, x, y, z)
```

Kód, který ukazuje výpis 5 je napsaný v programovacím jazyku Python. Vstupní parametry jsou v kilogramech (pro hmotnost) a v metrech (pro délky v jednotlivých osách). Je definována funkce s název „box“, která obsahuje vzorce pro výpočet jednotlivých hodnot, které následně vypíše do terminálu.

Momenty setrvačnosti válce s osou rotace rovnoběžnou s osou z jsou vypočítány těmito rovnicemi (pokud by byla osa rotace rovnoběžná s jinou osou, je potřeba prohodit indexy – rovnice 4.4 odpovídá momentu setrvačnosti k ose, která je zároveň osou rotace) [6].

$$I_z = \frac{1}{2} \cdot m_v \cdot r^2 \tag{4.4}$$

$$I_x = I_y = \frac{1}{12} \cdot m_v \cdot (3r^2 + h^2) \tag{4.5}$$

- m_v – hmotnost válce
- r – poloměr válce
- h - výška válce

Výpis 6 je kód umístěný ve složce „src“ pod názve „cyl.py“. Je podobný kódu z výpisu 5, liší se vstupními hodnotami a výstupem, protože pro válec jsou hodnoty momentů hybnosti ke dvěma osám vždy totožné (pokud je osa rotace ve středu) [6].

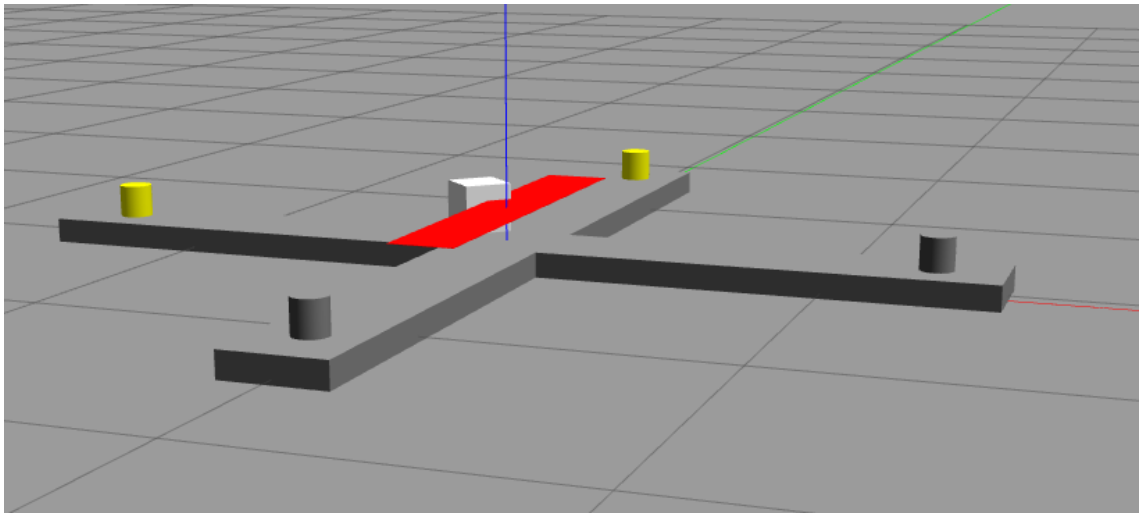
Pro spuštění kódu je nutné nastavit adresu terminálu do příslušné složky, kde se kód nachází. Pak se dá spustit následujícím příkazem:

```
python cyl.py
```

Výpis 6: Ukázka kódu pro výpočet momentů setrvačnosti válce

```
def cylinder(m, r, h):
    iz = (m*r*r)/float(2)
    ixy = (m/float(12))*((3*(r*r))+(h*h))
    print('iz: ',iz)
    print('ixy: ',ixy)

m = input('m= ')
r = input('r= ')
h = input('h= ')
cylinder(m, r, h)
```

V kódu jednotlivých linků jsou určeny rozměry v metrech. Tyto rozměry nesouhlasí s tabulkou v kapitole 4.1.3, kde je největší rozměr modelu stanoven na 300 milimetrů. Kvůli správnému fungování simulace muselo být přistoupeno ke změně modelu. Pokud jsou hodnoty momentu setrvačnosti příliš nízké (kvůli malým rozměrům nebo hmotnosti), může dojít k nestabilitě modelu a dokonce ke „zhroucení“ modelu do počátku jako na obrázku 17.



Obr. 17: Nestabilita simulace modelu

Tento jev byl zpočátku vyřešen zvýšením hmotnosti modelu na extrémně vysokou hodnotu. Podařilo se sice dosáhnout stability modelu, ale hustota materiálu dosahovala (vzhledem ke svému malému objemu) nereálných hodnot. Později nastaly problémy se vzletem modelu pomocí „LiftDragPluginu“. Kvůli své velké hmotnosti nemohl model vzlétnout dokonce ani při modifikaci parametrů pluginu na enormní hodnoty (podrobněji popsáno níž) a zvýšených otáčkách motorů. Bylo přistoupeno ke zvětšení rozměrů dronu a tím i momentů setrvačnosti. Poměry délek byly zachovány. Všechny rozměry byly zvětšeny o jeden řád (model v simulaci je desetkrát větší než je zakresleno v dokumentaci).

4.2.4 Vrtule

Jelikož je kód v této části téměř identický pro každou vrtuli (liší se v umístění) a rozsáhlý, bude zde uveden jen pro jednu vrtuli (kompletní kód v příloze). Sledování dokumentace vrtule v kapitole 4.1.1. pomůže lépe pochopit kód. Pro ukázku je vybrána vrtule 2, protože souhlasí s výkresem na obrázku 13. Skládá se ze čtyř částí.

Výpis 7: Ukázka kódu vrtule (část joint_2)

```
<!--2-->
    <joint name="joint_2" type="fixed">
        <axis xyz="0 0 1"/>
        <origin xyz="0 1.3 .075" rpy="0 0 0"/>
        <parent link="body_2"/>
        <child link="rod_2"/>
    </joint>
```

Výpis 7 je začátkem kódu druhé vrtule. Jedná se o joint s názvem „joint_2“. Názvy částí všech vrtulí se liší pouze číslem. Tento joint je pevně přidělán k části konstrukce „body_2“. Zde také došlo ke změně oproti dokumentaci, otáčet se bude jen vrtulka, hřídelka zůstane nepohyblivá. Bylo tak učiněno kvůli bugům v simulaci. Osa jointu je rovnoběžná s osou z a je posunuta o 1,3 metrů na ose y a 0,075 metrů (polovina z výšky „body_2“) na ose z tak, aby byla na povrchu součásti „body_2“ (viz dokumentaci).

Výpis 8: Ukázka kódu vrtule (část rod_2)

```
    <link name="rod_2">
        <inertial>
            <mass value=".1"/>
            <origin xyz="0 0 .05" rpy="0 0 0"/>
            <inertia    ixx="0.0003958333333333334" ixy="0" ixz="0"
                        iyy="0.0003958333333333334" iyz="0"
                        izz="0.00012500000000000003"/>
        </inertial>
        <collision>
           <origin xyz="0 0 .05" rpy="0 0 0"/>
            <geometry>
                <cylinder radius="0.05" length="0.1"/>
            </geometry>
        </collision>
        <visual>
            <origin xyz="0 0 .05" rpy="0 0 0"/>
            <geometry>
                <cylinder radius="0.05" length="0.1"/>
            </geometry>
        </visual>
    </link>
```

Kód linku „rod_2“ (viz výpis 8) je podobný jako již zmíněné linky. Hlavní rozdíl je ve tvaru, který je definován tagem <cylinder> (válec) a jeho argumenty „radius“ (zaoblení) a „lenght“ (délka). Tagem, <origin> je součást posunuta o 0,05 metrů na ose z. Tato vzdálenost odpovídá polovině výšky válce. Bylo tak učiněno vzhledem k poloze jointu, který spojuje link s modelem (na povrchu „body_2“). Takto určený válec přesně dosedne na tělo konstrukce.

Výpis 9: Ukázka kódu vrtule (část joint_22)

```
<joint name="joint_22" type="continuous">
    <axis xyz="0 0 1"/>
    <origin xyz="0 0 .1" rpy="0 0 0"/>
    <parent link="rod_2"/>
    <child link="blade_2"/>
</joint>
```

Joint ve výpisu 9 je typu „continuous“ (česky plynulý). Tento typ jointu umožňuje neomezené otáčení ve smyslu otáčení hodinových ručiček i proti němu (má jeden stupeň volnosti). Není zde nastavena žádná horní ani spodní hranice. Na součásti přidělané tímto jointem bude působit gravitační síla a různé vnější vlivy (např.: vítr). Pokud v tomto jointu nebude nastaven a zapnut žádný motor, může se kolem této osy otáčet v důsledku působení těchto vlivů [21].

Výpis 10: Ukázka kódu vrtule (část blade_2)

```
<link name="blade_2">
    <inertial>
        <mass value=".04"/>
        <origin xyz="0 0 .0005" rpy="0 0 0"/>
         <inertia ixx="0.005633336666666667"  ixy="0" ixz="0"
                  iyy="0.00013333666666666667"  iyz="0"
                  izz="0.005766666666666667"/>
    </inertial>
    <collision>
        <origin xyz="0 0 .0005" rpy="0 0 0"/>
        <geometry>
            <box size=".2 1.3 .001"/>
        </geometry>
    </collision>
    <visual>
        <origin xyz="0 0 .0005" rpy="0 0 0"/>
        <geometry>
            <box size=".2 1.3 .001"/>
        </geometry>
    </visual>
</link>
```

Část „blade_2“ (česky čepel) je poslední částí vrtule. Na tuto součást je, jak je níže uvedeno, aplikován „LiftDragPlugin“ a generuje vztlak. Link je v poměru s ostatními díly velmi tenký, protože jako jediný nebyl zvětšen o řád kvůli bugům v simulaci, jak vysvětluje kapitola 4.2.4. Výchozí bod je na povrchu „rod_2“, protože sem byl přesunut vazbou „joint_22“ (vazba „joint_22“ samotná má počátek v „joint_2“, tzn. na povrchu „body_2“). Odtud je „blade_2“ přemístěn o 0,0005 metrů v kladném směru osy z (polovina jeho tloušťky) tak, aby dosedala na hřídelku „rod_2“.

4.2.5 Transmission

Transmission (česky přenos) je rozšíření URDF formátu, které je určeno k definování vztahu mezi ovladačem a jointem [23]. Výpis 11 ukazuje, jak vypadá <transmission> tag pro již vytvořenou vrtuli 2.

Výpis 11: Ukázka kódu Transmission

```
<transmission name="tran2">
  <type>transmission_interface/SimpleTransmission</type>
  <joint name="joint_22">
     <hardwareInterface>hardware_interface/VelocityJointInterface
     </hardwareInterface>
  </joint>
  <actuator name="motor_joint_22">
     <hardwareInterface>hardware_interface/VelocityJointInterface
     </hardwareInterface>
     <mechanicalReduction>1</mechanicalReduction>
  </actuator>
</transmission>
```

Argument name tagu <joint> musí přesně odpovídat názvu příslušného jointu („joint_22“). Důležitá část kódu je tag <hardwareInterface>, je to hardwarové rozhraní pro podporu příkazů jointů (česky kloubů). Základní typy tohoto rozhraní jsou:

- **Effort Joint Interface** – Pro ovládání jointů, založené na úsilí (effort).
- **Position Joint Interface** – Pro ovládání jointů, založení na poloze.
- **Velocity Joint Interface** – Pro ovládání jointů, založené na rychlosti.

V tomto projektu je potřeba otáčet vrtulí, která generuje vztlak, proto bude nejvhodnější použít Velocity Joint Interface [24, 25].

4.2.6 ROS Control

Výpis 12 ukazuje kód, který otevře plugin gazebo_ros_control. Tento plugin musí být přidán do URDF souboru. Analyzuje <transmission> tagy a načítá příslušné hardwarové rozhraní a správce ovladačů (controller_manager node). Umožní tak ovládání jointů. Pokud není tento plugin součástí Gazeba, je potřeba ho nainstalovat následujícím příkazem [25]:

```
sudo apt-get install ros-melodic-gazebo-ros-control
```

Výpis 12: Ukázka přidání pluginu gazebo_ros_control

```
<gazebo>
    <plugin name="gazebo_ros_control"
            filename="libgazebo_ros_control.so">
        <robotNamespace>/qq</robotNamespace>
    </plugin>
</gazebo>
```

Posledním krokem ke zprovoznění motorů je soubor „qq.yaml“ (kód ve výpisu 13), který deklaruje typy ovladačů pro příslušné jointy a určuje jejich hodnoty PID regulátoru. Podle obrázku 15 se soubor nachází ve složce „config“.

Výpis 13: Ukázka kódu definujícího typy ovladačů

```
qq:
  joint_state_controller:
    type: joint_state_controller/JointStateController
    publish_rate: 50

  joint_motor_controller:
    type: velocity_controllers/JointGroupVelocityController
    joints:
        - joint_12
        - joint_22
        - joint_32
        - joint_42
    /qq/gazebo_ros_control/pid_gains/:
        joint_12: {p: 10, i: 0.1, d: 0}
        joint_22: {p: 10, i: 0.1, d: 0}
        joint_32: {p: 10, i: 0.1, d: 0}
        joint_42: {p: 10, i: 0.1, d: 0}
```

Ve [25] je přehledně vypsán seznam všech různých přednastavených typů ovladačů (lze vytvořit i vlastní). Tato práce se kvůli zvolenému hardwarovému rozhraní zaobírá jen ovladači typu **velocity_controllers:**

- **Joint_velocity_controller** – Pro ovládání jednoho samostatného jointu pomocí message typu Float64
- **Joint_group_velocity_controller –** Pro ovládání skupiny jointů zároveň, používá message typu Float64MultiArray

Pro účely této práce byl zvolen ovladač typu Joint_group_velocity_controller, protože kvůli stabilitě kvadrokoptéry nutné ovládat všechny vrtule zároveň.

Jak je možné si všimnou ve výpisu 13, hodnoty PID jsou poměrně nízké. Z autorovi neznámého důvodu se při zvýšení kteréhokoliv z těchto hodnot jen o řád stane model v simulaci nestabilní a dojde k jeho „zhroucení“ (viz obrázek 17) jak již bylo popsáno v kapitole 4.2.3.

4.2.7 Lift-Drag Plugin

V kapitole 4.2.4 bylo popsáno, že čepel má tvar kvádru. Není sklopena pod určitým úhlem a nemá profil jako reálné vrtule. Gazebo samotné po spuštění nepočítá s aerodynamikou, a tak musí být přidán plugin „LiftDragPlugin“, který tyto vlastnosti definuje.

Výpis 14: Ukázka přidání LiftDragPluginu

```
<gazebo>
    <plugin name="LiftDragPlugin_2"
            filename="libLiftDragPlugin.so">
        <a0>0.1</a0>
        <cla>0.100</cla>
        <cda>0.01</cda>
        <cma>0.00</cma>
        <alpha_stall>0.2</alpha_stall>
        <cla_stall>-0.2</cla_stall>
        <cda_stall>1.0</cda_stall>
        <cma_stall>0.0</cma_stall>
        <cp>0 0.1 0</cp>
        <area>0.26</area>
        <air_density>1.240</air_density>
        <forward>1 0 0</forward>
        <upward>0 0 -1</upward>
        <link_name>blade_2</link_name>
        <control_joint>joint_22</control_joint>
    </plugin>
</gazebo>
```

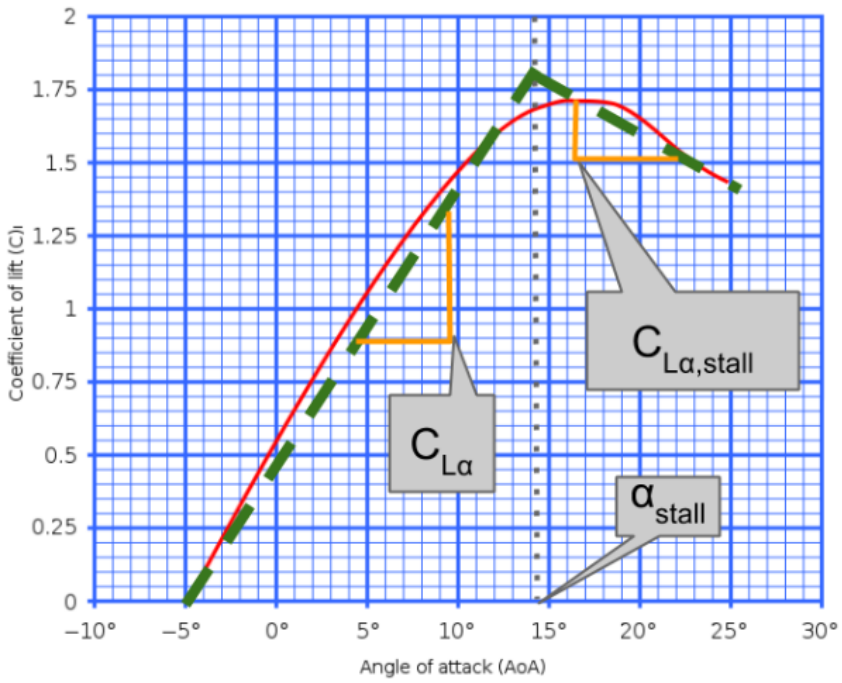
Vstupní parametry pro tento plugin jsou:

- **a0**: Počáteční „alfa“ neboli počáteční úhel náběhu [rad].
- **cla**: Poměr součinitele vztlaku a úhlu alfa před hodnotou „alfa stall“ – sklon první části křivky součinitele vztlaku.
- **cda:** Poměr součinitele odporu a úhlu alfa před hodnotou „alfa stall“.
- **alpha_stall:** Úhel náběhu, pro který dosahuje koeficient vztlaku/odporu maximální hodnoty [rad].
- **cla_stall:** Poměr součinitele vztlaku a úhlu alfa za hodnotou „alfa stall“ – sklon druhé části křivky součinitele vztlaku.
- **cda_stall:** Poměr součinitele odporu a úhlu alfa za hodnotou „alfa stall“.

- **cp:** Působiště vztlakových a odporových sil [m].
- **area:** Povrch čepele [m^2].
- **air_density:** Hustota vzduchu [$kg \cdot m^{-3}$].
- **forward:** 3D vektor reprezentující směr dopředného pohybu.
- **upward:** 3D vektor reprezentující směr vztlaku.
- **link_name:** Název linku, na který bude LiftDragPlugin uplatněn.

Vlastnosti LiftDragPluginu jsou uplatněny vždy jen na jeden link. V tomto případě kód obsahuje čtyři tyto pluginy (jako ve výpisu 14), protože kvadrokoptéra má čtyři čepele. Je důležité, aby hodnota tagu <upward> byla pro čepele 2 a 4 opačná než pro čepele 1 a 3. Tento faktor zapříčiní to, že všechny vztlakové síly budou působit v kladném směru osy z, i když se tyto dvě skupiny motorů točí opačnými směry. Graf na obrázku 18 pomůže lépe pochopit parametry LiftDragPluginu. Ve skutečnosti by hodnoty odpovídaly červené křivce, LiftDragPlugin zjednodušuje problematiku a dělá z křivky dvě lineární funkce [26].

Obr. 18: Graf závislosti součinitele vztlaku na úhlu náběhu [26]



4.2.8 Materiály

Kód ve výpisu 15 přiřadí k referenčnímu linku materiál. Ten nemá vliv na simulaci, jde pouze o vizuální stránku. Tento kód změní barvu linku „base_link“ z defaultní na šedou. Takto musí být určena barva pro každý link zvlášť.

Výpis 15: Ukázka kódu určujícího materiál

```
<!--Materials for Gazebo-->
        <gazebo reference="base_link">
            <material>Gazebo/Grey</material>
        </gazebo>
```

4.2.9 Kamera

Aby bylo možné zaznamenávat obraz nebo vidět, kam dron letí, pokud se jedná o let na delší vzdálenost, je nutné mít na konstrukci umístěnou kameru a přenášet obraz. Nejprve bude nutné si vytvořit link a připevnit jej na konstrukci.

Výpis 16: Ukázka kódu kamery (části camera_joint a camera_link)

```
<!--Camera-->
    <joint name="camera_joint" type="fixed">
        <axis xyz="0 0 1"/>
        <origin xyz="-.15 .15 .075" rpy="0 0 2.35619449"/>
        <parent link="base_link"/>
        <child link="camera_link"/>
    </joint>

    <link name="camera_link">
        <collision>
            <origin xyz="0 0 .075" rpy="0 0 0"/>
            <geometry>
                <box size=".15 .15 .15"/>
            </geometry>
        </collision>
        <visual>
            <origin xyz="0 0 .075" rpy="0 0 0"/>
            <geometry>
                <box size=".15 .15 .15"/>
            </geometry>
        </visual>
    </link>
```

Výpis 16 vytváří joint s názve „camera_joint“, který je posunutý o 0,15 metrů ve směru osy x a y. Tímto posunutím se dostane z úplného středu konstrukce blíž mezi motory 1 a 2 (směr hlavního pohybu – dopředu). Zároveň se posune o 0,075 metrů v kladném směru osy z. Takto se dostane na povrch konstrukce (výchozí bod pro joint je ve středu součásti „base_link“, která má výšku 0,015 metrů). V argumentu „rpy“ je na třetí pozici nenulová hodnota.

$$\theta = \frac{\varphi \cdot \pi}{180} = \frac{135 \cdot \pi}{180} = 2{,}35619449\ rad \tag{4.6}$$

- θ – úhel v radiánech
- φ – úhel ve stupních

Tato hodnota je uvedena v radiánech a reprezentuje natočení kolem osy z (yaw). V radiánech uvedená hodnota odpovídá 135° tzn. je natočená tak, aby zachycovala obraz mezi motory 1 a 2, což je pohyb dopředu.

Výpis 17: Ukázka přidání libgazebo_ros_camera pluginu ke camera_link

```
<gazebo reference="camera_link">
    <sensor type="camera" name="camera1">
        <update_rate>30</update_rate>
        <camera name="head">
            <horizontal_fov>1.3962634</horizontal_fov>
            <image>
                <width>800</width>
                <height>800</height>
                <format>R8G8B8</format>
            </image>
            <clip>
                <near>0.02</near>
                <far>300</far>
            </clip>
            <noise>
                <type>gaussian</type>
                <mean>0</mean>
                <stddev>0.007</stddev>
            </noise>
        </camera>
        <plugin name="camera_controller"
                filename="libgazebo_ros_camera.so" >
            <alwaysOn>true</alwaysOn>
            <updateRate>0.0</updateRate>
            <cameraName>qq/camera1</cameraName>
            <imageTopicName>image_raw</imageTopicName>
            <cameraInfoTopicName>camera_info
            </cameraInfoTopicName>
            <frameName>camera_link</frameName>
            <hackBaseline>0.07</hackBaseline>
            <distortionK1>1</distortionK1>
            <distortionK2>0.0</distortionK2>
            <distortionK3>0.0</distortionK3>
            <distortionT1>0.0</distortionT1>
            <distortionT2>0.0</distortionT2>
        </plugin>
    </sensor>
</gazebo>
```

Nejdůležitější části kódu z výpisu 17 jsou tagy <cameraName>, kde je určeno, pod jakým názvem budou topics spojené s touto kamerou. Tag <imageTopicName>, který určí jméno nejdůležitějšího topic. Další tag <cameraInfoTopicName> určí, jak se

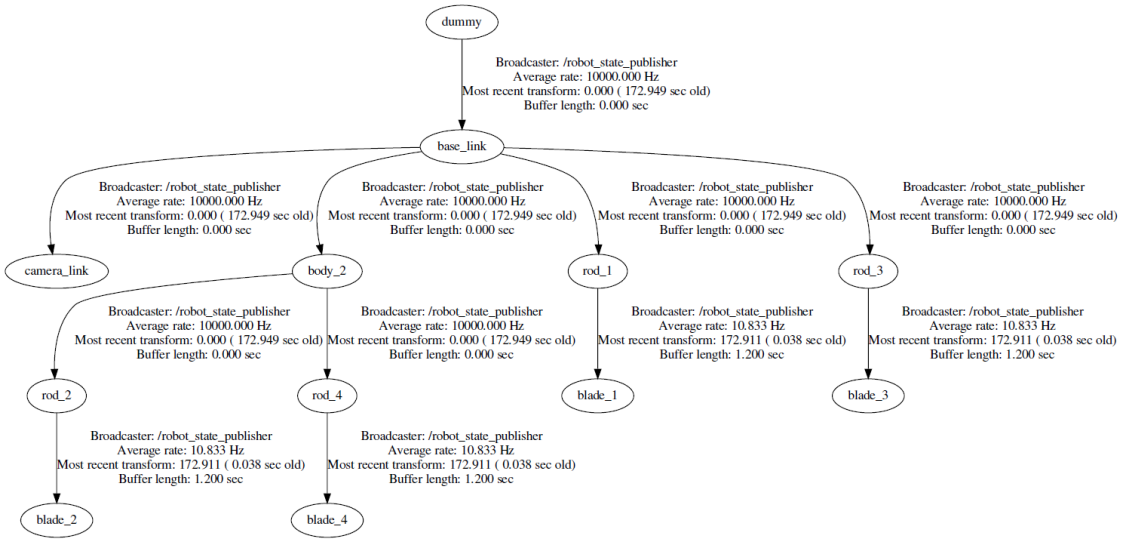
bude jmenovat topic, který poskytuje informace o kameře (jak napovídá název). Argument <frameName> musí odpovídat linku, který má reprezentovat kameru (v našem případě link „camera_link“).

Pro přenos obrazu do prostředí Rviz jako na obrázku 16 je potřeba přidat přes tlačítko „Add“ kolonku s názvem „Camera“. Jako ImageTopic musí být nastaven topic, který přenáší data s obrazem z kamery. V tomto případě je to topic, který jsme definovali v tagu <imageTopicName> pod názvem (namespace) určeným v tagu <cameraName>, tzn. „qq/camera1/image_raw“. Obraz je také možné sledovat přímo v prostředí Gazebo. V horní liště: Window > Topic Visualization. Zobrazí se seznam různých topics. Pro přenos obrazu kamery ze simulačního prostřední Gazebo musí být vybrán následující topic: „/Gazebo/default/qq/dummy/camera1/image“.

4.2.10 TF graf

TF graf (z anglického transform) přehledně ukazuje model dronu jako diagram. Na obrázku 19 jsou zakroužkované všechny součásti (linky). Šipky mezi nimi znamenají určitou vazbu (joint).

Obr. 19: TF graf



TF graf je užitečný nástroj, který může sloužit například jako kontrola modelu při debugovaní. Pro správné vykreslení je nutné spustit model (popsáno v kapitole 5) a „robot_state_publisher“ node. Soubor ve formátu PDF se uloží do aktuálního workspace po zadání:

```
rosrun tf view_frames
```

5 SIMULACE

Aby mohl být splněn poslední cíl této práce, tj. otestovat funkčnost modelu na definované úloze, musí být spuštěná simulace v Gazebu. Následně bude vytvořena úloha a napsán program, který bude řídit let kvadrokoptéru skrz překážku.

5.1 Launch

Kód ve výpisu 18 spustí model kvadrokoptéry vytvořený podle kapitoly 4. Soubor je uložen jako „qq.launch“ ve složce launch.

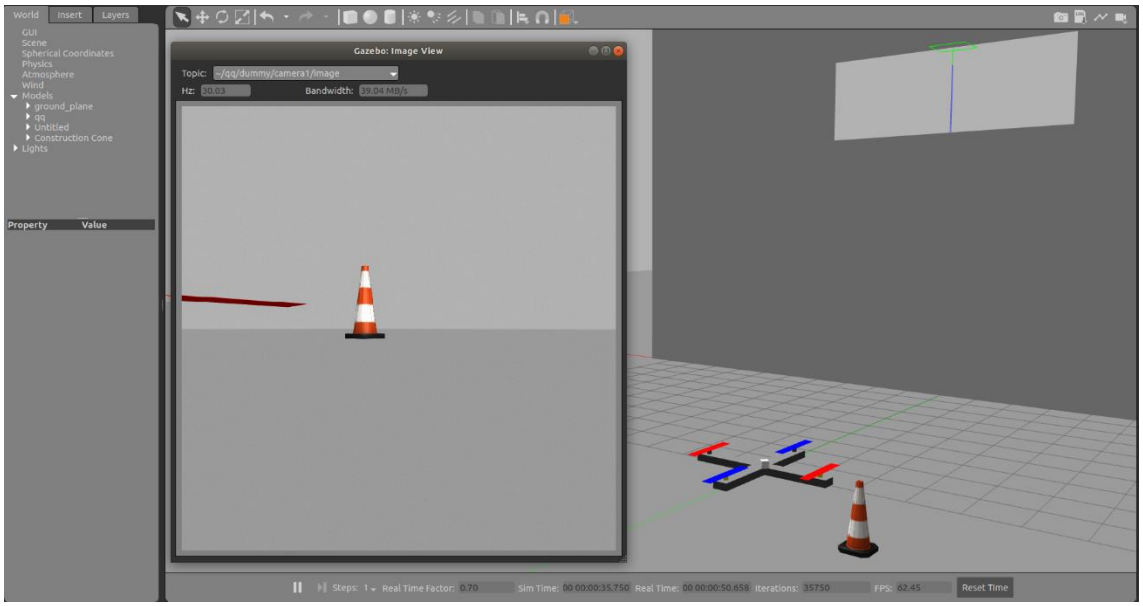
Výpis 18: Ukázka kódu spouštějícího Gazebo simulaci

```
<?xml version="1.0" encoding="UTF-8"?>
<launch>
    <!—Launch World-->
    <include file="$(find gazebo_ros)/launch/empty_world.launch">
    <arg name="verbose" value="true" />
    <arg name="paused" value="false"/>
    </include>

    <!—Spawn Model-->
    <param name="robot_description" textfile="$(find qq_des)/urdf/
                                              qq.urdf"/>
    <node  name="spawn_urdf" pkg="gazebo_ros" type="spawn_model"
    args="-file $(find qq_des)/urdf/qq.urdf -urdf -x 0 -y 0.5
          -z 0.5 -model qq" />

    <!--Load Controllers-->
    <rosparam file="$(find qq_des)/config/qq.yaml"
              command="load"/>
    <param name="robot_description"
           command="cat $(find qq_des)/urdf/qq.urdf"/>

    <node name="controller_spawner" pkg="controller_manager"
          type="spawner" respawn="false"
          output="screen" ns="/qq" args="
          joint_motor_controller
          joint_state_controller"/>
</launch>
```

První část kódu načítá prázdný svět z „gazebo_ros“ package, který je základní součástí Gazeba. V argumentech je nastaveno, aby simulace běžela rovnou po spuštění. „Verbose“ umožní získat v terminálu více informací o spouštění souboru a případných chybách. Druhá část načte model „qq.urdf“ z package „qq_des“ a umístí ho na příslušné souřadnice. Poslední část kódu aktivuje a načte ovladače pro motory pomocí souboru

„qq.yaml“ ve složce „config“. Simulace světa, do kterého byl vložen jednoduchý link jako překážka (na obrázku 20) je spuštěna příkazem:

```
roslaunch qq_des qq.launch
```



Obr. 20: Model v simulaci

5.2 Řízení

Program, díky kterému kvadrokoptéra proletí překážkou (okénko ve zdi na obrázku 20) je napsán v jazyku Python (výpis 19). Kód se nachází ve složce „src“ pod názvem „m.py“.

Výpis 19: Ukázka kódu definujícího prvky ovládání

```
import rospy
import time
from std_msgs.msg import Float64MultiArray
rospy.init_node('topic_publisher')
pub = rospy.Publisher('/qq/joint_motor_controller/command',
Float64MultiArray, queue_size=1)
rate = rospy.Rate(1)
count = Float64MultiArray()
def takeoff(x):
  sec = 0
  for i in range (x):
    sec += 1
    time.sleep(1)
    i = 800
    count.data = [-i, i, -i, i]
    pub.publish(count)
```

Nejprve byly naimportovány tzv. moduly. Modul „rospy“ pro interakci Pythonu s ROS. „Float64MultiArray“ z modulu „std_msgs.msg“ slouží k posílání příkazů na všechny motory dronu zároveň. Nakonec byl přidán modul „time“, protože řídící funkce potřebují vstupní argument (typu integer), který určí, kolik sekund bude určitá funkce běžet.

Ve druhé části byl spuštěn node „topic_publisher“. Metodou publisher bylo nastaveno posílání message typu „Float64MultiArray“ na topic „/qq/joint_motor_controller/command“.

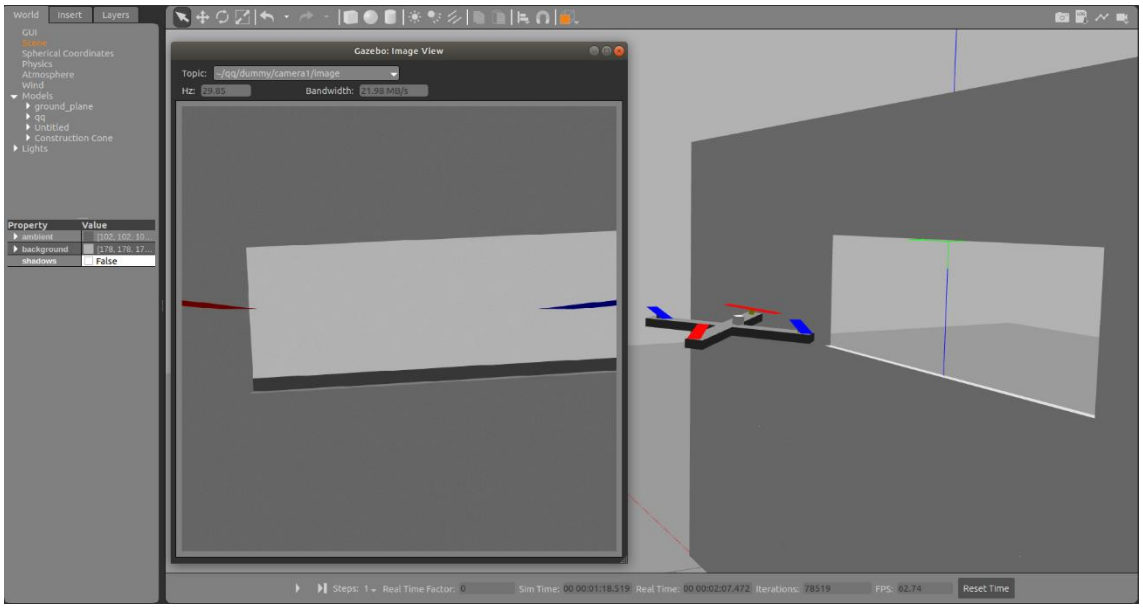
Nakonec byla definována funkce „takeoff“. Do této funkce vložíme hodnotu, která určí, kolik sekund bude funkce aktivní. Díky „time.sleep(1)“ trvá každá jedna iterace smyčky „for“ jednu sekundu. Takto je nadefinováno několik dalších funkcí, aby bylo obsaženo kompletní ovládání, které je teoreticky popsáno v kapitole 2.

Na základě těchto nadefinovaných funkcí byl vytvořen kód (výpis 20), který řídí kvadrokoptéru skrz překážku (obrázek 21 a 22).

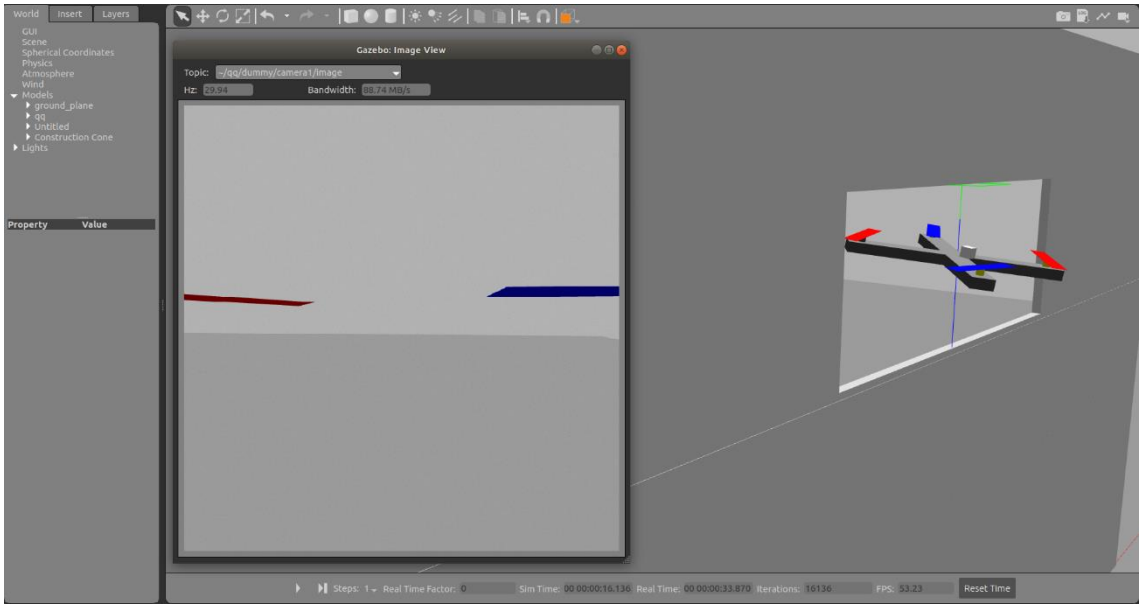
Výpis 20: Ukázka kódu řídícího kvadrokoptéru

```
takeoff(3)
up(3)
hover(2)
yaw_right(7)
forward(1)
hover(3)
turn_right(1)
backwards(1)
turn_left(1)
forward(1)
hover(1)
backwards(1)
hover(5)
land(3)
backwards(1)
turn_right(1)
forward(1)
turn_left(1)
land(2)
hover(2)
land(2)
off(1)
```

Tento program kombinuje několik prvků ovládání v různých časových intervalech, aby model letěl stabilně. Jedná se však o vzlet, natočení modelu kamerou směrem k překážce, průlet otvorem a přistání. Celý tento proces je zachycen na videu v příloze.



Obr. 21: Kvadrokoptéra před překážkou



Obr. 22: Kvadrokoptéra prolétající překážkou

6 ZÁVĚR

Cílem této bakalářské práce bylo sepsat stručnou rešerši o dané problematice, vytvořit model kvadrokoptéry a otestovat jeho funkčnost na dané úloze. Všechny z těchto bodů se podařilo splnit. Jsou v této práci podrobně popsány v rešeršní a praktické části.

První část se zabývá dynamikou pohybu kvadrokoptéry, kde byl popsán základní princip letu, rovnice pro výpočet vztlakové síly a prvky ovlivňující let. Dále vysvětluje framework ROS, jeho základní prvky, principy fungování, důležité příkazy pro interakci uživatele s ROS a operačním systémem Linux.

Druhá část obsahuje a podrobně popisuje výpisy z jednotlivých částí kódu modelu jako je tělo konstrukce, vrtule, kamera, ovladače pro motory, nejrůznější pluginy pro ovládání motorů, generování vztlaku pomocí čepelí, přenos obrazu ze simulačního prostředí. Je zde rozebráno, jak vypadá program pro řízení kvadrokoptéry, které vrtule se točí pro vykonání jednotlivých akcí a jakým způsobem. Dále komentuje soubor launch, který simuluje svět v Gazebu a vytvořený model. V tomto světě byla vytvořena překážka, kterou měl dron proletět a hladce přistát. Je zde také zdokumentován průběh simulace, která je celá nahrána na videu v příloze.

Práce bude sloužit jako základ pro nadcházející diplomový projekt. Z důvodu absence českých zdrojů na toto téma by mohla být také opěrným materiálem nejen pro samouky snažící se naučit ROS, vytvořit a simulovat svůj vlastní robot.

7 SEZNAM POUŽITÉ LITERATURY

[1] TILGNER, M. Řídící deska pro kvadrokoptéru. Brno: Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií, 2017. Vedoucí bakalářské práce Ing. František Burian, Ph.D.

[2] RAKOVSKÝ, K. Vztlaková mechanizace křídla. Brno: Vysoké učení technické v Brně, Fakulta strojního inženýrství, 2013. Vedúci bakalárskej práce Ing. Marek Horák.

[3] BRADÁČ, František. Quadrokopter – stabilizace pomocí inerciálních snímačů. Brno, 2009. Diplomová práce. Vysoké učení technické v Brně, Fakulta elektrotechniky a komunikačních technologií. Vedoucí práce Doc. Ing. Luděk Žalud, Ph.D.

[4] UHLÍŘ, Adam. Konstrukce a řízení kvadrokoptéry. Pardubice: Univerzita Pardubice, Fakulta elektrotechniky a informatiky, 2013. Vedoucí bakalářské práce Ing. Libor Havlíček, Ph.D.

[5] GOPALAKRISHNAN, Eswarmurthi. QUADCOPTER FLIGHT MECHANICS MODEL AND CONTROL ALGORITHMS. Praha, 2017. Master thesis. Czech Technical University, Faculty of Electrical Engineering. Luleå University of Technology, Department of Control Engineering. Supervisor: Prof. Dr. Martin Hromčík.

[6] RESNICK, Robert; WALKER, Jearl; HALLIDAY, David. *Fundamentals of physics*. Hoboken: John Wiley, 1988.

[7] Robot Operating System - Wikipedia. [online]. [cit 2020-05-23] Dostupné z: https://en.wikipedia.org/wiki/Robot_Operating_System

[8] DOČKAL, Ondřej. Využití frameworku Tapestry pro vývoj moderních webových aplikací Utilization of Tapestry framework for modern web application development. Ostrava: VŠB-Technická univerzita Ostrava, Fakulta elektrotechniky a informatiky, 2010. Vedoucí diplomové práce Ing. Filipec

[9] Middleware. IT Slovník [online]. [cit 2020-05-23] Dostupné z: https://www.it-slovnik.cz/pojem/middleware

[10] ROS/Introduction - *ROS Wiki. Documentation* - ROS Wiki [online]. [cit. 2020-06-02] Dostupné z: https://wiki.ros.org/ROS/Introduction

[11] O'KANE, Jason M. A gentle introduction to ROS. University of South Carolina, Department of Computer Science and Engineering, 2014.

[12] rosbash - ROS Wiki. *Documentation - ROS Wiki* [online]. [cit 2020-06-03] Dostupné z: https://wiki.ros.org/rosbash

[13] Creating a catkin workspace. *Packt Subscription | Learn more for less* [online]. [cit 2020-06-05] Dostupné z: https://subscription.packtpub.com/book/hardware_and_creative/9781782175193/1/ch01lvl1sec11/creating-a-catkin-workspace

[14] Nodes - ROS Wiki. *Documentation - ROS Wiki* [online]. [cit 2020-06-05] Dostupné z: https://wiki.ros.org/Nodes

[15] Topics - ROS Wiki. *Documentation - ROS Wiki* [online]. [cit 2020-06-06] Dostupné z: https://wiki.ros.org/Topics

[16] Intro to ROS. *Clearpath robotics* [online]. [cit 2020-06-06] Dostupné z: http://www.clearpathrobotics.com/assets/guides/kinetic/ros/Intro%20to%20the%20Robot%20Operating%20System.html#general-concepts

[17] Gazebo: Tutorial: Beginner: Overview. Gazebo [online]. Open Source Robotics Foundation [cit. 19.06.2020]. Dostupné z: http://gazebosim.org/tutorials?tut=guided_b1&cat=

[18] Slovník základních pojmů | Svět hardware. *Svět hardware* [online]. [cit. 19-06-2020]. Dostupné z: https://www.svethardware.cz/slovnik/

[19] Gazebo: Tutorial: Import Meshes. Gazebo [online]. Open Source Robotics Foundation [cit. 19.06.2020]. Dostupné z: http://gazebosim.org/tutorials/?tut=import_mesh

[20] Rviz - ROS Wiki. *Documentation - ROS Wiki* [online]. Dostupné z: https://wiki.ros.org/action/fullsearch/rviz?action=fullsearch&context=180&value=linkto%3A%22rviz%22

[21] urdf/XML/link - ROS Wiki. *Documentation - ROS Wiki* [online]. [cit 21-06-2020] Dostupné z: https://wiki.ros.org/urdf/XML/link

[22] urdf/XML/joint - ROS Wiki. *Documentation - ROS Wiki* [online]. [cit 21-06-2020] Dostupné z: https://wiki.ros.org/urdf/XML/joint

[23] urdf/XML/Transmission - ROS Wiki. *Documentation - ROS Wiki* [online]. [cit 22-06-2020] Dostupné z: https://wiki.ros.org/urdf/XML/Transmission

[24] ros_control - Documentation - ROS Wiki [online]. [cit 22-06-2020] Dostupné z: https://wiki.ros.org/ros_control#Hardware_Interfaces

[25] Gazebo tutorial: ROS control. *Gazebo* [online].Open Source Robotics Foundation [cit. 22.06.2020]. Dostupné z: http://gazebosim.org/tutorials/?tut=ros_control

[26] Gazebo Tutorial: Aerodynamics. *Gazebo* [online]. Open Source Robotics Foundation [cit. 22.06.2020]. Dostupné z: http://gazebosim.org/tutorials?tut=aerodynamics&cat=physics

8 SEZNAM ZKRATEK A SYMBOLŮ

- F_l [N] – vztlaková síla
- ρ [kg·m^{-3}] – hustota vzduchu
- v [m·s^{-1}] – rychlost
- S [m^2] – povrch listů čepele
- c_L [-] – součinitel vztlaku
- G [N] – tíhová síla
- m [kg] – hmotnost kvadrokoptéry
- g [m·s^{-2}] – tíhové zrychlení (cca. 9,81 m/s^{-2})
- M [N·m] – moment síly
- l [m] – vzdálenost působiště vztlakové síly motoru od těžiště modelu
- I [kg·m^2] – moment setrvačnosti
- ω [rad·s^{-1}] – úhlová rychlost
- L [kg·m^2·s^{-1}] – moment setrvačnosti
- a [m·s^{-2}] – zrychlení
- I_x, I_y, I_z – momenty hybnosti k osám x, y, z
- m_k – hmotnost kvádru
- x, y, z – rozměry kvádru v příslušných osách
- m_v – hmotnost válce
- r – poloměr válce
- h - výška válce
- θ – úhel v radiánech
- φ – úhel ve stupních
- Debugování: odstraňování chyb v kódu
- Tzv.: takzvaně
- Tzn.: to znamená
- Defaultní: výchozí
- Tj.: to je
- Např.: například

9 SEZNAM PŘÍLOH

Kód
Video

PŘÍLOHY