Integrace bezdrátového gyroskopického ovladače do reálného systému řízení pohybu s grafickou vizualizací snímaných veličin

Patrik Braborec

Bakalářská práce
2020

Univerzita Tomáše Bati ve Zlíně
Fakulta aplikované informatiky

Univerzita Tomáše Bati ve Zlíně

Fakulta aplikované informatiky

Ústav automatizace a řídicí techniky

Akademický rok: 2019/2020

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Patrik Braborec**
Osobní číslo:	**A17209**
Studijní program:	**B3902 Inženýrská informatika**
Studijní obor:	**Inteligentní systémy s roboty**
Forma studia:	**Kombinovaná**
Téma práce:	**Integrace bezdrátového gyroskopického ovladače do reálného systému řízení pohybu s grafickou vizualizací snímaných veličin**
Téma práce anglicky:	**Wireless Gyro Controller Integration into a Real Motion Control System with Graphical Visualisations of Scanned Variables**

Zásady pro vypracování

1. Vypracujte teoretický základ ke gyroskopickému ovládání.
2. Navrhněte vhodný způsob integrace tohoto typu ovládání do reálného modelu „Kulička na nakloněné rovině" s ohledem na použitý hardware/software.
3. Přidejte volbu generování žádaných trajektorií z gyroskopického ovladače a vše odlaďte na výše uvedeném reálném modelu.
4. Zvolte vhodnou formu záznamu a grafické interpretace snímaných veličin s ohledem na použitý Framework a IDE.
5. Implementujte vybrané řešení do existujícího řídicího softwaru.
6. Kriticky zhodnoťte použité metody a výsledky práce, případně navrhněte odpovídající alternativy.

Rozsah bakalářské práce:
Rozsah příloh:
Forma zpracování bakalářské práce: **tištěná/elektronická**

Seznam doporučené literatury:

1. Fraden, J. Handbook of Modern Sensors: Physics, Designs, and Applications, 4th ed.; Springer-Verlag: New York, 2010; pp. 663.
2. Zhi Eng, L. Qt5 C++ GUI Programming Cookbook, 2nd ed.; Packt Publishing: Birmingham, England, 2019; pp. 428.
3. Tavasalkar, D. Hands-On Robotics Programming with C++, 1st ed.; Packt Publishing: Birmingham, UK, 2019; pp. 312.
4. Prata, S. Mistrovství v C++, 4th ed.; Computer Press, Albatros Media a.s.: 2013; pp. 1176.
5. Halfacree, G. The Official Raspberry Pi Beginner's Guide, 1st ed.; Raspberry Pi Press: Cambridge, UK, 2018; pp. 241.

Vedoucí bakalářské práce: **Ing. Jiří Zátopek**
Ústav automatizace a řídicí techniky

Datum zadání bakalářské práce: **20. prosince 2019**
Termín odevzdání bakalářské práce: **15. května 2020**

doc. Mgr. Milan Adámek, Ph.D.
děkan

prof. Ing. Vladimír Vašek, CSc.
ředitel ústavu

Ve Zlíně dne 21. prosince 2018

Prohlašuji, že

- beru na vědomí, že odevzdáním bakalářské práce souhlasím se zveřejněním své práce podle zákona č. 111/1998 Sb. o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších právních předpisů, bez ohledu na výsledek obhajoby;
- beru na vědomí, že bakalářská práce bude uložena v elektronické podobě v univerzitním informačním systému dostupná k prezenčnímu nahlédnutí, že jeden výtisk diplomové/bakalářské práce bude uložen v příruční knihovně Fakulty aplikované informatiky Univerzity Tomáše Bati ve Zlíně a jeden výtisk bude uložen u vedoucího práce;
- byl/a jsem seznámen/a s tím, že na moji bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb. o právu autorském, o právech souvisejících s právem autorským a o změně některých zákonů (autorský zákon) ve znění pozdějších právních předpisů, zejm. § 35 odst. 3;
- beru na vědomí, že podle § 60 odst. 1 autorského zákona má UTB ve Zlíně právo na uzavření licenční smlouvy o užití školního díla v rozsahu § 12 odst. 4 autorského zákona;
- beru na vědomí, že podle § 60 odst. 2 a 3 autorského zákona mohu užít své dílo – diplomovou/bakalářskou práci nebo poskytnout licenci k jejímu využití jen připouští-li tak licenční smlouva uzavřená mezi mnou a Univerzitou Tomáše Bati ve Zlíně s tím, že vyrovnání případného přiměřeného příspěvku na úhradu nákladů, které byly Univerzitou Tomáše Bati ve Zlíně na vytvoření díla vynaloženy (až do jejich skutečné výše) bude rovněž předmětem této licenční smlouvy;
- beru na vědomí, že pokud bylo k vypracování bakalářské práce využito softwaru poskytnutého Univerzitou Tomáše Bati ve Zlíně nebo jinými subjekty pouze ke studijním a výzkumným účelům (tedy pouze k nekomerčnímu využití), nelze výsledky bakalářské práce využít ke komerčním účelům;
- beru na vědomí, že pokud je výstupem bakalářské práce jakýkoliv softwarový produkt, považují se za součást práce rovněž i zdrojové kódy, popř. soubory, ze kterých se projekt skládá. Neodevzdání této součásti může být důvodem k neobhájení práce.

Prohlašuji,

- že jsem na bakalářské práci pracoval samostatně a použitou literaturu jsem citoval. V případě publikace výsledků budu uveden jako spoluautor.
- že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

Ve Zlíně, dne …….

podpis diplomanta

ABSTRAKT

Cílem této bakalářské práce je integrovat bezdrátový gyroskopický ovladač do reálného modelu „Kulička na nakloněné rovině“ s ohledem na použitý hardware/software a graficky interpretovat průběhy snímaných veličin v reálném čase. Gyroskopický ovladač bude použitý pro vzdálené ovládání modelu s možností generování žádané trajektorie jak přímo řízených (akční členy), tak nepřímo řízených (pozice kuličky na nakloněné rovině) parametrů. K bezdrátovému gyroskopickému ovládání bude vypracován teoretický základ. Dále bude práce zahrnovat výběr vhodné formy záznamu a grafické interpretace snímaných veličin s ohledem na použitý Framework a IDE. Vybrané řešení bude implementováno do existujícího řídicího softwaru výše zmíněného reálného modelu, na kterém bude probíhat ladění. Použité metody a výsledky práce budou kriticky zhodnoceny, případně budou navrženy odpovídající alternativy.

Klíčová slova: Gyroskopické ovládání, grafická vizualizace, řízení pohybu, Raspberry Pi, C++, Qt Framework

ABSTRACT

The goals of bachelor thesis is to integrate wireless gyroscopic controller to the real model „Ball on the inclined plane“ with respect to used hardware/software and to graphically display scanned values in real time. Gyroscopic controller will be used for remote controlling of the model with a possibility to to generate desired trajectory both on directly controlled (acutators) and on indirectly controlled (position of the ball on inclined plane) parameters. Theoretical base will be developed for the wireless gyroscopic control. Furthermore, the thesis will include choice of appropriate form of record and graphical interpretation of scanned values with respect to used framework and IDE. Selected solution shall be implemented into the existing control software of the real model mentioned above, on which debugging will take place. Used methods and results of this work shall be evaluated critically and eventually, appropriate alternatives will be proposed.

Keywords: Gyro Controller, Graphic Visualization, Motion Control, Raspberry Pi, C++, Qt Framework

Rád bych poděkoval vedoucímu bakalářské práce Ing. Jiřímu Zátopkovi za vedení, pomoc a rady během tvorby této práce.

Prohlašuji, že odevzdaná verze bakalářské práce a verze elektronická nahraná do IS/STAG jsou totožné.

OBSAH

ÚVOD

Cílem bakalářské práce je integrace bezdrátového gyroskopického ovladače do reálného modelu "Kulička na nakloněné rovině" s ohledem na použitý hardware a software a graficky interpretovat průběhy snímaných veličin v reálném čase.

Důvodem integrace bezdrátového gyroskopického ovladače je možnost prozkoumat tento způsob ovládání na reálném modelu vyrobeném pro akademické účely a kriticky zhodnotit, zda by tento typ ovládání robotů mohl mít využití v průmyslu anebo v jiném odvětví.

Důvod pro záznam veličin a jejich grafickou interpretaci vychází z potřeby zkoumání jevů, které probíhají uvnitř reálné modelu "Kulička na nakloněné rovině" během provozu. Díky záznamu a grafické interpretaci je například možné odhalovat chyby v reálném čase, případně díky záznamu je možné veličiny zkoumat zpětně a hledat důvody, proč k chybám došlo. Další oblastí zájmu u záznamu veličin a následném exportu je analýza exportovaných veličin v jiných softwarových nástrojích určené k tomuto účelu.

I. TEORETICKÁ ČÁST

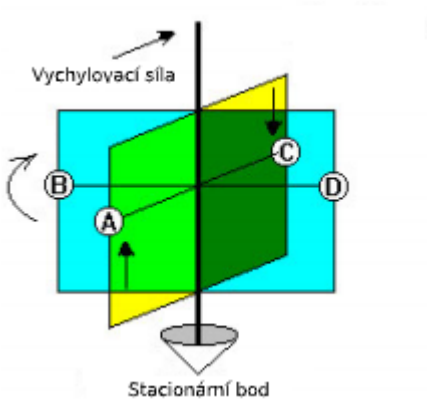
1 GYROSKOPICKÉ OVLÁDÁNÍ

Gyroskopické ovládání má v sobě dva základní senzory, a to je gyroskop a snímač zrychlení neboli akcelerometr. V úvodu kapitoly je vysvětlen princip mechanického gyroskopu pro správné pochopení, co vlastně gyroskop je a následně je popsána a vysvětlena funkce elektronického gyroskopu a akcelerometru ve formě MEMS. V závěru je vysvětlen přepočet informací z gyroskopu a využití gyroskopického ovladače v praxi.

1.1 Mechanický gyroskop

Gyroskop využívá první Newtonův zákon. V jakémkoliv systému částic, totální moment hybnosti systému relativní k jakémukoliv bodu zafixovaném v prostoru, zůstává konstantní, pokud na něj nepůsobí žádné vnější síly. Celý princip gyroskopu je možné teoreticky demonstrovat a jednoduše vysvětlit na mechanickém zařízení, které se skládá z otáčející se osy a čtyř bodů *A, B, C, D*. Toto zařízení je zobrazeno na Obrázek 1. [1]

Spodní část osy je držena stacionárně, ale je možné s ní pohybovat všemi směry. V případě působení vychylovací síly v horní části osy, se bod *A* začne pohybovat směrem nahoru a bod *C* směrem dolů. [2]



Obrázek 1. Znázornění působení síly [2]

Když se gyroskop při otáčení po směru hodinových ručiček otočí o 90 stupňů a bod *A* se dostane tam, kde byl bod *B* na začátku působení vychylovací síly (stejná situace nastane i s body *C* a *D*). Bod *A* svým působení stále působí směrem nahoru a bod *C* směřuje stále směrem dolů. [2]

Kombinace pohybů bodů *A, C* má za následek to, že se osa v tzv. precesní rovině bude pohybovat ve směru podle Obrázek 1 – tento jev se nazývá precese. Při otočení gyroskopu o dalších 90 stupňů se bod *C* dostane tam, kde byl na začátku působení vychylovací síly bod

A, a protože směr pohybu bodu *C* dolů je nyní potlačen vychylovací silou, tak se osa nepohybuje ve směru působení vychylovací síly. To má za následek to, že při působení vychylovací síly, bude bod na opačném konci působení síly posunovat osu zpět po otočení o 180 stupňů. Jev precese lze potom využít pro snímání natočení vůči určenému referenčnímu bodu a následně určit i úhel natočení. [2]

1.2 MEMS senzory

MEMS je zkratka pro mikro-elektro-mechanický systém a jedná se o senzory, které jsou charakteristické svou malou velikostí a způsobem, jakým jsou vyrobeny. Jejich velikost se nejčastěji pohybuje mezi 1 až 100 mikrometry a obecně sestávají z mechanické struktury a elektroniky v jednom provedení. [1]

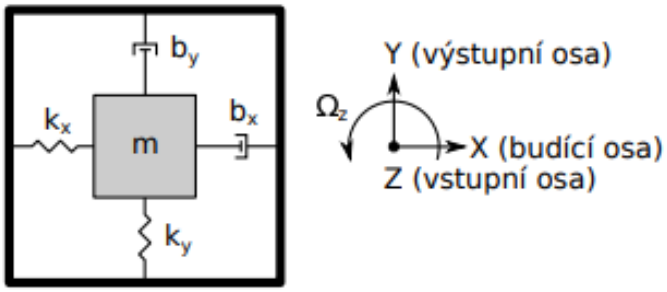
1.3 MEMS gyroskop

Díky MEMS technologii je možné použití miniaturních gyroskopů, kde je mechanická část, tedy zejména rotující disk nahrazen vibračním mechanickým elementem pro zjištění rotačního pohybu. Vibrační gyroskopy spoléhají na jev zvaný Coriolisova síla, což je jev popsaný francouzským inženýrem a matematikem G. G. De Coriolis. [1][3].

Coriolisova síla je důsledek aplikace Newtonových pohybových zákonů na rotující hmotná tělesa. Lze ji definovat jako setrvačnou hmotnost *m*, která se lineárně pohybuje rychlostí *v* a současně koná rotační pohyb úhlovou rychlostí **Ω**. Tento vztah popisuje rovnice (1). [3][4]

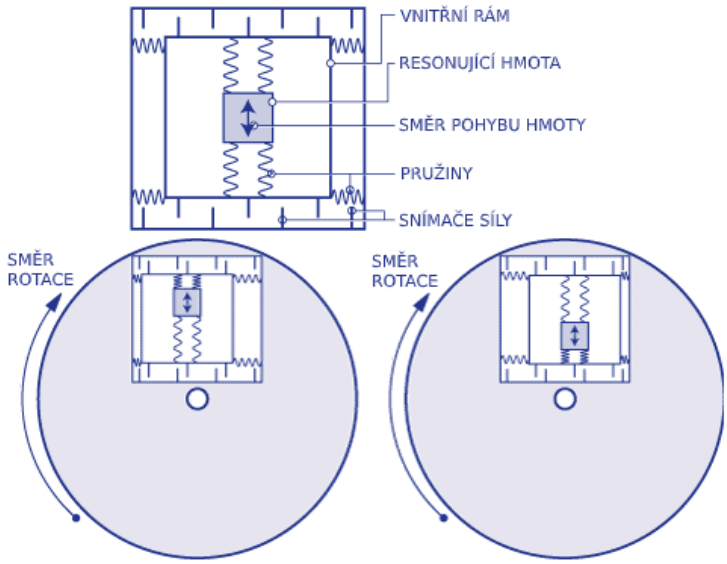
$$F_C = -2 \cdot m \cdot (\Omega \times v) \qquad (1)$$

Princip Coriolisovy síly je možné vysvětlit na mechanickém rezonátoru se dvěma stupni volnosti, který je znázorněn na Obrázek 2. Rezonátor se dvěma stupni volnosti [4]. Aby rezonátor v ose *X* vykonával lineární harmonický pohyb je buzen silou F_x. Pokud budou na mechanický rezonátor působit účinky úhlové rychlosti Ω_z, které budou působit v ose *Z*, což je vstupní osa, potom podle rovnice (1) vznikne síla, která bude působit v ose *Y*, což je výstupní osa. Mezi budící a výstupní osou potom existuje vazba, což zapříčiní, že se pohyb z budící osy přenáší do výstupní osy. [4]



Obrázek 2. Rezonátor se dvěma stupni volnosti [4]

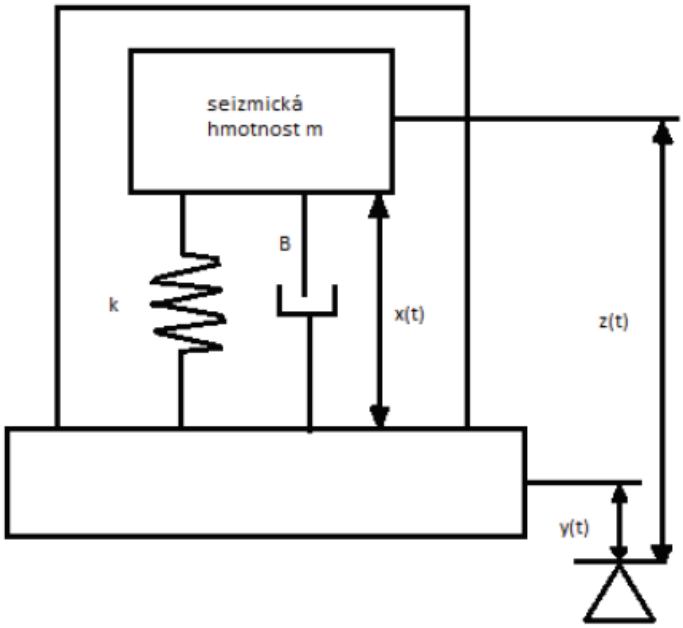
Jak již bylo zmíněno, v praxi se používají miniaturní gyroskopy s využitím technologie MEMS. Jeho funkci je možné ukázat na Obrázek 3. Na čipu je mechanický rezonátor s přesnou hmotností upevněn pomocí pružin v rámu a spolu s elektrickými obvody tvoří celý snímač. Základním předpokladem je, že směr pohybu mechanického rezonátoru je vždy kolmý ke směru otáčení. Na mechanický rezonátor pak působí Coriolisova síla, jejíž velikost je úměrná úhlové rychlosti otáčení a působení této síly stlačí vnějších pružiny, přičemž dojde k posuvu měřících plošek, které fungují jako elektrody kondenzátorů. Výstupem snímače je pak změna kapacity, která je právě úměrná úhlové rychlosti otáčení. [5]



Obrázek 3. Struktura MEMS gyroskopu

1.4 MEMS akcelerometr

Na Obrázek 4. Struktura akcelerometru je zobrazena základní struktura akcelerometru, která je společná jak u mechanických akcelerometrů, tak i u elektronických akcelerometrů. Akcelerometr je spojen s neinerciální soustavou, což je těleso nebo objekt u kterého měříme zrychlení. Vnější část je spojena pružinou k a tlumičem B se seismickou hmotou m. [6]



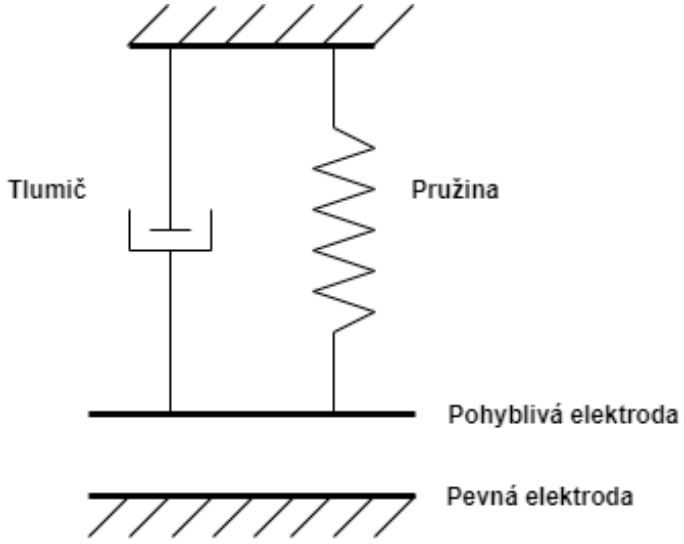
Obrázek 4. Struktura akcelerometru [6]

Výše popsanou strukturu akcelerometru lze popsat následujícími rovnicemi (2,3), kde *m* je rovno velikosti seizmické hmotnosti, *k* vyjadřuje tuhost pružiny a *b* je koeficient tlumení. [6]

$$z(t) = x(t) + y(t) \tag{2}$$

$$m \cdot \frac{d^2z}{dt^2} + b \cdot \frac{dx}{dt} + kx = 0 \tag{3}$$

MEMS technologie dovoluje výše popsanou strukturu integrovat do kapacitního akcelerometru, který je zobrazen na Obrázek 5. Struktura kapacitního akcelerometru. U MEMS akcelerometru je na jednom čipu mechanická struktura akcelerometru a vyhodnocovací a výstupní obvody. [6]



Obrázek 5. Struktura kapacitního akcelerometru [6]

Kapacitní akcelerometr je složen ze dvou desek. Jedna z desek kapacitního akcelerometru je spojena s měrným objektem a druhá se seismickou hmotností. Obě pak tvoří jednu elektrodu kondenzátoru, kde se mění velikost mezery mezi elektrodami a tím i kapacita kondenzátoru. Pokud měrný objekt zrychluje nebo zpomaluje, způsobí to změnu vzdálenosti desek a tím pádem i změnu kapacity kondenzátoru. Výstupem MEMS akcelerometru je pak napěťový signál a ten je úměrný zrychlení objektu. [6]

1.5 Rozdíl mezi gyroskopem a akcelerometrem

Gyroskop i akcelerometr využívají podobný pohyblivý rám se snímacími ploškami, které se chovají jako plošky kondenzátoru, ale z pohledu principu existuje jeden významný rozdíl. U gyroskopu je s pohyblivou strukturou periodicky pohybováno, aby se projevil efekt Coriolisovy síly, ale u akcelerometru je tato pohyblivá struktura volná a pohybuje se jen při působení síly. Z toho lze vyvodit, že gyroskop je určen pro zjištění rychlosti ustáleného rotačního pohybu, ale akcelerometr měří hodnotu vychýlení pohyblivé struktury, což je důsledkem působícího zrychlení. [5]

1.6 Přepočet dat z gyroskopického ovladače

Jelikož není možné se dostat přímo k datům z gyroskopu, nebo akcelerometru, ale gyroskopický ovladač posílá už přepočtená data, jež jsou výslednou pozicí kurzoru na obrazovce, je zde teoreticky rozebrán algoritmus pro tento přepočet. Pro výpočet pozice kurzoru musí proběhnout určitý přepočet z dat akcelerometru a gyroskopu. Data z akcelerometru je možné označit podle os X_a, Y_a a data z gyroskopu jako X_g, Y_g. Úhel

natočení získaný z dat akcelerometru je možné vyjádřit okolo *X* jako (4) a úhel natočení okolo *Y* jako (5). [7]

$$accX \approx \tan^{-1}\left(\frac{Y_a}{\sqrt{{X_a}^2+{Z_a}^2}}\right) \tag{4}$$

$$accY \approx \tan^{-1}\left(\frac{X_a}{\sqrt{{Y_a}^2+{Z_a}^2}}\right) \tag{5}$$

Akcelerometr je velmi citlivý na vibrace a mechanický šum. Data z akcelerometru je tedy nutné kombinovat s daty z gyroskopu, protože gyroskop není tolik citlivý na mechanický šum a vibrace. Vyjádření úhlu natočení z dat gyroskopu je možné vyjádřit jako (6,7), kde *k* je konstanta, která udává míru důvěry k datům z gyroskopu ve srovnání s daty z akcelerometru a volí se experimentálně podle konkrétního gyroskopu a akcelerometru. [7]

$$gyroX \approx \frac{X_g}{k} \tag{6}$$

$$gyroY \approx \frac{Y_g}{k} \tag{7}$$

Obě hodnoty se pak aplikují v tzv. Kalmanově filtru (8,9), který slouží k odhadu odchylky naměřené hodnoty od hodnoty, která pravděpodobně odpovídá skutečnému stavu. [7]

$$kalX = (accX, gyroX) \tag{8}$$

$$kalY = (accY, gyroY) \tag{9}$$

Sekvence dat o úhlu natočení obou senzorů a jejich kombinace, následná filtrace pomocí Kalmanova filtru zajistí znalost o pohybu ve směru *X* a ve směru *Y*, které je možné následně přepočíst podle rozlišení obrazovky na výslednou pozici kurzoru. [7]

1.7 Využití gyroskopického ovládání v praxi

Gyroskopické ovládání má široké využití v herním průmyslu, kde takový ovladač lze nalézt u herních konzolí Xbox nebo PlayStation. Mimo odvětví herního průmyslu je možné nalézt využití v robotice, například u tzv. pohybové rukavice pomocí které je možné ovládat různé části robota. Zejména s rozvojem robotiky v průmyslu by taková pohybová rukavice, případně jiné druhy gyroskopických ovládání v propojení s virtuální realitou mohly najít široké uplatnění v místech, kde je přítomnost člověka životu nebezpečná, nebo z nějakých jiných důvodů nemožná. Mimo herní průmysl a robotiku je možné gyroskopické ovládání

taky nalézt jako speciální typ ovladače pro televize. S nástupem tzv. Smart TV je tento typ ovladače velmi přínosný, protože díky jeho pomoci je možné televizi velmi jednoduše ovládat.

2 POPIS REÁLNÉHO MODELU

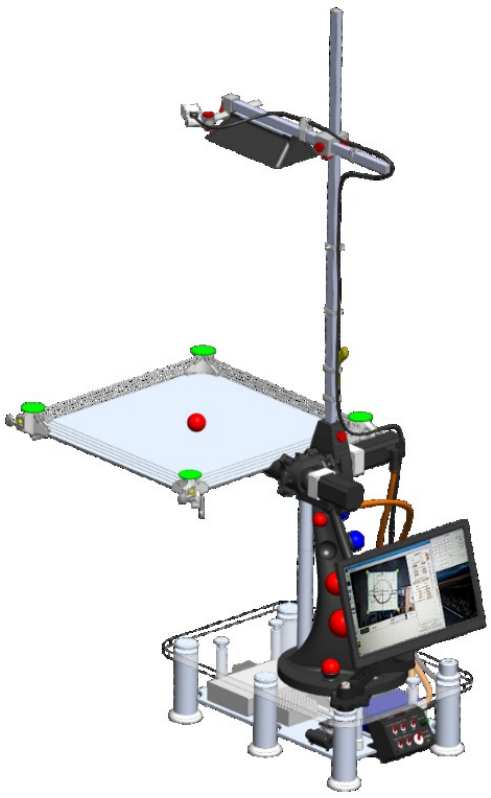
V této kapitole je rozebráno, jak vypadá reálný model „Kulička na nakloněné rovině“ co se týká hardwarové, tak i softwarové stránky. Postupně je rozebrána robotická struktura a jednotlivé hardwarové i softwarové technologie.

2.1 Popis hardwaru

V úvodu kapitoly je popsána obecně celá robotická struktura reálného modelu a následně jsou popsány klíčové komponenty tohoto modelu.

2.1.1 Robot s nakloněnou rovinou

Jedná se o sériovou robotickou strukturu, která má čtyři stupně volnosti, z čehož dva jsou řiditelné. Robotická struktura se skládá z nakloněné roviny, která je řízena dvěma elektromotory, jež jsou uloženy sériově. Objektem řízení je pak kulička, která se po rovině odvaluje a tím přidává modelu další dva nepřímo řízené zobecněné stupně volnosti. Téměř celý reálný model je vyroben za pomoci technologie 3D tisku, jen některé části jsou vyrobeny za pomoci jiných technologií, například na CNC frézce, kvůli požadavkům na tvrdost materiálu. [8]



Obrázek 6. Robot s nakloněnou rovinou [8]

2.1.2 Servomotor

Jako akční členy reálného modelu, tedy hlavní část pohybového systému, jsou zvoleny servomotory. Servomotor je elektromotor s řídicím systémem – je to tedy systém v němž se při řízení pohybu současně přeměňuje elektrická energie na mechanickou práci. Mezi nejpoužívanější servomotory patří trojfázový synchronní motor s permanentním magnetem v rotoru (PMSM). Jsou využívány zejména kvůli přesnému polohovému řízení a díky řídící jednotce jsou dobrou volbou u systému, kde není potřeba řešit vlastní návrh výkonové elektroniky. [8]

Obrázek 7. Servomotory [8]

2.1.3 Raspberry Pi

Jako řídící počítač reálného modelu se využívá Raspberry Pi. Je to plně funkční počítač v malém provedení. Je vhodný pro prohlížení webových stránek, hraní her anebo vytváření vlastních programů. Motivací pro vývoj tohoto počítače byla podpora informatiky ve školách a navrhla jej britská nadace Raspberry Pi Foundation v roce 2012. [9]

Raspberry Pi je jednodeskový počítač, což znamená, že je to počítač s jednou deskou plošných spojů různých komponent a za jednu z nejdůležitějších komponent pak lze považovat SoC, která je zabalena v „kovové čepici“ pod niž se nachází integrovaný obvod jenž obsahuje většinu systémů Raspberry Pi mezi které patří CPU, obecně považovaný za mozek počítače a GPU, která je zodpovědná za vizuální část. [9]

Další nezbytnou komponentou tohoto počítače je RAM, což je paměť, která drží v paměti to, co se na Raspberry Pi aktuálně provádí. V případě odpojení počítače od zdroje elektrické energie RAM svůj obsah ztrácí, a proto je zde přítomna komponenta pro čtení microSD karet, kde lze uložit obsah, který má přetrvat i po odpojení Raspberry Pi od zdroje elektrické energie. Na microSD kartě jsou taky uloženy všechny soubory, software a operační systém, který počítač zprovozní. [9]

Pro bezdrátovou komunikaci s jinými zařízeními je možné využívat komponentu rádio, která se skládá ze dvou hlavních komponent a těmi jsou wifi pro připojení počítače k síti a bluetooth pro připojení k periferiím jako jsou bezdrátová myš, nebo mikron. Bluetooth lze využít taky pro komunikaci s tzv. chytrými zařízeními. [9]

Raspberry Pi má řadu portů mezi které patří USB pro připojení periférií, Ethernet port pro připojení počítače k síti. Mimo tyto dva porty na desce lze nalézt taky 3,5mm AV jack, konektor CSI, jenž je určen pro připojení kamery a umožní použít speciálně navržený kamerový modul. Pro připojení Raspberry Pi k obrazovce je zde umístěn port HDMI a pro připojení k dotykové obrazovce je zde konektor DSI. [9]

Při praktickém používání Raspberry Pi je nutné mít funkční operační systém. NOOBS je software, který je navržený tak, aby instalace operačního systému byla co možná nejjednodušší. NOOBS obsahuje standardně dva operační systémy mezi kterými si lze vybrat. Mezi tyto operační systémy patří Raspbian, což je speciální verze operačního systému Debian Linux navržená pro Raspberry Pi a LibreELEC, což je verze Kodi Entertainment Centre software. V případě, že je počítač připojen k síti, tak je možné stáhnout i jiné operační systémy. [9]

Obrázek 8. Raspberry Pi 3 Model B+ [9]

2.1.4 Kamera

Protože pro řízení je nezbytné snímat polohu řízeného objektu, tedy kuličky na nakloněné rovině, je pro tyto účely využívána kamera CCD/CMOS. CCD snímače mají oproti CMOS výhodu v lepší světelné citlivosti, což se projeví při špatném osvětlení. CMOS jsou oproti CCD snímačům levnější, ale tato výhoda je vykoupena horší světelnou citlivostí. Samotná kamera je pro určení polohy nedostatečná, a proto se pro správné určení polohy využívá metod strojového vidění, které mají velkou podporu ze strany open-source softwarových knihoven. [8]

2.2 Popis softwaru

Důležitou součástí reálného modelu je software, který dává modelu jeho správnou funkčnost. V jednotlivých podkapitolách jsou nastíněny klíčové softwarové technologie, které reálný model využívá a také popsáno prostředí ve kterém lze reálný model programovat.

2.2.1 Jazyk C++

Jazyk C++ vyvinul začátkem 80. let Bjarne Stroustrup a jeho hlavní motivací pro tvorbu tohoto jazyka bylo, aby nemusel programovat v assambleru. Stroustrup vycházel u C++ na stručnosti programovacího jazyka C a jeho vhodností pro systémový programovací jazyk, kde jsou úzké vazby na operační systém UNIX. Jazyk C++ lze tedy chápat jako nadstavbu jazyka C a to znamená, že každý funkční program v C je zároveň funkčním programem v C++. Velkou výhodou C++ je, že může využívat softwarové knihovny C, a to je určitě jedna z věcí, která pomohla k rozšíření jazyka C++. Protože tento programovací jazyk podporuje nízkoúrovňové a zároveň i vysokoúrovňové programování, tak je s ním možné programovat jak mikroprocesory, tak i běžné uživatelské aplikace. Jazyk C++ je velmi populární programovací jazyk, a proto lze jeho využití nalézt ve velkém množství hardwarových i OS platformách. Velká výhoda C++ je jeho výkonost a díky této vlastnosti se používá například pro programování embedded systémů, nebo vysoce výkonných serverů. [10]

2.2.2 Qt

Qt je multiplatformní framework, který může být použit díky sadě grafických nástrojů pro tvorbu okenních aplikací, ale může být také velmi užitečný při vytváření tzv. CLI aplikací. Lze ho použít na desktopových operačních systémech, ale jeho použití je možné i na mobilních operačních systémech. [11]

Qt samo o sobě není programovací jazyk. Je to framework napsaný v C++. Qt obsahuje sadu modulů, které pomáhají při tvorbě aplikací. Mezi tyto moduly patří například QtCore, což je základní softwarová knihovna, která poskytuje správu vláken, správu událostí atd. QtGui a QtWidgets jsou softwarové knihovny, kde jsou umístěny grafické sady nástrojů pro tvorbu desktopových aplikací. [11]

Jak již bylo zmíněno, tak Qt obsahuje sadu modulů:

- QtCore – základní knihovna, která poskytuje správu vláken, správu událostí atd.
- QtGui a QtWidgets – sada nástrojů pro tvorbu desktopových aplikací, která poskytuje velké množství grafických komponent k návrhu aplikací.
- QtNetwork – poskytuje sadu tříd pro síťovou komunikaci.
- QtWebkit – jedná se o webkit engine, který dovoluje používat webové stránky a webové aplikace uvnitř Qt aplikace.
- QtSQL – plně kompatibilní abstrakční vrstva pro SQL databáze rozšířená vlastním ovladačem. Podporuje například SQLITE, MySQL a PostgreSQL.
- QtXML – podpora pro jednoduché parsování XML souborů. [11]

Mimo moduly poskytuje Qt taky sadu funkcionalit. Jedna z nejdůležitějších, která se využívá pro komunikaci mezi objekty jsou Signály & Sloty (angl. Signals & Slots). Signál vysílá určitou zprávu, nebo data, pokud nastane určitá událost. Qt samo o sobě má několik předdefinovaných signálů, ale vývojář může přidat i své vlastní. Slot je funkce, která je zavolána jako odezva na určitý signál. [11]

2.2.3 Qt Creator IDE

Qt Creator IDE je multiplatformní IDE (integrované vývojové prostředí angl. Integrated Development Environment) navržené pro potřeby vývojářů v Qt. Qt Creator se hlavně soustředí na poskytování funkcí, které pomůžou novým uživatelům Qt, ale také zároveň aby zvyšovaly produktivitu zkušeným Qt vývojářům. Jedná se o velmi pokročilé IDE, které dovoluje grafický návrh aplikací a ladění (angl. debugging) aplikací. [11]

Mezi hlavní funkce Qt Creator IDE patří:

- Editor kódu s podporou C++, QML a ECMAscript.
- Rychlé nástroje pro navigaci v kódu.

- Zvýraznění syntaxe kódu, výběr dokončení kódu a statická analýza kódu.
- Podpora pro zefektivnění a údržbu kódu. [11]

2.2.4 QCustomPlot

Jedná se o softwarovou knihovnu napsanou pro framework Qt pro potřeby zobrazení datových vizualizací. Hlavní záměr této softwarové knihovny je, aby vizualizace vypadaly dobře a zároveň poskytovaly vysoký výkon pro vytváření a aktualizování vizualizací v reálném čase. QCustomPlot může grafy exportovat do několika formátů jako jsou například PDF soubory, nebo obrázky typy PNG, JPG a BMP. Zároveň je možné QCustomPlot využít pro tvorbu grafů, kde se data aktualizují v reálném čase. [12]

3 GENEROVÁNÍ ŽÁDANÝCH TRAJEKTORIÍ

Kapitola generování žádaných trajektorií se zabývá teoretickým popisem jednotlivých smyček řízení, nad které se při zvoleném typu řízení připojí gyroskopický ovladač a udává tak žádané hodnoty pro řízení. Podle zvoleného typu je pak řízena pozice kuličky na nakloněné rovině, náklon roviny, úhlová rychlost roviny anebo proud motorů. V úvodu je vypracován krátký teoretický základ ke kaskádní P(I)(D) regulaci, jelikož je v reálném modelu tento typ použit k řízení momentu, rychlosti a polohy jednotlivých kloubů reálného modelu.

3.1 PID regulátor

PID regulátor je kombinací tří menších regulátorů, které jsou proporcionální, integrační a derivační. [13]

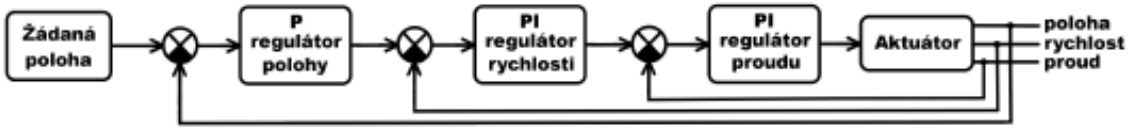
Proporcionální regulátor je ze všech tří nejjednodušší a výstup tohoto regulátoru je přímo úměrný vstupu (odchylce). Pokud je u řízení rychlosti odchylka velká, tak se rychlost zvyšuje maximálně, ale pokud je rychlost malá, tak se rychlost zvyšuje jen minimálně. Problémem u tohoto typu regulátoru je vysoká možnost překmitů od požadované hodnoty, tedy v tomto případě rychlosti a zároveň má proporcionální regulátor trvalou regulační odchylku (matematický model je vždy jen aproximací modelu reálného). [13]

Integrační regulátor nepracuje pouze s hodnotou aktuální, ale pracuje i s předchozími hodnotami odchylek (jedná se o integrál, který se minimalizuje) a díky tomu je možné upravovat řízení systému tak, aby přibližování k minimální odchylce bylo plynulé. [13]

Derivační regulátor má za úkol zkoumat, jaký byl vývoj odchylky – tedy jak moc klesá nebo roste a podle toho upravuje svůj výstup. Derivační složka je nejvíce patrná tam, kde je velká rychlost změny regulační odchylky (tedy její derivace) a proto může systém výrazně rozkmitat a dále zvyšuje vliv šumu v systému. S derivační složkou se musí zacházet opatrně, protože může mít nepříznivý vliv na celý systém. [13]

3.2 Kaskádní regulace

Kaskádní regulace je nejrozšířenější regulace, která se v průmyslu používá pro řízení pohybu. Používá zmíněné regulátory PID, kde jsou dvě smyčky pro řízení rychlosti a tři smyčky pro řízení polohy a parametry regulátorů se pak nastavují od proudové smyčky, přes rychlostní až po polohovou. [8]



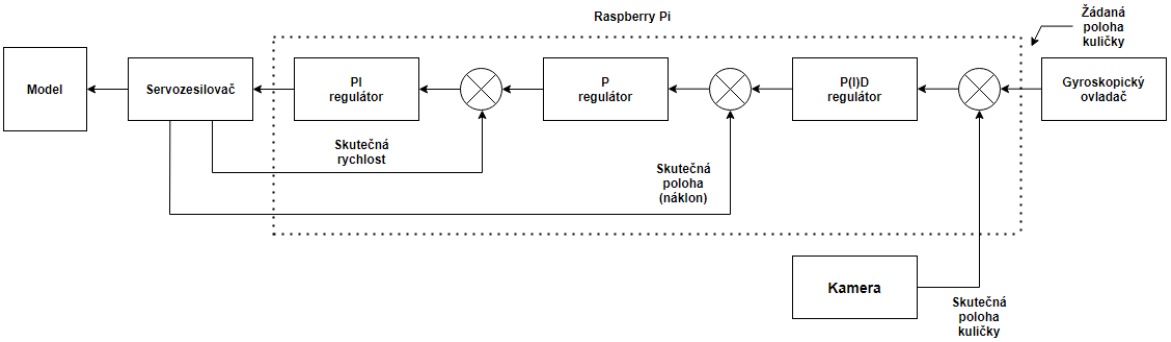
Obrázek 9. Zjednodušený blokový diagram kaskádní regulace [8]

Často používaná struktura regulátorů je zobrazena na Obrázek 9, kde PI je regulátor proudu, PI regulátor otáček a P regulátor polohy. [8]

3.3 Poloha kuličky

U nejvyššího stupně řízení se gyroskopický ovladač napojuje přímo na smyčku, která reguluje skutečnou polohu kuličky na nakloněné rovině a zjednodušený blokový diagram tohoto řízení je zobrazen na Obrázek 10. Výstupem z gyroskopického ovladače je žádaná poloha kuličky, která vstupuje do sumačního členu a do kterého zároveň jako zpětná vazba z kamery vstupuje skutečná pozice kuličky na nakloněné rovině. Výsledkem z tohoto sumačního členu je regulační odchylka, jenž vstupuje do P(I)D regulátoru, který reguluje polohu náklonu plošiny (integrační složka je v závorkách, protože se připojuje jen když je kulička v definované vzdálenosti od žádané hodnoty). Výstup z tohoto P(I)D regulátoru je potom žádaná poloha náklonu plošiny, která vstupuje do sumačního členu, do kterého zároveň vstupuje zpětnou vazbou skutečná poloha náklonu plošiny a výsledkem je regulační odchylka pro P regulátor, který reguluje úhel náklonu plošiny. Výstupem z P regulátoru je žádaná úhlová rychlost náklonu plošiny a vstupuje opět do sumačního členu a zároveň druhý vstup do tohoto sumačního členu je přes zpětnou vazbu skutečná úhlová rychlost náklonu plošiny. Výstupem ze sumačního členu je regulační odchylka pro PI regulátor, který reguluje úhlovou rychlost náklonu plošiny. Výstupem tohoto PI regulátoru je žádaný proud, jenž vstupuje do stereozesilovače, což je proudová smyčka, která reguluje akční členy reálného modelu. [8]

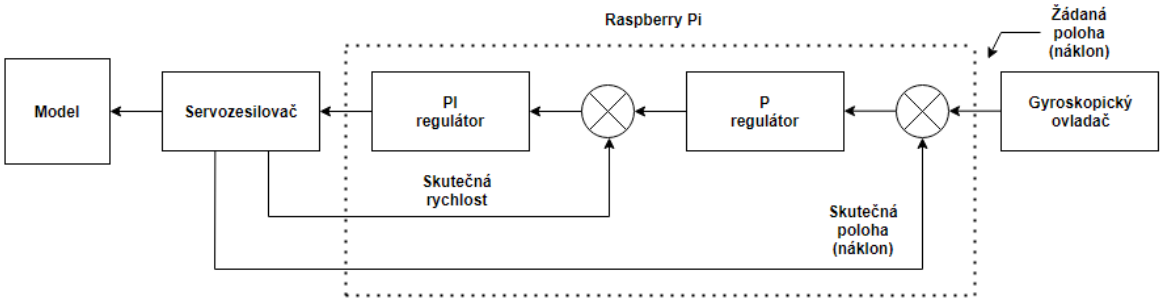
Všechny tři regulátory jsou naprogramovány na platformě Raspberry Pi a gyroskopický ovladač se tedy napojuje pouze na zvolenou smyčku řízení. Nutné je poznamenat, že na Obrázek 10. je pouze zjednodušený blokový diagram, protože každá osa reálného modelu má vlastní regulaci. [8]



Obrázek 10. Zjednodušený blokový diagram řízení pozice kuličky [8]

3.4 Náklon roviny

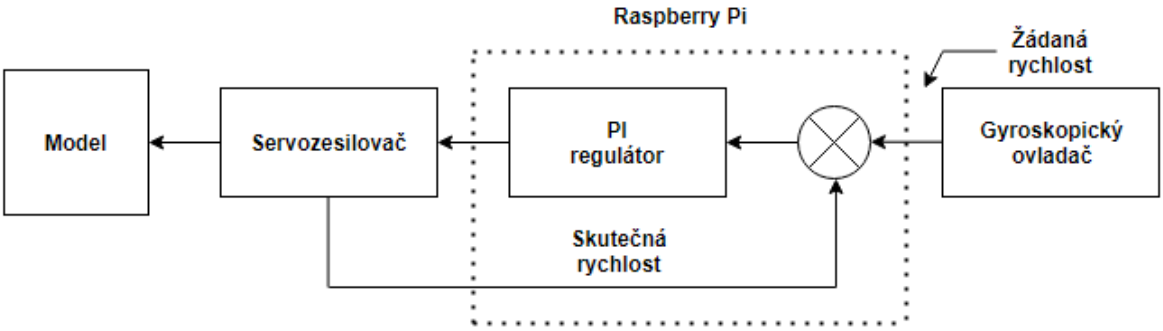
Řízení náklonu roviny je druhý nejvyšší stupeň řízení a je zobrazen na Obrázek 11. Výstupem z gyroskopického ovladače je žádaná poloha náklonu roviny, jenž vstupuje do sumačního členu a do kterého zároveň vstupuje přes zpětnou vazbu skutečná poloha náklonu roviny. Výstupem sumačního členu je pak regulační odchylka pro P regulátor, který reguluje úhel náklonu plošiny. Další regulace v tomto stupni řízení je totožná s řízením polohy kuličky popsaném v předešlé podkapitole Poloha kuličky. Jak lze tedy pochopit z popisu a ze zjednodušeného diagramu na Obrázek 11. je řízení náklonu roviny přímo z gyroskopického ovladače možné docílit napojením na nižší smyčku řízení. [8]



Obrázek 11. Zjednodušený blokový diagram řízení náklonu roviny [8]

3.5 Úhlová rychlost roviny

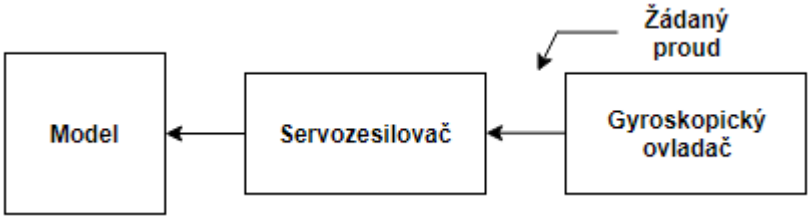
Řízení úhlové rychlosti roviny je dosaženo opět napojením na nižší stupeň řízení a zjednodušený diagram je zobrazen na Obrázek 12. Zjednodušený blokový diagram řízení úhlové rychlosti roviny [8] V tomto případě je výstupem z gyroskopického ovladače žádaná úhlová rychlost roviny, která vstupuje do sumačního členu a do kterého přes zpětnou vazbu vstupuje skutečná úhlová rychlost roviny a výsledkem je potom regulační odchylka pro PI regulátor, který reguluje úhlovou rychlost náklonu plošiny. Další regulace je pak zase opět totožná s předešlými popsanými řízeními. [8]



Obrázek 12. Zjednodušený blokový diagram řízení úhlové rychlosti roviny [8]

3.6 Proud motorů

Řízení proudu motorů je nejnižší stupeň řízení, protože výstupem z gyroskopického ovladače je přímo žádaný proud pro servozesilovač. Zjednodušený diagram pro toto řízení je zobrazen na Obrázek 13. Toto řízení nemá žádnou zpětnou vazbu a je možné ho tedy nazvat spíše ovládáním. [8]



Obrázek 13. Zjednodušený blokový diagram řízení proudu motorů [8]

4 SNÍMÁNÍ VELIČIN, VÝPOČET VELIČIN, GRAFICKÁ VIZUALIZACE A EXPORT VELIČIN

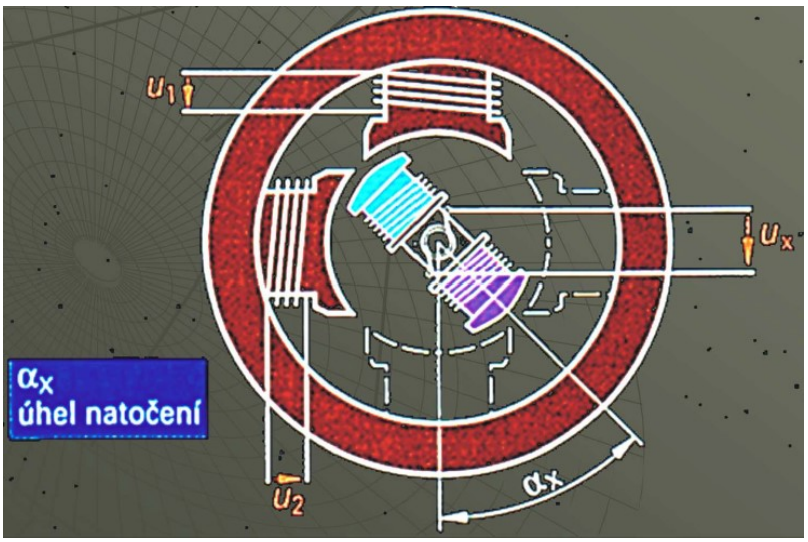
Reálný model pro měření veličin, tedy úhlu natočení plošiny, používá enkodéry kombinované s resolverem, a proto je v začátku kapitoly teoreticky rozebrán resolver a enkodér. Následně je teoreticky rozebráno, jak jsou další potřebné veličiny jako například úhlová rychlost a úhlové zrychlení počítány. Součástí této kapitoly je také popis pojmů, které se používají v grafické interpretaci snímaných veličin, konkrétně perioda vzorkování a spouštění (angl. trigger). Na závěr je popsána grafická vizualizace a export těchto veličin.

4.1 Resolver

Resolver je transformátorový senzor, který vyhodnocuje úhel natočení. Konstrukčně je podobný asynchronnímu stroji a je tedy složený ze dvou statorových vinutí, která jsou vzájemně posunuta o 90 stupňů a napájena střídavými napěťovými signály. Tyto střídavé napěťové signály jsou fázově posunuta o 90 stupňů (10,11). [14][15]

$$u_1 = \widehat{u_1} \cdot \sin \omega t \tag{10}$$

$$u_2 = \widehat{u_2} \cdot \cos \omega t \tag{11}$$

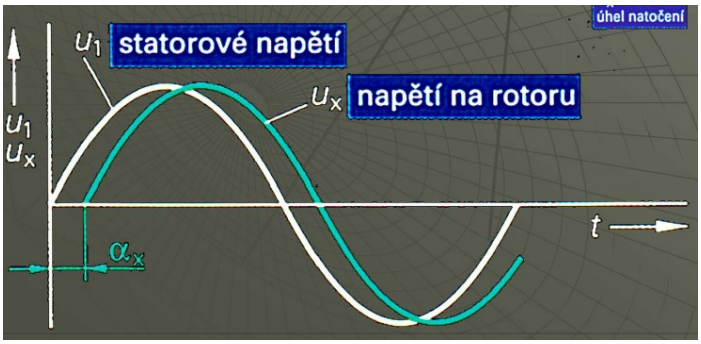


Obrázek 14. Resolver [15]

V pasivním vinutí rotoru je indukován napěťový signál, který je složený ze střídavých napěťových signálů indukovaných oběma statorovými vinutími. Natočení rotoru pak odpovídá poměru amplitud indukovaných napěťových signálů. Výstupní napětí na rotoru u_x je pak při úhlu natočení α_x rovno (6). [15]

$$u_x(t) = u_1(t) \cos \alpha_x + u_2(t) \sin \alpha_x \tag{12}$$

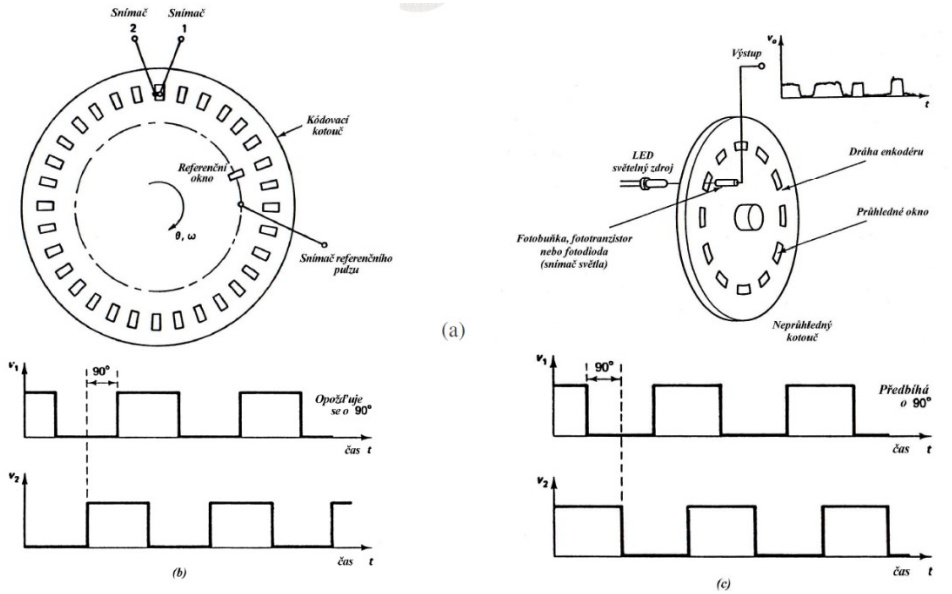
Výstupní napětí u_x je posunuto oproti napájecímu napětí statorů u_1, u_2 o úhel, který souhlasí s úhlem natočení rotoru. Úhel fázového posunu indukovaného napětí je rovný úhlu natočení rotoru snímače. Výsledné natočení je tedy možné vyhodnotit a zjistit úhel natočení. [15]



Obrázek 15. Graf výstupního napětí [15]

4.2 Enkodér

Enkodér neboli rotační senzor je zařízení, které je možné použít pro měření délky, pozice nebo úhlu natočení. Měření těchto veličin probíhá pomocí přeměny mechanického pohybu na elektrický signál. Nejdéle zavedenými a hojně využívanými enkodéry v robotice jsou optické enkodéry, které při rotačním pohybu generují pulsy a určitý počet těchto pulsů je úměrný vzdálenosti, natočení anebo rychlosti (závislost počtu pulsů na časový úsek). [15]



Obrázek 16. Enkodér [15]

Na Obrázek 16 je znázorněn inkrementální úhlový enkodér, který pro vyhodnocení využívá přenos světla a jedná se tedy o optický enkodér. Světelný signál, který vychází z LED světelného zdroje předzpracovává disk, jehož štěrbinami prochází tento generovaný světelný

signál. Generovaný světelný signál je přijatý fotobuňkou, fototranzistorem nebo fotodiodou, jenž jsou předsazeny o půl štěrbiny. Jakmile se disk otáčí, tak generované světlo dopadá na fotobuňku, fototranzistor nebo fotodiodu a tím vytvoří výstupní sledy impulzů a tímto sčítáním pulzů lze určit výslednou vzdálenost. Úhlové rozlišení, které enkodéru je možné vyjádřit funkcí (13), kde n_s je počet štěrbin a D_w je tzv. aktivní délka fotodetektorů. [15]

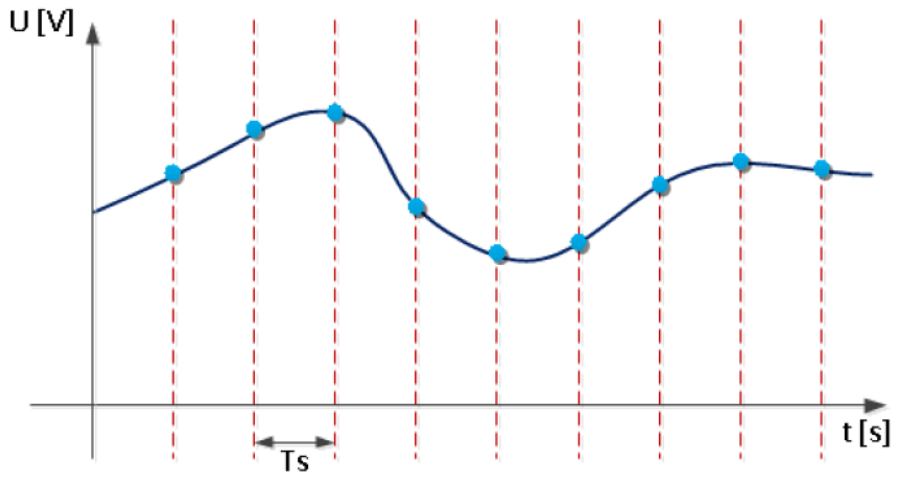
$$\Delta\theta = \frac{360}{n_s};\ D_w = r \cdot \sin\left(\frac{\Delta\theta}{2}\right) \tag{13}$$

Výstupní signál optického enkodéru však informuje jen o pohybu, a ne o přesné pozici. Výslednou pozici však lze získat ze znalosti výchozí pozice a externí měřící jednotky. [14][15]

4.3 Perioda vzorkování

Perioda vzorkování T_s je časový interval mezi dvěma po sobě následujícími vzorky. Jedná se o převrácenou hodnotu vzorkovací frekvence f_s (14). [15][16]

$$f_s = \frac{1}{T_s} \tag{14}$$



Obrázek 17. Perioda vzorkování [16]

Perioda vzorkování je obvykle konstantní, ale v pokročilých systémech se může měnit v závislosti na dynamice měření. Zvolení vhodné vzorkovací frekvence je nezbytné pro stabilní systém. Pokud by byla perioda vzorkování zvolena nevhodně, tak by se ze stabilního systému mohl stát systém nestabilní. [16]

4.4 Spouštění

Při záznamu hodnot ze systému není vždy žádoucí zaznamenávat všechny hodnoty, ale jen část, která je pro pozorovatele užitečná. Zpravidla se spouští zaznamenávání hodnot v momentě, kdy vstupní signál splňuje předem definované podmínky. Aby bylo možné tento cíl splnit, tak je k tomu nutné využít princip spouštění (angl. trigger), který funguje tak, že vyhodnocovací část systému neustále sleduje vstupní, nebo výstupní signál a jakmile jsou podmínky pro záznam signálu splněny, tak se signál začne zaznamenávat a následně zobrazovat. [17]

4.5 Výpočet veličin

Jelikož ne všechny veličiny poskytují senzory reálného modelu, tak je nutné určité veličiny vypočítat. Mezi tyto veličiny patří rychlost kuličky na nakloněné rovině, kterou je možné vypočítat z polohy kuličky. Dále se jedná o úhlovou rychlost a úhlové zrychlení nakloněné roviny, které je nutné dopočítat z úhlu natočení nakloněné roviny.

4.5.1 Výpočet rychlosti kuličky na nakloněné rovině

Jak již bylo popsáno v začátku této podkapitoly, reálný model poskytuje pouze informace o poloze. Poloha kuličky na nakloněné rovině je určena z kamery a rychlost je tedy nutné dopočítat. Polohu kuličky na nakloněné rovině je možné popsat jejím polohovým vektorem *r*. V kartézské soustavě souřadnic je možné tento polohový vektor vyjádřit ve tvaru (15). [18]

$$r = xi + yj + zk \tag{15}$$

kde i, j, k jsou průměty vektoru do souřadnicových os a x, y, z jsou složky vektoru. Při pohybu kuličky se zároveň mění i její polohový vektor. Složky polohového vektoru jsou pak tedy funkcemi času $x(t)$, $y(t)$, $z(t)$ a polohový vektor kuličky pak lze také označit jako funkci času $r(t)$. [18]

Pokud je poloha kuličky v okamžiku t_1 stanovena vektorem r_1, pak v následujícím okamžiku je stanovena jako $t_1 + \Delta t$ vektorem r_2, kde posunutí Δr kuličky v časovém intervalu Δt je dáno rozdílem (16) a výsledné posunutí kuličky je pak možné vyjádřit rovnicí (17). [18]

$$\Delta \mathrm{r} = r_2 - r_1 \tag{16}$$

$$\Delta \mathrm{r} = (x_2 - x_1)i + (y_2 - y_1)j + (z_2 - z_1)k \tag{17}$$

Průměrná rychlost kuličky $\bar{v}$ v časovém intervalu Δt, která je měřena od okamžiku t do okamžiku $t + \Delta t$ je formulována jako podíl vektoru posunutí Δr a délky časového intervalu Δt (18). [18]

$$\bar{v} = \frac{\Delta \mathrm{r}}{\Delta \mathrm{t}} \tag{18}$$

Okamžitá rychlost kuličky v je formulována jako limita průměrné rychlosti kuličky $\bar{v}$ pro $\Delta t \to 0$ a to je derivace polohového vektoru kuličky r (19,20). [18].

$$v = \frac{dr}{dt} \tag{19}$$

$$v = \frac{d}{dt}(xi + yj + zk) = \frac{dx}{dt}i + \frac{dy}{dt}j + \frac{dz}{dt}k \tag{20}$$

Jelikož z reálného modelu nejsou dostupné informace o poloze v každém okamžiku, ale jen v určitých okamžicích, je pro výpočet nezbytné použít numerickou derivaci (21). [19]

$$f'(x) = \lim_{h \to 0} \frac{f(x+h) - f(x)}{h} \tag{21}$$

4.5.2 Výpočet úhlové rychlosti a úhlového zrychlení nakloněné roviny

Obdobně jako u rychlosti kuličky reálný model neposkytuje informace o úhlové rychlosti a úhlovém zrychlení roviny. Máme jen informace o úhlu natočení roviny θ a zbylé dvě hodnoty je nutné dopočítat. Úhlu natočení roviny θ lze rozumět jako úhel, který svírá rovina se vztažnou soustavou. Pokud se úhel natočení roviny mění v čase, tak lze tomuto termínu říkat otáčení. Otáčení roviny $\Delta\theta$ je vyjádřeno jako (22). [18]

$$\Delta\theta = \theta_2 - \theta_1 \tag{22}$$

Průměrná úhlová rychlost roviny $\bar{\omega}$ v časovém intervalu Δt je formulována vztahem (23). [18]

$$\bar{\omega} = \frac{\Delta\theta}{\Delta t} \tag{23}$$

Okamžitá úhlová rychlost roviny ω je formulována jako limita průměrně úhlové rychlosti roviny $\bar{\omega}$ pro $\Delta t \to 0$ a to je derivace úhlu natočení roviny θ, kterou lze formulovat vztahem (24). [18]

$$\omega = \frac{d\theta}{dt} \tag{24}$$

Pokud rychlost roviny ω není konstantní, tak má rovina nenulové úhlové zrychlení ϵ, jenž je možné vyjádřit vztahem (25) jako derivaci okamžité úhlové rychlosti ω. [18]

$$\epsilon = \frac{d\omega}{dt} \tag{25}$$

Stejně jako u výpočtu rychlosti kuličky nemáme informace o úhlu natočení v každém okamžiku, ale jen v určitých okamžicích, a proto je pro výpočet nezbytné použít opět numerickou derivaci (21).

4.6 Grafická vizualizace veličin

Veličiny, které reálný model poskytuje jsou v mnoha případech vyhovující, ale pro určité případy je lepší tyto veličiny graficky vizualizovat, například kvůli analýze chování reálného modelu. Děje, které při fungování reálného modelu probíhají může být složité pochopit jen pomocí matematického vyjádření, a proto je důležité mít možnost tyto děje znázornit pomocí grafů. [20]

Pro správný uživatelský přínos grafu je však nutné dodržovat určitá pravidla, aby se uživatel dokázal v grafu orientovat a pochopit výsledky grafické vizualizace. Velmi důležité je mít graf správně popsaný a správně popsané hodnoty, protože bez těchto informací jen prázdný graf neposkytne uživateli téměř žádné informace. [20]

4.6.1 Zásady a terminologie vizualizace dat

Graf je jedním z nejvíce používaným stylem pro grafickou vizualizaci veličin. Jelikož veličinu jako takovou lze vyjádřit číslem, tak je nutné mít pro správné zobrazení sadu takových veličin, kterou lze ve správné terminologii nazvat jako datová řada. Velký podíl grafů používá k zobrazení dat do grafu kartézskou soustavu souřadnic. Kromě zobrazení dat v kartézských souřadnicích lze použit jiný styl zobrazení, například tzv. koláčového grafu, mozaikového grafu apod. [20]

V případě vizualizace dat, kde graf je jednorozměrný, jsou na vodorovné ose (osa *x*) data nezávislé proměnné a na svislé ose (osa *y*) jsou data závislé proměnné. Pokud se jedná o statická data, tak se obvykle na vodorovné ose vyskytuje časová složka dat. Jedná-li se o zkoumání nějaké statistické sady dat, která je rozmístěna ve více skupinách, tak se můžou na vodorovné ose vyskytnout typy těchto skupin. [20]

Jak již bylo zmíněno na začátku této podkapitoly, tak nezbytnou součástí správné interpretace grafů jsou jeho popisky. Důležitým prvkem grafu je název grafu, který informuje uživatele, o jakou vizualizaci se v grafu jedná. Další nezbytnou součástí správného grafu jsou popisky os, které by měly poskytovat informace o jednotkách a zároveň

poskytnout informaci o hodnotě v daném místě osy. Pro zdůrazňování trendu jsou obvykle jednotlivé osy propojeny úsečkami. V případě, že je v grafu více než jedna křivka, je nezbytné tyto křivky popsat pomocí legendy. [20]

4.7 Export veličin

Z grafického rozhraní pro ovládání a nastavení reálného modelu je možné vyexportovat veličiny. Export a ukládání veličin do určitého standardu je důležitý pro zachování integrity s ostatními typy softwarových nástrojů a aplikací, díky čemuž jsou veličiny reálného modelu exportovány ve formátu CSV (angl. comma separated values).

4.7.1 Obecný popis formátu CSV

CSV formát je používán pro výměnu a konvertování dat mezi různými programy. Soubor, který je ve formátu CSV se skládá z řádků, kde jsou položky odděleny čárkou. Existují jazyky (mezi které patří i čeština), kde se čárka používá jako oddělovač desetinných míst a z toho důvodu je nutné, aby formát CSV podporoval i jiné oddělení než čárkou a z tohoto důvodu je možné využít pro oddělení jednotlivých hodnot středník nebo tabulátor. Díky své jednoduchosti a nenáročnosti je možné CSV číst i bez specializovaného softwaru, což je velká výhoda oproti jiným formátům. [21]

4.7.2 Definice formátu CSV

Pro správnou implementaci CSV formátu je nutné dodržet následující pravidla:

- Každý záznam je umístěn na samostatném řádku a je ohraničen odřádkováním (CRLF).
- Poslední řádek v záznamu může, ale nemusí mít odřádkování (CRLF).
- CSV soubor může, ale nemusí mít řádek s hlavičkou. Pokud soubor CSV hlavičku obsahuje, tak musí být na prvním řádku souboru. Tato hlavička by měla obsahovat jména odpovídající sloupcům v souboru.
- Každý řádek by měl mít stejný počet sloupců v celém souboru. Mezery jsou považovány za část sloupce a neměly by být ignorovány.
- Každý řádek může, ale nemusí být uzavřen ve dvojitých uvozovkách (některé programy však dvojité uvozovky nevyužívají – například Microsoft Excel).

- Sloupce, které obsahují odřádkování by měly být uzavřeny ve dvojitých uvozovkách. [21]

II. PRAKTICKÁ ČÁST

5 DEFINOVÁNÍ POŽADAVKŮ, ANALÝZA APLIKACE A NÁVRH IMPLEMENTACE

V této části jsou vydefinovány požadavky pro implementaci, provedena analýza již existující aplikace pro reálný model a návrh implementace žádaných trajektorií, grafické vizualizace a exportu snímaných veličin do této existující aplikace pro reálný model. Jsou zde prakticky využity a znázorněny informace, které jsou popsány v teoretické části ohledně generování žádaných trajektorií, grafické implementace nebo exportu veličin.

5.1 Definování požadavků a analýza aplikace

Při návrhu je nutné vzít v potaz, že se pro reálný model nevytváří nová aplikace, ale implementace by měla probíhat do již existující aplikace, která ovládá a nastavuje reálný model. Proto je před samotným návrhem implementace důležité definovat požadavky a zdokumentovat, jak je aplikace navržena.

5.1.1 Požadavky pro generování žádaných trajektorií

Požadavek pro tuto část je funkční napojení na jednotlivé smyčky řízení, které jsou popsány v teoretické části. Jedná se o připojení na smyčky, které regulují skutečnou polohu kuličky na nakloněné rovině, náklon roviny, úhlovou rychlost roviny a na závěr proud motorů. Bude tedy nezbytné v aplikaci přidat novou záložku, kde budou vybrány jednotlivé formy řízení a následný pohyb s gyroskopickým ovladačem, který bude generovat žádané hodnoty a bude ovládat reálný model podle zvoleného typu řízení. V záložce se také budou zobrazovat žádané a aktuální snímané veličiny podle vybraného typu řízení.

5.1.2 Požadavky pro grafickou vizualizaci

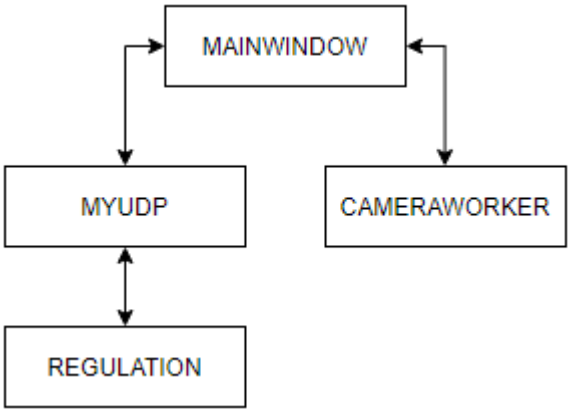
Požadavek pro tuto část je funkční grafická vizualizace snímaných veličin a schopnost nastavení jednotlivých prvků, které grafická vizualizace potřebuje pro správnou funkčnost, mezi které patří zvolení, jaké veličiny se mají zobrazovat, nastavení periody vzorkování, nastavení spouštění (angl. triggering) atd. Důležité je dodržení zásad a terminologie u vizualizace dat, jež jsou popsány v teoretické části této práce. Celkově se tedy bude jednat o přidání nové záložky do aplikace, kde bude umístěn hlavní graf, který bude zobrazovat žádané i aktuální veličiny. Dále by se na spodní části aplikace měl vyskytovat graf, který bude zaznamenávat hodnoty nepřetržitě od spuštění aplikace, ale na rozdíl od hlavního grafu by měl zaznamenávat pouze aktuální hodnoty.

5.1.3 Požadavky pro export veličin

Požadavek pro tuto část je funkční export veličin do formátu CSV a zároveň dodržet správný formát souboru, aby bylo možné tento soubor otevřít v jiných programech a udělat případnou analýzu nad exportovanými veličinami. Export veličin bude probíhat jen u veličin, které jsou zobrazovány v hlavním grafu – tedy aktuální a žádané veličiny. Tlačítko pro export veličin by se mělo vyskytovat v nově vytvořené záložce pro grafickou vizualizaci.

5.1.4 Analýza aplikace

Pro analýzu aplikace je vytvořen diagram, který je zobrazen na Obrázek 18. Hlavní třída, která zajišťuje komunikaci a prostředky pro další třídy v aplikaci se nazývá MAINWINDOW. V třídě MAINWINDOW zároveň probíhá veškerá správa uživatelského rozhraní, mezi které patří registrace na zachytávání kliku na tlačítka, změna hodnot ve formulářích atd. V třídě MYUDP je zajištěna komunikace mezi servozesilovačem a řídícím počítačem, což je potřebné pro správnou výměnu dat. Třída CAMERAWORKER má za úkol strojové zpracování obrazu, tedy hlavně určení správné polohy kuličky. Poslední třída, která prozatím nebyla popsána je REGULATION, která zajišťuje správnou regulaci celého reálného modelu.



Obrázek 18. Diagram aplikace pro reálný model

Veškerá komunikace mezi jednotlivými třídami probíhá pomocí funkcionality Signály & Sloty (angl. Signals & Slots). Tuto funkcionalitu je nutné vzít v potaz, protože grafická vizualizace si musí nastavit vlastní periodu vzorkování, a tedy musí komunikovat s ostatními objekty, a to je právě zajištěno pomocí funkcionality Signály & Sloty.

Aplikace využívá starší verzi Qt (konkrétně verzi 5.7) a Qt má nativní třídu pro tvorbu grafů až od verze 5.13, a proto je nutné využít pro grafickou vizualizaci veličin ve formě grafů externí knihovnu, která bude požadavky pro účel reálného modelu splňovat.

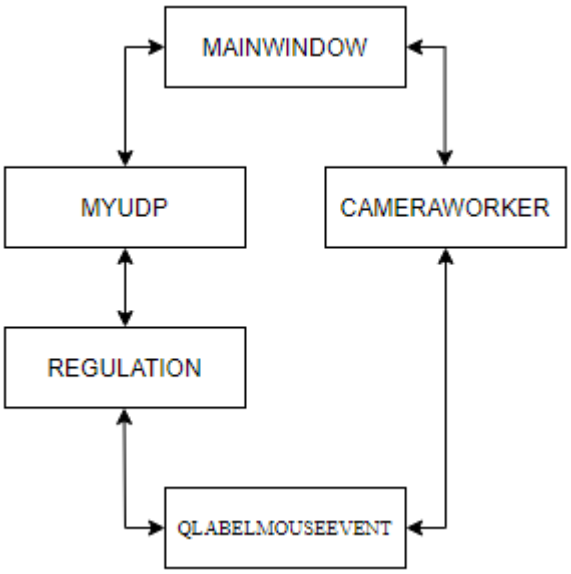
Třída MAINWINDOW zahrnuje hlavičkový soubor mainwindow.h, kde se deklaruje, co by třída měla obsahovat a soubor mainwindow.cpp pak obsahuje implementaci vybraných metod.

5.2 Návrh implementace

Po definování a analýze aplikace je možné udělat návrh implementace výše zmíněných požadavků. Návrh je rozdělen do dvou fází, kde první se zabývá návrhem pro implementaci generovaní žádaných trajektorií a druhá implementací grafické vizualizace a exportu snímaných veličin.

5.2.1 Návrh implementace generování žádaných trajektorií

Při návrhu implementace generování žádaných trajektorií je nutné vycházet z již existující struktury aplikace, která je zobrazena na Obrázek 18. Jelikož není možné se připojit přímo na gyroskop, případně akcelerometr v gyroskopickém ovladači a ovladač si sám přepočítává data na pozici kurzoru na obrazovce, což je popsáno v teoretické části, tak je nutné vytvořit třídu QLABELMOUSEEVENT, která bude pozici kurzoru přepočítávat na požadované hodnoty pro řízení reálného modelu. Uspořádání zahrnující novou třídu je zobrazeno na Obrázek 19.



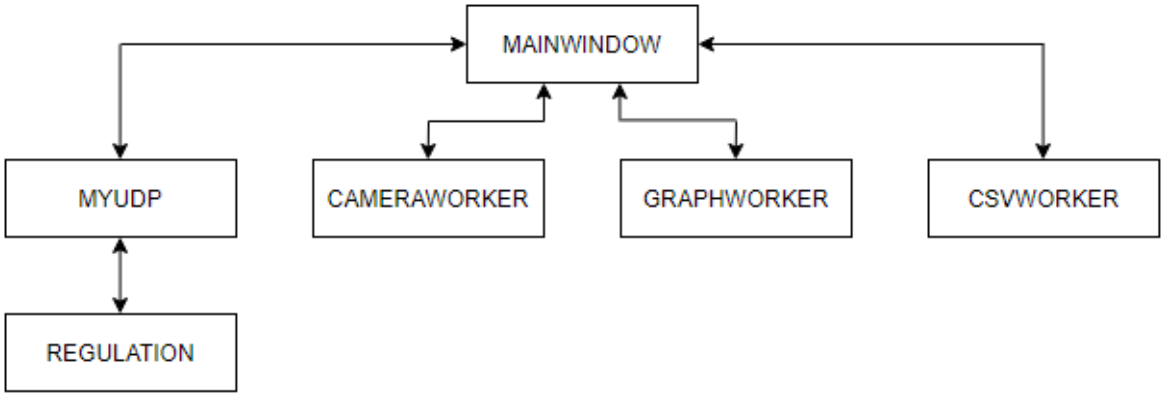
Obrázek 19. Diagram aplikace s novou třídou

Třída QLABELMOUSEEVENT by měla zajišťovat zmíněný přepočet pro požadované hodnoty a odesílat tyto hodnoty pomocí funkcionality Signály & Sloty (angl. Signals & Slots) do třídy CAMERAWORKER, pokud se jedná o žádanou hodnotu pro polohu kuličky

na nakloněné rovině, nebo do třídy REGULATION, kde se zpracovává náklon roviny, úhlová rychlost roviny nebo proud motorů.

5.2.2 Návrh implementace grafické vizualizace a exportu snímaných veličin

Návrh vychází již z existující aplikace, jejíž struktura je znázorněna na Obrázek 18. Protože se v této části jedná o doplnění možnosti grafické vizualizace a exportu veličin, tak do stávajícího řešení by měly být doimplementovány dvě nové třídy GRAPHWORKER a CSVWORKER. Uspořádání zahrnující dvě nové třídy je zobrazeno na Obrázek 20.



Obrázek 20. Diagram aplikace pro reálný model s novými třídami

Třída GRAPHWORKER by měla zajišťovat abstrakci nad knihovnou QCustomPlot a poskytovat sadu metod pro vytváření grafů a komunikaci s grafy.

Třída CSVWORKER by měla zajišťovat export veličin do CSV. Tedy formátování a validaci exportovaných veličin a zároveň by měla poskytnout výběr uživateli, kde chce výsledný CSV soubor s exportovanými veličinami uložit v souborovém systému.

6 IMPLEMENTACE

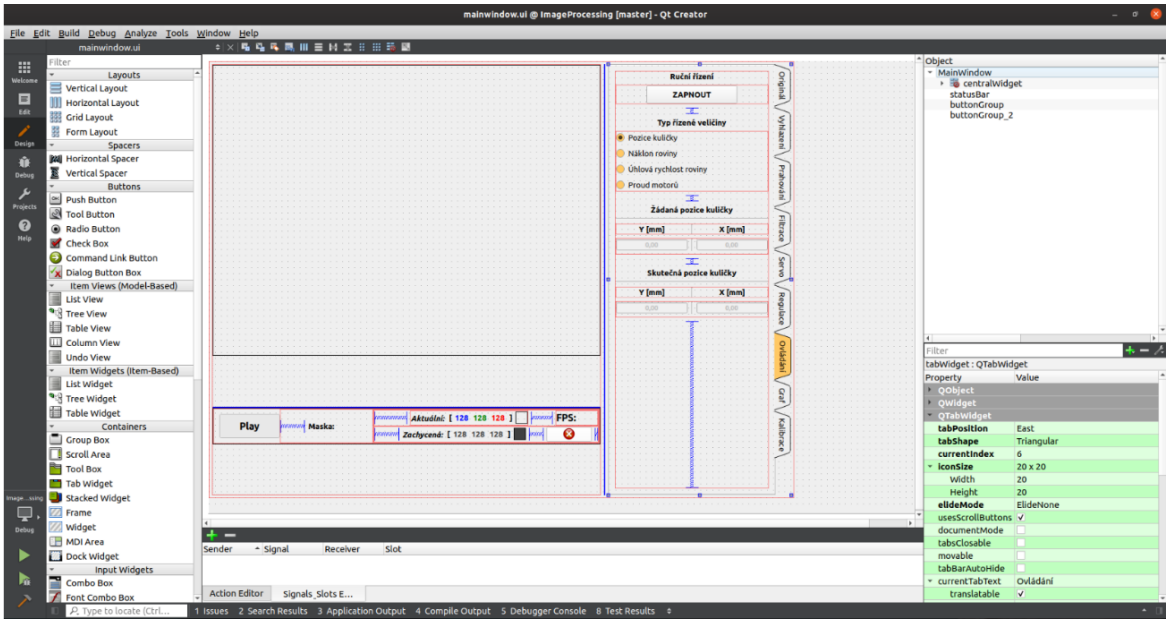
V této části jsou popsány všechny klíčové části implementace potřebné ke správnému splnění požadavků definovaných v předešlé kapitole a případnému vydefinování problémů, které nastaly při implementaci a návrh řešení. V jednotlivých podkapitolách jsou popsány všechny klíčové kroky implementace, ať se jedná o práci s IDE, nebo ukázkou funkčního kódu implementace – není zde však zobrazen kompletní kód, ale jen klíčové funkcionality. Kompletní kód je možné nalézt v příloze.

6.1 Implementace žádaných trajektorií

Implementace žádaných trajektorií zahrnuje úpravu uživatelského rozhraní aplikace pro reálný model a následné napojení na jednotlivé smyčky řízení, které budou generovat žádané trajektorie podle zvoleného typu řízení.

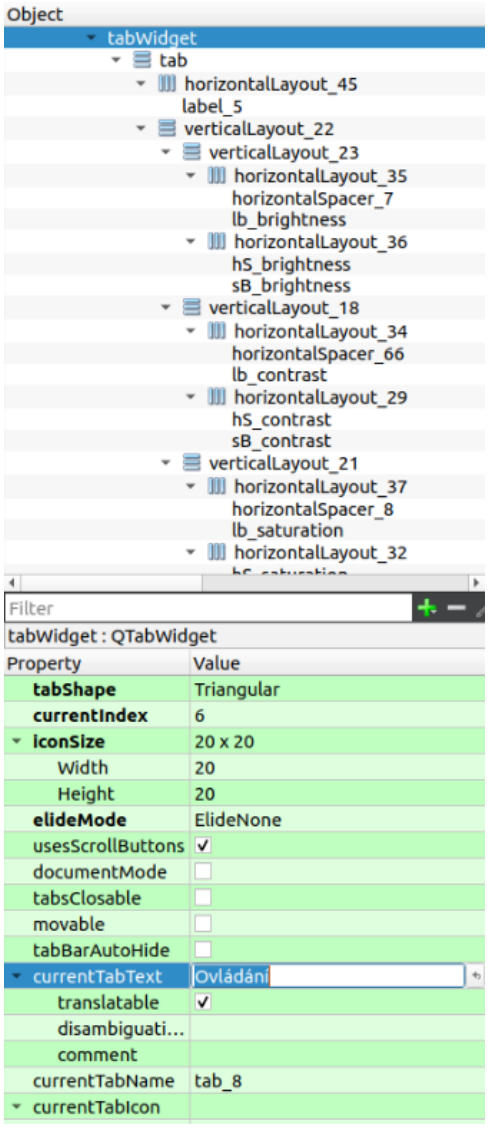
6.1.1 Přidání nové záložky

V aktuální aplikaci existuje soubor mainwindow.ui, kde je definováno uživatelské rozhraní a IDE Qt Creator má grafickou nadstavbu nad tímto souborem zobrazeném na Obrázek 21. Díky této grafické nadstavbě je možné snadno vytvářet, nebo editovat uživatelské rozhraní. Pro přidání nové záložky je tedy nutné otevřít tuto grafickou nadstavbu a kliknout na záložku za kterou má být přidána nová záložka, kliknout pravým tlačítkem myši a zvolit vložit stránku → za aktuální stránku (angl. Insert Page → After Current Page).



Obrázek 21. Qt Creator – grafická nadstavba pro vytváření uživatelského rozhraní

Záložka je pojmenována ve výchozím nastavení jako Stránka (angl. Page), kterou je nutné přejmenovat. Přejmenování se provede opět v grafické nadstavbě tak, že se musí označit nově vytvořená záložka tím, že se provede klik levým tlačítkem myši na zvolenou záložku. V nastavení poté napsat požadovaný název v políčku currentTabIndex. Výše popsané nastavení nového jména záložky je zobrazeno na Obrázek 22.

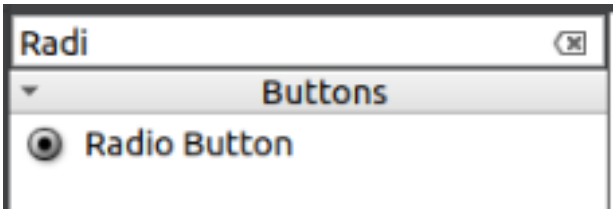


Obrázek 22. Přejmenování záložky

6.1.2 Přidání ovládacích prvků a implementace zobrazování veličin

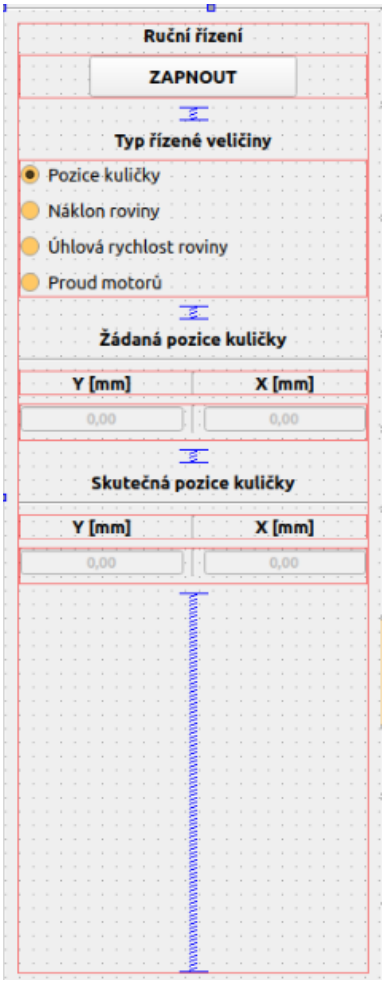
Po přidání nové záložky se vytvoří prostor pro implementaci ovládacích prvků, které jsou potřeba ke správnému generování žádaných trajektorií. Je třeba přidat možnost zapnutí ovládání reálného modelu pomocí gyroskopického ovladače, výběr typu řízení a zobrazení žádaných a skutečných veličin. Přidání nových ovládacích prvků probíhá tak, že se ve vyhledávacím poli zobrazeném na Obrázek 23 vyhledá požadovaný ovládací prvek a pomocí

funkcionality vezmi a přesuň (angl. drag and drop) vloží na vybrané místo v uživatelském rozhraní aplikace. Qt Creator IDE vygeneruje pro tento nový ovládací prvek specifický identifikátor, ale je možné ho změnit.



Obrázek 23. Vyhledávání ovládacích prvků

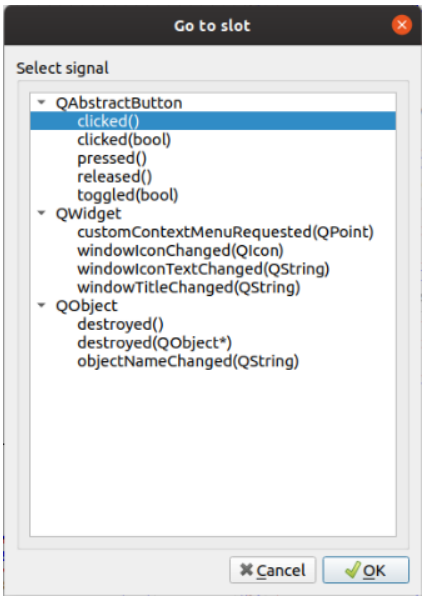
Obdobně pak probíhá přidání dalších ovládacích prvků do uživatelského rozhraní aplikace a finální stav uživatelského rozraní je na Obrázek 24.



Obrázek 24. Uživatelské rozhraní pro ovládání

Přidání ovládacích prvků je první část, protože ovládací prvky musí reagovat na akce od uživatelů (například kliknutí), je nutné tuto část naprogramovat. Je třeba kliknout pravým

tlačítkem na ovládací prvek a zvolit jít do slotu (angl. Go to slot), což způsobí otevření dialogu zobrazeném na Obrázek 25.



Obrázek 25. Výběr akcí
pro ovládací prvky

V dialogu na Obrázek 25 je několik možností, které je možné vybrat. Při implementaci chování tlačítka ZAPNOUT, které je vidět na Obrázek 24 se volí možnost clicked() a výběr se potvrdí tlačítkem OK. Tato popsaná akce vytvoří v souboru mainwindow.cpp prázdnou metodu on_pb_ManualControl_clicked (tlačítko ZAPNOUT má identifikátor pb_ManualControl) a do této metody je pak možné doprogramovat logiku, která po kliknutí na tlačítko ZAPNOUT má nastat. Tato logika je popsána v další části této kapitoly při popisování implementování žádaných trajektorií. Další ovládací prvky, které jsou na Obrázek 24 a k nim příslušné metody se vytvářejí obdobným způsobem, jak bylo popsáno zde.

Prvky, které zobrazují žádané a skutečné hodnoty v uživatelském rozhraní fungují „obráceným“ způsobem, protože jejich obsah se aktualizuje v mainwindow.cpp. Pomocí funkcionality Signály & Sloty (angl. Signals & Slots) je možná komunikace mezi mainwindow.cpp a regulation.cpp a tím pádem možnost dostat se k žádaným i skutečným hodnotám. Pro správné zobrazení žádaných veličin se tedy v mainwindow.cpp musí provést pomocí metody connect registrace na signál SendGetReqValServo, který se emituje z regulation.cpp a vytvořit slot ShowReqValServo v mainwindow.cpp, který se zavolá po emitaci signálu. Parametry jsou potom požadované hodnoty.

```
connect(
      udp->regulator,
      SIGNAL(SendGetReqValServo(float, float, float, float, float, float, int, int)),
      this,
      SLOT(ShowReqValServo(float, float, float, float, float, float, int, int))
);
```

V metodě ShowReqValServo probíhá zobrazení žádaných veličin pomocí metody setValue, která se volá pro vybraný prvek, kde se má veličina zobrazit. Ukázka následujícího kódu pak zobrazuje požadovanou hodnotu proudu.

```
ui->dsb_manContrPar1->setValue(current1);
```

Pro zobrazení skutečných veličin je postup stejný s tím rozdílem, že probíhá registrace na signál SendActValServo, který se opět emituje z regulation.cpp a volá vytvořený slot ShowActValServo. Nastavení hodnot probíhá opět pomocí metody setValue.

6.1.3 Implementace řízení polohy kuličky

Pro řízení polohy kuličky je nutné v uživatelském rozhraní aplikace jako typ řízené veličiny vybrat možnost „Pozice kuličky“ a to způsobí, že se proměnná, kde je uložena informace o aktuálním typu manuálního řízení aktualizuje na novou hodnotu a touto hodnotou bude 1 (1 určuje, že se jedná o řízení pozice kuličky). Aktualizace této proměnné probíhá v metodě on_rb_manContrPos_clicked a aktualizovaná proměnná se jmenuje manualControlType.

Jakmile je nastaven typ řízení veličiny, tak je možné kliknout na tlačítko ZAPNOUT, a to způsobí zavolání metody on_pb_ManualControl_clicked. V této metodě se provede kontrola, o jaký typ manuálního řízení se jedná a jelikož byla proměnná manualControlType aktualizována na hodnotu 1, tak se vykoná blok uvnitř této podmínky pro řízení polohy kuličky. Vnitřek podmínky zde není vypsaný celý, kvůli kratšímu zobrazení kódu, ale dále jsou popsány klíčové funkcionality, které probíhají uvnitř této podmínky.

```
if(manualControlType == 1) {
       // implementace vnitřku podmínky pro řízení polohy kuličky
} else {
       // implementace vnitřku podmínky pro jiné typy řízení
}
```

Uvnitř probíhá nastavení proměnné vybraného typu řízení v qlabelmouseevent.cpp a výpočet pro vycentrování kurzoru do středu rámce pro kameru, jelikož to bude výchozí poloha.

```
ui->lb_frame->manualControlType = manualControlType;

double xCenter = this->pos().x() + ui->lb_frame->pos().x() + ui->lb_frame->width()/2;
double yCenter = this->pos().y() + ui->lb_frame->pos().y() + ui->lb_frame->height()/2;
```

Dále probíhá aktualizace proměnné manualControlFixPos, jenž slouží k přesunutí kurzoru do středu v qlabelmouseevent.cpp a tím se i manuálně nastaví pozice kurzoru do středu rámce pro kameru.

```
ui->lb_frame->manualControlFixPos = QPoint(xCenter, yCenter);
cursor.setPos(ui->lb_frame->manualControlFixPos);
```

Aktuálně je vše potřebné nastaveno a může se zvolit nová pozice kuličky na nakloněné rovině. To lze uskutečnit tak, že se v rámci pro kameru klikne tlačítkem, které je na gyroskopickém ovladači (simulace kliku na levé tlačítko na myši), na novou pozici, kde se má kulička na nakloněné rovině vyskytovat. Po tomto kliku je v qlabelmouseevent.cpp zavolána metoda mousePressEvent, která právě reaguje na stisknutí tlačítka a zároveň je provedena kontrola, jestli se jedná o levé tlačítko na myši. Dále je uvnitř provedena kontrola, zda se jedná o řízení polohy kuličky a výpočet pro novou hodnotu proměnné mousePos, která určuje, kde by měla kulička být. Tato nová pozice je pomocí funkcionality Signály & Sloty (angl. Signals & Slots) odeslána do cameraworker.cpp, kde dojde jako nová žádaná poloha kuličky a pomocí skutečné polohy kuličky bude postupně generovat nové žádané hodnoty pro nižší smyčky, což je popsáno v teoretické části. Vizuálně je tato celá funkcionalita zobrazena na Obrázek 26.

```
if(event->button() == Qt::LeftButton) {
        if(manualControlType == 1){
                mousePos = event->pos();

                mousePos.setX(mousePos.x() * ((float)qLableRes.x() / this->width()));
                mousePos.setY(mousePos.y() * ((float)qLableRes.y() / this->height()));

                emit SendMousePressMoveLeftPositionBallContr(mousePos);
        }
}
```

Obrázek 26. Manuální řízení – pozice kuličky

6.1.4 Implementace řízení náklonu roviny

Pro řízení náklonu roviny je nutné v uživatelském rozhraní aplikace jako typ řízené veličiny vybrat možnost „Náklon roviny“ a to způsobí, že se proměnná manualControlType aktualizuje na hodnotu 2 (2 určuje, že se jedná o řízení náklonu roviny). Aktualizace této proměnné probíhá v on_rb_manContrPos_clicked. Jelikož se nyní jedná o regulaci plošiny, nikoliv kuličky, tak v případě, že je regulace kuličky stále zapnuta se provede její vypnutí. To je docíleno pomocí zavolání metody SetBallRegulation v cameraworker.cpp.

```
if(manualControl && ballRegulation) {
        ballRegulation = false;
        worker->SetBallRegulation(ballRegulation);
}
```

Jakmile je nastaven typ řízení veličiny, tak je možné kliknout na tlačítko ZAPNOUT, což způsobí zavolání metody on_pb_ManualControl_clicked. Uvnitř této metody probíhá kontrola, jestli se jedná o řízení polohy kuličky, ale jelikož byla proměnná manualControlType aktualizována na hodnotu 2, vybere se větev else. Opět je zde jen zkrácená verze kódu, kvůli kratšímu zobrazení kódu. Uvnitř else bloku se provede nastavení proměnné vybraného typu řízení v qlabelmouseevent.cpp a výpočet pro vycentrování

kurzoru do středu rámce pro kameru. Celý proces je velmi podobný tomu v předešlé podkapitole.

```
if(manualControlType == 1) {
	// implementace vnitřku podmínky pro řízení polohy kuličky
} else {
	// implementace vnitřku podmínky pro jiné typy řízení
}
```

Rozdíl oproti řízení polohy kuličky je v tom, že nyní neprobíhá nastavení žádané hodnoty kliknutím do rámce pro kameru, ale stisknutím tlačítka na gyroskopickém ovladači (simulace kliku na levé tlačítko na myši) a pohybem gyroskopického ovladače. Celý tento proces je v qlabelmouseevent.cpp odchytáván v metodě mouseMoveEvent, kterým je volána při každém pohybu gyroskopického ovladače. Protože by se generovalo velké množství žádaných hodnot, tak je na začátku podmínka, díky které se vygeneruje nová žádaná hodnota každých 30ms. Pro jednoduchost tu není vypsán celý vnitřek podmínky.

```
if(timer.elapsed() > 30) {
	// implementace vnitřku podmínky
}
```

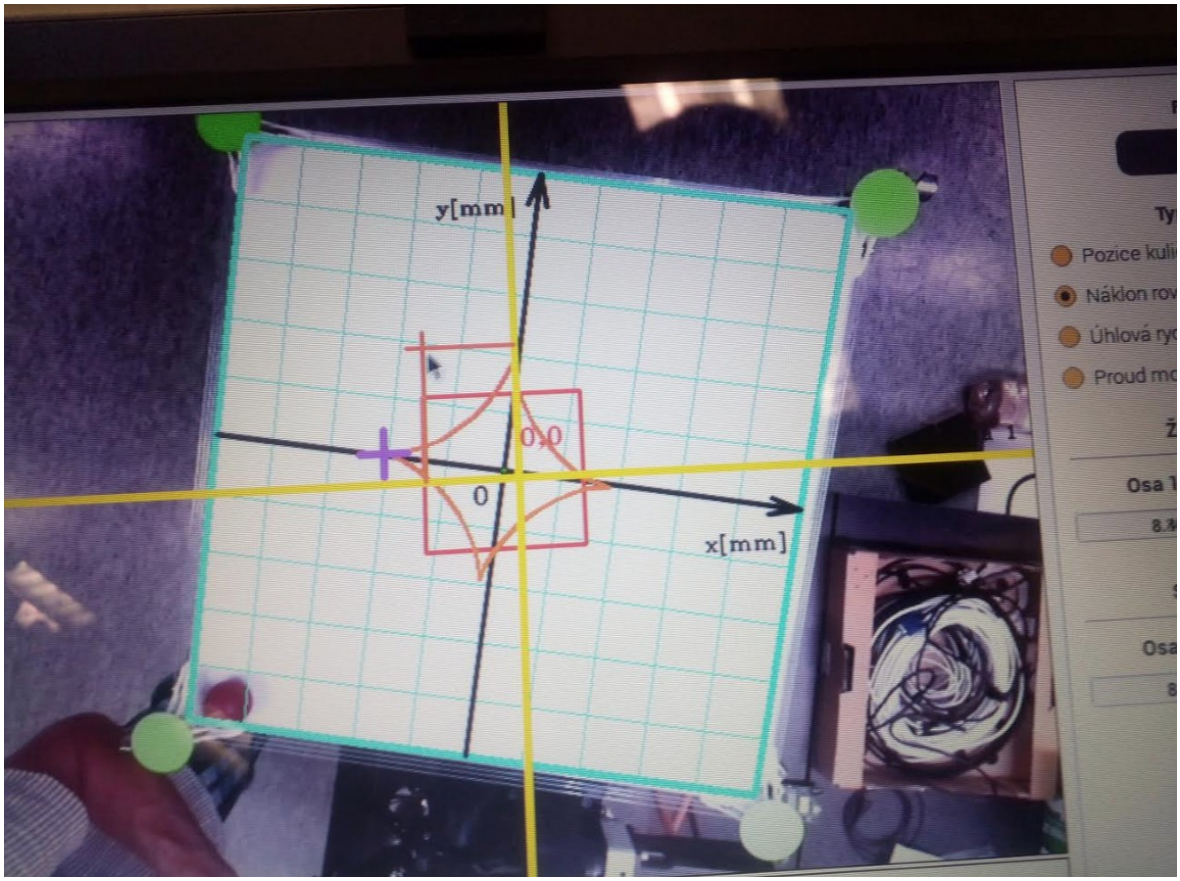
Uvnitř této podmínky probíhá kontrola, o jaký typ řízení se jedná (v aktuálním případě jde o řízení náklonu roviny) a následný výpočet žádané hodnoty pro regulátor. Důležité je také zmínit, že zde probíhá kontrola, aby nebyla překročena oblast povolená pro regulaci. Pokud je tato oblast překročena je pozice uměle zastavena na hranici. Ukázka pro hlídání oblasti pro souřadnici x je v následujícím kódu. Obdobně je pak vytvořena kontrola pro další hraniční podmínky.

```
if(mousePos.x() > this->size().width()-1) {
	mousePos.setX(this->size().width()-1);
	double maxX = manualControlFixPos.x() + this->size().width()/2;
	cursor().setPos(QPoint(maxX, cursor().pos().y()));
}
```

Žádaná hodnota pro náklon roviny je pak vypočítána a odeslána pomocí funkcionality Signály & Sloty (angl. Signals & Slots) do regulation.cpp. Žádaná hodnota je předávána ve stupních. V regulation.cpp jsou pak postupně generovány žádané hodnoty pro nižší smyčky. Vizuálně je tato funkcionalita zobrazena na Obrázek 27.

```
case 2:
manualContrDesirVal = QPointF(-(mousePos.x()-qLableRes.x()/2)/15.0,-(mousePos.y()-
qLableRes.y()/2)/10.0);


emit SendMousePressMoveLeftPositionPlateContr(manualContrDesirVal);
```



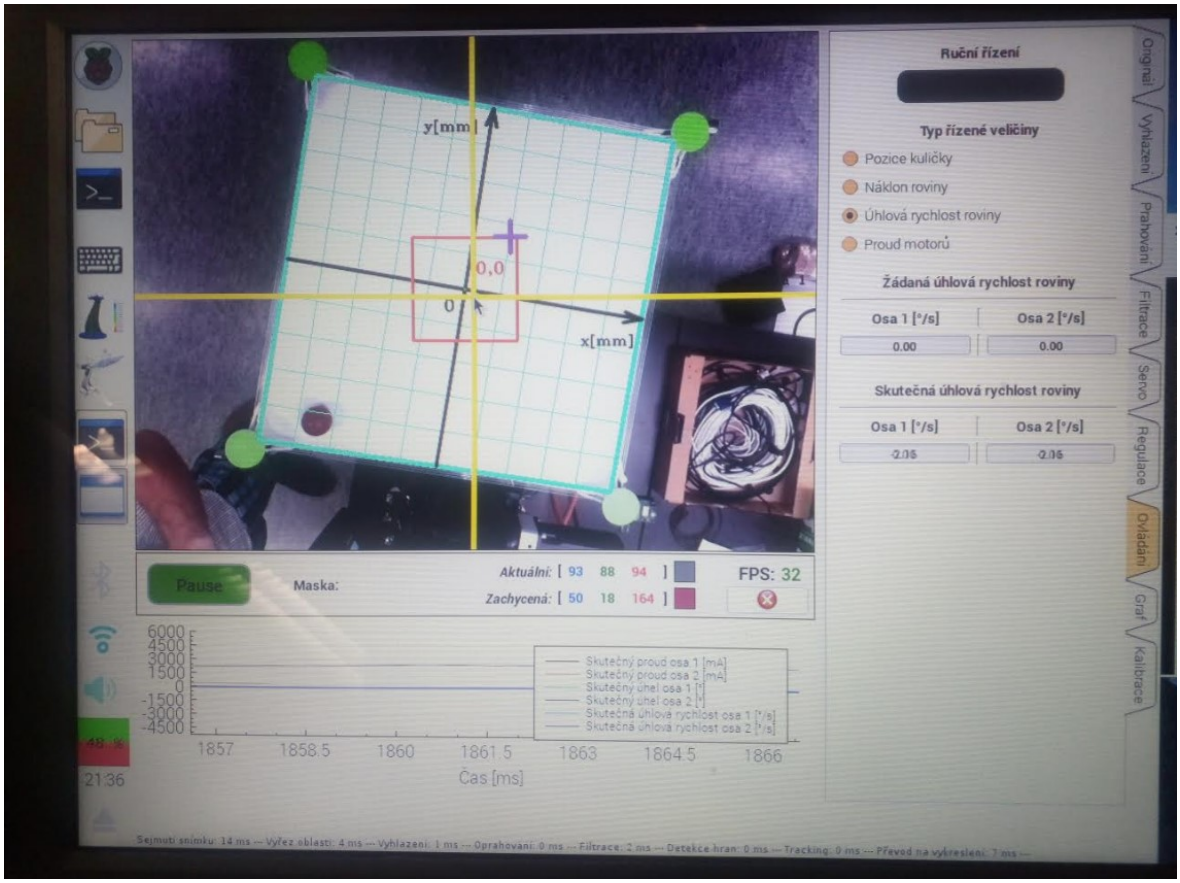
Obrázek 27. Manuální řízení – náklon roviny

6.1.5 Implementace řízení úhlové rychlosti roviny

Pro řízení náklonu roviny je nutné v uživatelském rozhraní aplikace jako typ řízené veličiny vybrat možnost „Úhlová rychlost roviny“ a to způsobí, že se proměnná manualControlType aktualizuje na hodnotu 3 (3 určuje, že se jedná o řízení úhlové rychlosti roviny). Další postup je totožný jako postup popsaný v podkapitole o řízení náklonu roviny, a proto zde není celý znovu opakován. Rozdíl je hlavně ve výpočtu žádané hodnoty pro regulaci úhlové rychlosti roviny, která se do regulation.cpp odesílá jako ve stupních za sekundu a odesílání probíhá opět pomocí funkcionality Signály & Sloty (angl. Signals & Slots) a v regulation.cpp jsou pak postupně generovány žádané hodnoty pro nižší smyčky a v tomto případě už se jedná o druhou nejnižší smyčku řízení hned za řízením proudu. Vizuálně je tato funkcionalita zobrazena na Obrázek 28.

```
case 3:
manualContrDesirVal = QPointF(-(mousePos.x()-qLableRes.x()/2)/5.0, -(mousePos.y()-
qLableRes.y()/2)/3.0);

emit SendMousePressMoveLeftPositionPlateContr(manualContrDesirVal);
```



Obrázek 28. Manuální řízení – úhlová rychlost náklonu roviny

6.1.6 Implementace řízení proudu motorů

Pro řízení proudu motorů je nutné v uživatelském rozhraní aplikace jako typ řízené veličiny vybrat možnost „Proud motorů“ a to způsobí, že se proměnná manualControlType aktualizuje na hodnotu 4 (4 určuje, že se jedná o řízení proudů motorů). Další postup je totožný jako postup popsaný v podkapitole o řízení náklonu roviny, a proto zde není celý znovu opakován. Rozdíl je tedy opět ve výpočtu žádané hodnoty pro regulaci proudů motorů, která se do regulation.cpp odesílá v miliampérech a odesílání probíhá opět pomocí funkcionality Signály & Sloty (angl. Signals & Slots). V tomto případě se jedná o nejnižší smyčku řízení hned nad servozesilovačem.

```
case 4:
manualContrDesirVal = QPointF(-(mousePos.x()-qLableRes.x()/2)*10, -(mousePos.y()-
qLableRes.y()/2)*10)

emit SendMousePressMoveLeftPositionPlateContr(manualContrDesirVal);
```

6.1.7 Přerušení manuálního řízení

Jelikož řízení a generování žádaných hodnot probíhá (kromě řízení pozice kuličky) při stisknutém tlačítku, tak bylo nutné implementovat funkcionalitu pro zastavení řízení. Tato

funkcionalita je implementována v qlabelmouseevent.cpp v metodě mouseReleaseEvent, která je zavolána po uvolnění tlačítka. Uvnitř metody je pak provedena kontrola, zda je zapnuto manuální řízení pro náklon roviny, úhlovou rychlost roviny nebo proudu motorů. Pokud je vybrán jeden ze zmíněných typů řízení je pomocí funkcionality Signály & Sloty (angl. Signals & Slots) odeslána vynulovaná pozice do regulation.cpp, což zajistí vyrovnání plošiny do výchozího stavu.

```
if(manualControlType > 1) {
        emit SendMousePressMoveLeftPositionPlateContr(QPointF(0, 0));
        manualControlType = 0;
        emit SendMouseReleaseLeftPosition();
}
```

6.2 Implementace vizualizace veličin

Implementace grafické vizualizace veličin popisuje implementaci grafů do uživatelského rozhraní aplikace. Jak už bylo zmíněno, tak je využita knihovna QCustomPlot, kvůli starší verzi Qt a tudíž nemožnosti využití grafů Qt Charts, jenž jsou součást Qt v novějších verzích. Nemožnost využití Qt Charts se nakonec stalo překážkou pro grafickou interpretaci snímaných veličin, protože QCustomPlot při vyšších periodách vzorkování nezvládá vykreslovat grafy v reálném čase, a proto je v závěru popsán tento problém a navržen způsob řešení.

6.2.1 Popis grafického rozhraní pro graf

Grafické rozhraní pro graf je umístěno v nové záložce Graf, která je přidána stejným způsobem, jako je popsáno v předešlé kapitole v části Přidání nové záložky a zároveň ovládací prvky pro graf jsou přidány obdobným způsobem, jaký je popsaný v Přidání ovládacích prvků a implementace zobrazování veličin. Výsledné grafické rozhraní je zobrazeno na Obrázek 29.

Obrázek 29. Grafické rozhraní pro ovládání grafů

Jelikož se do aplikace přidávají dva grafy – spodní a hlavní graf, je zde možnost nastavení pro oba dva grafy. Více specifičtější nastavení pak platí jen pro graf hlavní, který se zobrazuje jen v záložce Graf. V grafickém rozhraní pro graf je tedy možné nastavit jaké průběhy se v grafech budou vyskytovat, nastavit styl vykreslování (čárové, impulsní anebo schodovité) a nastavení periody vzorkování. V nastavení pro hlavní graf je možné zvolit průběhy na ose x (čas, nebo poloha) a spouštění (angl. triggering). V sekci ovládání jsou pak ovládací tlačítka pro hlavní graf.

6.2.2 Import knihovny QCustomPlot

Knihovnu QCustomPlot je možné stáhnout z oficiálních webových stránek knihovny qcustomplot.com/index.php/download a následně vložit do projektu s aplikací pro reálný model. Knihovna je vložena do adresáře external. Důležité je rozlišit sadu souborů, které

mají být kompilovány. Toho lze docílit v souboru ImageProcessing.pro, kde probíhá různé nastavování věcí pro projekt.

6.2.3 Implementace grafů do grafického rozhraní

Jak již bylo zmíněno, tak se v aplikaci budou vyskytovat grafy dva a to hlavní, který je zobrazen jen v záložce Graf a spodní graf, který je zobrazen pořád. Pro přidání obou grafů musí proběhnout rozšíření v mainwindow.ui, tedy v grafické nadstavbě nad tímto souborem. Přidání grafů proběhne tak, že se v grafické nadstavbě vyhledá klíčové slovo Widget a vloží do uživatelského rozhraní aplikace na vybrané místo. U Widgetu, kde bude zobrazen hlavní graf se identifikátor nastaví jako mainGraph a u Widgetu, kde bude spodní graf se nastaví identifikátor jako bottomGraph. Důležité je udělat změnu obou Widgetů na tzv. vlastní Widgety, které budou podporovat QCustomPlot. Toho lze docílit pravým klikem na Widget a zvolení Propagovat do → QCustomPlot (angl. Promote to → QCustomPlot).

Pro práci s knihovnou QCustomPlot je vytvořena nová třída graphworker. Jelikož hlavní i spodní graf mají spoustu věcí společných, tak je v jednotlivých metodách v graphworkeru.cpp implementována hlavní logika a v mainwindow.cpp jsou pak tyto metody volány.

Pro hlavní (zároveň i spodní) graf je vytvořena proměnná v hlavičkovém souboru mainwindow.h a v těchto proměnných je uložena instance třídy graphworker pro vybraný graf. Proměnná pro hlavní graf se jmenuje graphMainWorker a proměnná pro spodní graf graphBottomWorker. V následující ukázce kódu je tedy vytvořena nová instance třídy GraphWorker (předávají se tři argumenty – rodič, tzv. vlastní Widget s QCustomPlot a boolean hodnota, zda se jedná o hlavní nebo spodní graf) a následně zavolána metoda add_graph, která nastaví identifikátor pro průběh a popis tohoto průběhu.

```
graphMainWorker = new GraphWorker(nullptr, ui->mainGraph, true);

graphMainWorker->add_graph(
        GraphWorker::GRAPH_TYPE_BY_INDEX::ACTUAL_CURRENT_AXIS_1,
        "Skutečný proud osa 1 [mA]"
);
```

Identifikátory se volí podle typu průběhu. V ukázkovém kódu jde o skutečný proud v ose 1 a má identifikátor ACTUAL_CURRENT_AXIS_1. Další identifikátory jsou uloženy v proměnné výčtového typu GRAPH_TYPE_BY_INDEX. Vnitřek metody add_graph obsahuje přidání nového záznamu do interního stavu, který se používá pro export veličin a přidání průběhu do grafu jenž se volá nad knihovnou QCustomPlot. Následně se nastaví

jméno a zavolá funkce render, která celý graf aktualizuje. Obdobně pak probíhá přidání dalších typů průběhů.

```
internal_state.append({graph_type_by_index, name, {}, {}});
graph->addGraph();
set_name(graph_type_by_index, name);
render();
```

Zvolení průběhů, které se mají zobrazit probíhá v uživatelském rozhraní a ve výchozím stavu jsou vybrány všechny. Pokud uživatel chce nějaký průběh skrýt, je třeba kliknout na „zaškrtávací“ tlačítko pro vybraný průběh. Například pro skrytí průběhu pro proud na ose 1 a po kliku na „zaškrtávací“ tlačítko proběhne v mainwindow.cpp zavolání metody on_graphAxisYCurrent1_clicked, která má jako parametr boolean hodnotu checked, která určuje, zda se má průběh zobrazovat nebo nikoliv. Uvnitř této metody se pak volá metoda set_visibility pro dané průběhy a rovněž grafy.

```
graphMainWorker->set_visibility(
        GraphWorker::GRAPH_TYPE_BY_INDEX::ACTUAL_CURRENT_AXIS_1,
        checked
);
```

Metoda set_visibility je implementována v graphworker.cpp. Uvnitř probíhá kontrola, zda vybraný identifikátor existuje a pokud ano, tak zavolá metoda setVisible v knihovně QCustomPlot a následně funkce render, která aktualizuje graf.

```
if (!graph->graph(graph_type_by_index)) {
        return;
}
graph->graph(graph_type_by_index)->setVisible(visibility);
render();
```

Implementace stylu vykreslování průběhů v grafu probíhá velmi podobně. Po výběru nového stylu vykreslování se v mainwindow.cpp zavolá metoda on_graphLineStyle_currentIndexChanged, která má parametr index. Podle tohoto indexu je následně rozpoznáno, jestli se jedná o styl vykreslování čárový, impulsní anebo schodovitý. Podle indexu se tedy vybere daný typ a zavolá metoda set_line_style, která je implementována v graphworkeru.cpp.

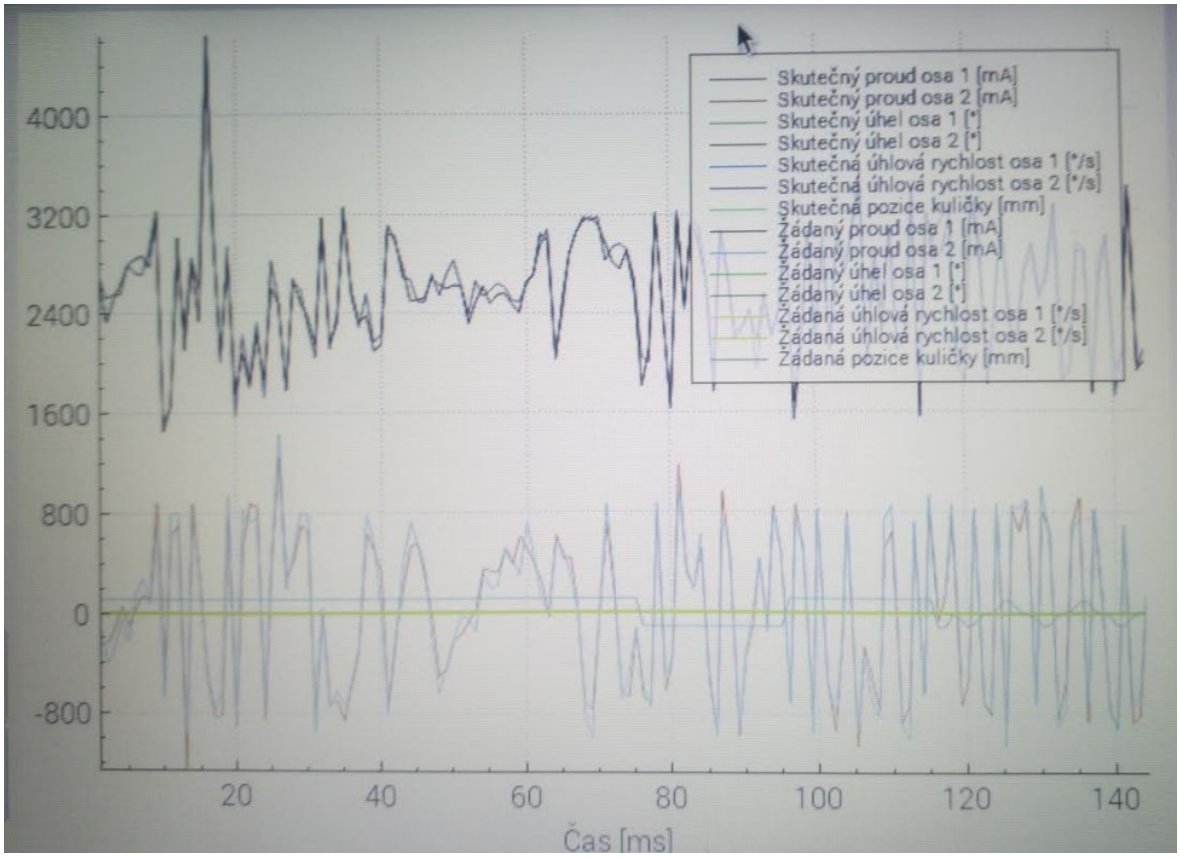
```
graphMainWorker->set_line_style(
        GraphWorker::GRAPH_TYPE_BY_INDEX::ACTUAL_CURRENT_AXIS_1,
        line_style_list[index]
);
```

Metoda set_line_style poté jen zavolá metodu setLineStyle v knihovně QCustomPlot pro vybraný indentifikátor a zavolá metodu render, která aktualizuje graf.

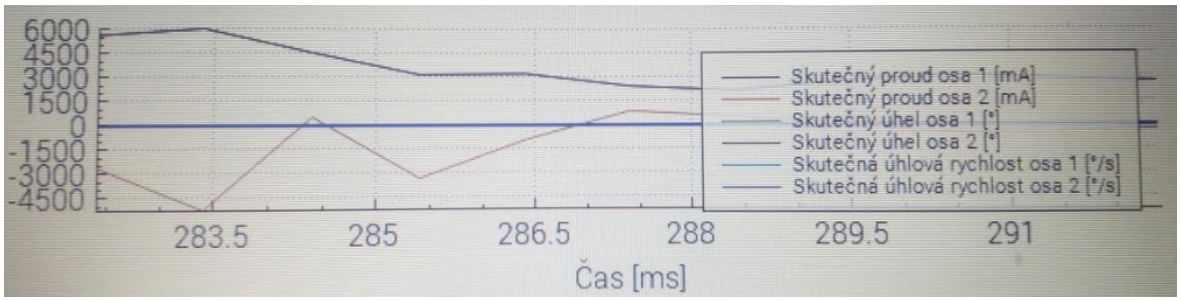
```
graph->graph(graph_type_by_index)->setLineStyle(style);
render();
```

Dále jsou ve spodu uživatelského rozhraní pro graf tlačítka Zapnout a Vypnout, které buď zapínají nebo vypínají aktualizaci veličin v grafu. Funkcionalita je dosažena voláním metody v graphworker.cpp enable_updating (případně disable_updating).

V této kapitole byly popsány vybrané prvky pro nastavování grafů. Třída graphworker.cpp obsahuje i další možnosti nastavení grafů, jejichž implementaci je možné nalézt v příloze. Hlavní graf je zobrazen na Obrázek 30 a spodní graf je zobrazen na Obrázek 31.



Obrázek 30. Hlavní graf



Obrázek 31. Spodní graf

6.2.4 Nastavení periody vzorkování

Nastavení periody vzorkování probíhá v záložce Graf a platí pro oba dva grafy. Po definování nové periody vzorkování a kliku na tlačítko Nastavit periodu vzorkování proběhne zavolání metody on_graphPeriodButton_clicked v niž se vyresetují dosavadní průběhy podle typů průběhů v obou grafech, získá se perioda vzorkování a zavolá se metoda set_graphs_period, která je implementována ve třídě myudp.cpp kde probíhá komunikace mezi servozesilovačem a hlavním počítačem. Po nastavení nové periody vzorkování pak pomocí funkcionality Signály & Sloty (angl. Signals & Slots) budou chodit aktualizovaná data podle periody vzorkování do třídy mainwindow.cpp, kde můžou být dále zpracovávána. Zároveň je v metodě aktualizována proměnná graphPeriod, se kterou mainwindow pracuje při aktualizaci veličin v grafech.

```
graphMainWorker->reset_data(graph_type_by_index);
float period = ui->graphPeriod->value();
graphPeriod = period;
udp->set_graphs_period(period);
```

6.2.5 Nastavení spouštění

Nastavení spouštění (angl. triggering) probíhá v záložce Graf a platí pouze pro hlavní graf. V uživatelském rozhraní jsou definovány dva vstupy, kde první vstup určuje minimální hodnotu pro vykreslení průběhů a druhý vstup pak definuje maximální hodnotu pro vykreslení průběhu. Po změně těchto vstupů se zavolá v mainwindow.cpp buď metoda on_mainGraphTriggeringFrom_valueChanged, nebo metoda on_mainGraphTriggeringTo_valueChanged. Jejich funkcionalita je velmi podobná. Zavolají metodu v graphworker.cpp, která povolí spouštění (angl. triggering) a následně nastaví minimální nebo maximální hodnotu.

```
graphMainWorker->enable_triggering();
graphMainWorker->set_x_triggering_max(x_triggering_max_value);
```

Metoda enable_triggering změní v graphworkeru.cpp proměnnou is_triggering_enabled na boolean hodnotu true a metoda set_x_triggering_max (podobně pak set_x_triggering_min) změní v graphworkeru.cpp proměnnou x_triggering_max (obdobně proměnnou x_triggering_min) na novou hodnotu, se kterou spouštění pracuje.

Proměnné x_triggering_min a set_x_triggering_max jsou při vykreslování průběhů používány pro kontrolu, zda se má nová hodnota vykreslit, nebo nikoliv.

6.2.6 Aktualizace veličin

Aktualizace veličin grafu probíhá v mainwindow.cpp v metodě set_graphs_data. Tato metoda je pomocí funkcionality Signály & Sloty (angl. Signals & Slots) volána z myupd.cpp, kde jsou pomocí signálu SendDataForGraphs odeslány veličiny a do metody set_graphs_data jsou předány veličiny (žádané i aktuální), které oba dva grafy potřebují pro svou aktualizaci. Následující ukázka kódu ukazuje připojení na signál v mainwindow.cpp a následné volání slotu pro aktualizaci veličin v grafech.

```
connect(
      udp,
      SIGNAL(SendDataForGraphs(/*data pro graf*/)),
      this,
      SLOT(set_graphs_data(/*data pro graf*/))
);
```

V Metodě set_graphs_data probíhá kontrola, zda aktuální perioda vzorkování není příliš vysoká (tzn. aktualizace veličin probíhá rychle). Pokud je perioda vzorkování příliš vysoká, tak se ukládají data do pole. Jakmile toto pole dosáhne určité délky, tak se teprve aktualizují data v grafech. Tato funkcionalita je vytvořena kvůli optimalizaci vykreslování veličin v grafu.

```
if (graphPeriod > minimumGraphPeriodForSingleData) {
        set_graphs_single_data(/* data pro graf */);
} else {
        set_graphs_buffer_data((/* data pro graf */);
}
```

V metodě set_graphs_single_data proběhne zavolání metody set_data, která je implementována v graphworker.cpp. V případě set_graphs_buffer_data jsou data ukládána do zmíněného pole a pokud pole dosáhne předem definované délky, tak je zavolána metoda set_data, která je implementována v graphworker.cpp. Důležité je toto pole v případě dosažení definované délky vyčistit zavoláním metody clear_data_buffer.

```
if (timeBufferBottomGraph->size() == bufferSizeLimit) {
        graphBottomWorker->set_data(
                GraphWorker::GRAPH_TYPE_BY_INDEX::ACTUAL_CURRENT_AXIS_1,
                timeBufferBottomGraph,
                aCurrent1Buffer
        );
        clear_data_buffer();
} else {
        timeBufferBottomGraph->append(bottom_graph_time);
        aCurrent1Buffer->append(aCurrent1);
}
```

Implementace metody set_data využívá tzv. přetěžování, což je vlastnost C++. Jde o to, že metoda může mít stejný název, ale chovat se jinak v zavilosti na typech parametrů. V

případě, že se metoda set_data volá s parametry, které jsou uloženy v poli, tak uvnitř graphworker.cpp probíhá vyčtení dat z toho pole a zavolání interní metody set_data, která jako parametry už má jednotlivé proměnné, kde na začátku této metody je kontrola, zda se mají data v grafu aktualizovat (například kvůli spouštění). Dále probíhá kontrola, zda se má aktualizovat interní stav (používá se jen u hlavního grafu pro export veličin) a následné zavolání metody addData, která je implementována v knihovně QCustomPlot. Pokud je zapnuto tzv. posouvání grafu, kdy je zobrazeno jenom okno určitých hodnot, tak je volána metoda setRange v knihovně QCustomPlot. Dále probíhá volání metody replot se speciálním parametrem, který zajistí větší optimalizaci při vykreslování nových dat.

```
if (!should_graph_set_new_data(graph_type_by_index, x_value, y_value)) {
        return;
}
if (main_graph) {
        update_internal_state(graph_type_by_index, x_value, y_value);
}
graph->graph(graph_type_by_index)->addData(x_value, y_value);
graph->rescaleAxes(true);
if (is_shifting_enabled) {
        graph->xAxis->setRange(x_value, shifting_range, Qt::AlignRight);
}
graph->replot(QCustomPlot::RefreshPriority::rpQueued);
```

6.2.7 Problém aktualizace dat a možné řešení

Při implementaci grafů do reálného modelu bylo zjištěno, že při vyšší periodě vzorkování (100ms) začne mít knihovna QCustomPlot problém s aktualizací dat a v celém uživatelském rozhraní se začnou snižovat FPS, až se dostanou na hranici 1 FPS. Celé uživatelské rozhraní se pak stane nepoužitelným a aplikace se musí vypnout.

Pro řešení problému byly implementovány doporučení, které nabízí přímo QCustomPlot ohledně zlepšení výkonu, ale problém s aktualizací dat to bohužel nevyřešilo. Mezi tyto doporučení patří nevykreslovat grafy, které mají větší tloušťku hran než 1 mezi jednotlivými body. Pro aktualizaci dat se nepoužívá metoda setData, ale na doporučení metoda addData. Dále se zmenšila možnost viditelných dat na menší rámec a je tedy v grafu možné zobrazit maximálně 20 datových bodů. Při vytváření grafů je také vypnuto použití tzv. antialiasingu, což by podle doporučení mělo pomoci při vykreslování. [22]

Dalším možným způsobem řešení, které bylo vyzkoušeno, je ukládání dat pro graf do zásobníku a následná aktualizace. Bohužel ani tento způsob optimalizace nepřinesl zlepšení výkonu.

Všechny výše zmíněné postupy nepomohly a možné řešení by mohlo tedy být na reálném modelu aktualizovat framework Qt na verzi, kde bude možné využívat Qt Charts. Qt Charts je podporovaná součást frameworku Qt, což by i z hlediska údržby bylo do budoucna lepší. Použitím Qt Charts by se měl vyřešit problém s aktualizací dat, a tím i vyřešit problém s výkonem. Dalším možným řešením je pak využití výkonnějšího řídícího počítače, než je Raspberry Pi.

6.3 Implementace exportu veličin

V poslední části je vysvětlena funkcionalita exportu veličin v aplikaci pro reálný model. Pro tuto funkcionalitu je vytvořena nová třída csvworker, která zajišťuje převod veličin z grafu do formátu CSV.

6.3.1 Získání dat z grafu

Jak již bylo zmíněno v části Implementace vizualizace veličin, v třídě graphworker existuje interní stav, kde se ukládají data pro export. Tento interní stav je struktura GRAPH_INTERNAL_STATE, která obsahuje identifikátor grafu, jméno grafu a veličiny pro daný identifikátor.

```
struct GRAPH_INTERNAL_STATE {
        GRAPH_TYPE_BY_INDEX graph_type_by_index;
        QString name;
        QVector<double> x_values;
        QVector<double> y_values;
};
```

V graphworker.cpp je pak implementována metoda get_internal_state, která vrací tento interní stav.

6.3.2 Export veličin

Třída csvworker má svoji strukturu, z které umožňuje exportovat data. Tato struktura obsahuje jméno exportovaných dat a veličiny pro export.

```
struct CSV_DATA {
        QString name;
        QVector<double> x_values;
        QVector<double> y_values;
};
```

V grafickém rozhraní pro aplikaci v záložce Graf je tlačítko Export do CSV. Po kliknutí na tlačítko je v mainwindow.cpp zavolána metoda on_graphExport_clicked. V této metodě probíhá získání dat z graphworker.cpp, které se následně uloží do struktury CSV_DATA a zavolá se metoda save_to_csv, která je implementována v csvworker.cpp.

```
QList<GraphWorker::GRAPH_INTERNAL_STATE> main_graph_internal_state =
graphMainWorker->get_internal_state();
QList<CsvWorker::CSV_DATA> csv_data;

for (int i = 0; i < main_graph_internal_state.size(); i++) {
        csv_data.append({
                main_graph_internal_state[i].name,
                main_graph_internal_state[i].x_values,
                main_graph_internal_state[i].y_values
        });
}
```

```
csvWorker->save_to_csv(csv_data);
```

V metodě save_to_csv probíhá výběr složky, kde se má nový CSV soubor uložit. Tato funkcionalita je zprostředkována pomocí metody QFileDialog::getExistingDirectory. Následný zápis dat do CSV souboru dodržuje správný formát dat, který je popsány v teoretické části. Celou metodu pro export dat je možné nalézt v příloze.

ZÁVĚR

Při implementaci gyroskopického ovladače do reálného modelu bylo zjištěno, že nelze využívat data přímo z gyroskopu (případně akcelerometru) v ovladači, protože gyroskopický ovladač poskytuje pouze data o pozici myši. Přepočet těchto dat z gyroskopu a akcelerometru na pozici myši je vysvětlen v teoretické části. Aktuálně je tedy nutné využívat pozici myši na generování žádaných veličin.

Využití gyroskopického ovladače se tedy ukázalo jako možné, i přes původní potíže s implementací (kvůli tomu, že jsou dostupná pouze data o pozici myši). Pro akademické účely je tento typ gyroskopického ovládaní dostačující a lze na něm ukázat další typ možného ovládání, tedy generování žádaných trajektorií, robotů/robotických zařízení.

Dále bylo při implementaci zjištěno, že v aplikaci pro reálný model není aktuální verze Qt a tím pádem nelze využít Qt Charts pro tvorbu grafů a bylo tedy nutné zvolit externí knihovnu. Z tohoto důvodu navrhuji aktualizaci Qt pro zlepšení výkonu u grafů, což je popsáno v praktické části a především je pro údržbu aplikace lepší do budoucna nevyužívat externí knihovnu pro tvorbu grafů, která přidává na komplexitě celého softwaru.

V softwaru pro reálný model „Kulička na nakloněné rovině“ navrhuji použít pro vývoj dalšího softwaru jeden z běžných způsobů verzování kódu (například Git). Verzování kódu je velmi důležité pro tvůrce softwaru, protože se kdykoliv může vrátit k naposledy funkční verzi a zároveň usnadní práci budoucím kolegům, kteří budou spolupracovat na vývoji softwaru pro reálný model. Další možné zlepšení softwaru by mohlo být využití techniky softwarového testování, které by hlavně u klíčových funkcí softwaru mohlo mít z hlediska údržby velký přínos.

SEZNAM POUŽITÉ LITERATURY

[1] FRADEN, Jacob. Handbook of modern sensors: physics, designs, and applications. 3rd ed. New York: Springer, c2004. ISBN 978-0387007502.

[2] How a gyroscope works. Accs.net [online]. 1997, 1999, 1-4 [cit. 2020-08-02]. Dostupné z: http://www.gyroscopes.org/how%5Chagwa4.pdf.

[3] Modelování inerciálních snímačů [online]. Brno,2013 [cit. 2020-08-02]. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=64301. Diplomová práce. Vysoké učení technické v Brně.

[4] Návrh a identifikace rozšířeného modelu MEMS gyroskopu [online]. Brno, 2015 [cit. 2020-08-02]. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=110963. Disertační práce. Vysoké učení technické v Brně.

[5] Integrované MEMS GYROSKOPY. Automatizace.hw.cz [online]. Praha: automatizace.hw.cz, 2009 [cit. 2020-08-02]. Dostupné z: https://automatizace.hw.cz/integrovane-mems-gyroskopy.

[6] Quadrocopter - stabilizace pomocí inerciálních snímačů [online]. Brno, 2011 [cit. 2020-08-02]. Dostupné z: https://www.vutbr.cz/www_base/zav_prace_soubor_verejne.php?file_id=38682. Diplomová práce. Vysoké učení technické v Brně.

[7] Inerciální měřicí jednotka, zpracování dat ze senzorů. Radiolokační a radionavigační systémy [online]. 2013, 9 [cit. 2020-07-25]. Dostupné z: http://www.urel.feec.vutbr.cz/~sebestaj/MRAR/MRAR_L_CZ02.pdf.

[8] ZÁTOPEK, J. Moderní řízení pohybových stavů mechanické soustavy průmyslového robota prostřednictvím elektromechanických akčních členů. [cit. 2020-07-15]. FAI UTB ve Zlíně, 2018.

[9] The Official Raspberry Pi Beginner's Guide: How to use your new compute. 3rd. United Kingdom: Raspberry Pi Trading, 2018. ISBN 978-1912047581.

[10] PRATA, Stephen. Mistrovství v C++. 4., aktualiz. vyd. Přeložil Boris SOKOL. Brno: Computer Press, 2013. Bestseller (Computer Press). ISBN 978-80-251-3828-1.

[11] Qt Documentation. Doc.qt.io [online]. Helsinki: Qt Documentation, 2020 [cit. 2020-08-02]. Dostupné z: https://doc.qt.io/.

[12] QCustomPlot. Qcustomplot [online]. Birkenallee: qcustomplot, 2018 [cit. 2020-08-02]. Dostupné z: https://www.qcustomplot.com/.

[13] Automatizace a automatizační technika: systémové pojetí automatizace. Brno: Computer Press, 2014. ISBN 978-80-251-3628-7.

[14] RIPKA, Pavel. Senzory a převodníky. Praha: Vydavatelství ČVUT, 2005. ISBN 80-01-03123-3.

[15] ÚŘEDNÍČEK, Z. Přednášky z předmětu Mechatronické systémy. [cit. 2020-07-15]. FAI UTB ve Zlíně, 2012.

[16] NAVRÁTIL, M. Přednášky z předmětu Instrumentace a měření. [cit. 2020-07-15]. FAI UTB ve Zlíně, 2018.

[17] Digitální osciloskop [online]. Ostrava, 2014 [cit. 2020-07-18]. Dostupné z: https://dspace.vsb.cz/bitstream/handle/10084/103833/HRA0031_FEI_N2647_2612T025_2014.pdf?sequence=1&isAllowed=y. Diplomová práce. VŠB –Technická univerzita Ostrava.

[18] HALLIDAY, David, Robert RESNICK a Jearl WALKER, DUB, Petr, ed. Fyzika. 2., přeprac. vyd. Přeložil Miroslav ČERNÝ. Brno: VUTIUM, c2013. Překlady vysokoškolských učebnic. ISBN 978-80-214-4123-1.

[19] KREJSA, M. Přednášky z předmětu Algoritmizace inženýrských výpočtů. [cit. 2020-07-15]. FAST VŠB v Ostravě, 2018.

[20] Vizualizace dat [online]. Praha, 2012 [cit. 2020-07-19]. Dostupné z: https://vskp.vse.cz/32439_vizualizace_dat. Bakalářská práce. Vysoká škola ekonomická v Praze.

[21] SHAFRANOVICH, Y. Common Format and MIME Type for CSV Files [online]. [cit. 2020-07-15].

[22] Plot Performance Improvement. Qcustomplot [online]. Birkenallee: qcustomplot, 2018 [cit. 2020-08-02]. Dostupné z: https://www.qcustomplot.com/documentation/performanceimprovement.html

SEZNAM POUŽITÝCH SYMBOLŮ A ZKRATEK

MEMS	Mikro-elektro-mechanický systém
3D	Prostor, který lze popsat třemi rozměry
CNC	Počítačem řízený obráběcí stroj
PMSM	Trojfázový synchronní motor s permanentním magnetem v rotoru
SoC	Integrovaný obvod, který zahrnuje všechny součásti počítače
CPU	Centrální procesorová jednotka
GPU	Grafický procesor
RAM	Polovodičové paměti s přímým přístupem umožňující čtení i zápis
USB	Univerzální sériová sběrnice
AV	Audio/video
CSI	Konektor pro připojení kamery
HDMI	Nekomprimovaný obrazový a zvukový signál v digitálním formátu
NOOBS	Instalátor pro operační systém
CCD	Elektronická součástka používaná pro snímání obrazové informace
CMOS	Způsob vytváření logických členů
CMOS	Způsob vytváření logických členů
UNIX	Operační systém vytvořený v Bellových laboratořích
CLI	Příkazový řádek
OS	Operační systém
LED	Elektroluminiscenční dioda, též světelná dioda
CRLF	Označení pro nový řádek v informatice
DSI	Sériové rozhraní pro display
IDE	Integrované vývojové prostředí

SEZNAM OBRÁZKŮ

SEZNAM PŘÍLOH

PŘÍLOHA P I: CD

Obsah přiloženého CD:

- Bakalářská práce v elektronické podobě.
- Soubory se softwarem pro reálný model „Kulička na nakloněné rovině“.