ZÁPADOČESKÁ UNIVERZITA V PLZNI
FAKULTA ELEKTROTECHNICKÁ

Katedra aplikované elektroniky a telekomunikací

DIPLOMOVÁ PRÁCE

Snímání polohy a pohybu pro robotickou ruku

Bc. Josef Justa **2015**

Anotace

Předkládaná diplomová práce je zaměřena na využití moderních sensorů pro řízení. Součástí práce bylo vytvoření senzorického systému se senzorickou rukavicí, kde stěžejní jsou MARG a Flex senzory. V práci je také řešena bezdrátová komunikace mezi těmito senzory a řízeným systémem robotické ruky, která je zajištěna modulem bluetooth. Součástí práce bylo vytvoření robotické ruky poháněné servomotory. Také byla vytvořena jednoduchá aplikace pro testování MARG senzorů.

Klíčová slova

MARG, IMU, gyroskop, akcelerometr, kompas, magnetometr, MEMS, flex senzor, XNA Framework, STM, řízení, servomotor, 3D tisk, rukavice, kvaternion

Abstract

Application of position sensors for control

The diploma thesis is focused on the application of modern sensors for control. Part of the thesis concerns the construction of sensory system with a sensory glove equipped with flex and MARG sensors. The communication between these sensors and controlled system of robotic hand is also designed. This communication is provided by a Bluetooth module. Subsequently, the robotic hand driven by servomotors was created. A simple application for use of the sensor system was also designed as a part of the thesis.

Key words

MARG, IMU, Accelerometer, gyroscope, magnetometer, flex sensor, MEMS, glove, quaternion, XNA Framework, STM, controlling, servomotor, 3D print

Prohlášení

Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně, s použitím odborné literatury a pramenů uvedených v seznamu, který je součástí této diplomové práce.

Dále prohlašuji, že veškerý software, použitý při řešení této diplomové práce, je legální.

.....

podpis

V Plzni dne 11.5.2015 Bc. Josef Justa

Obsah

Seznam symbolů a zkratek

MARG	Magnetic, angular rate, gravity. Označení senzorů snímajících magnetické pole, rotační rychlost a zrychlení.
q	Kvaternion
A/D	Analog to digital converter. Převodník analogového signálu na číslicový.
USART	Universal Synchronous / AsynchronousReceiever and Transmitter. Synchronní / asynchronní sériové rozhraní.
USB	Universal Serial Bus.
DPS	Deska plošných spojů.
LGA	Land Grid Array.
φ	Eulerův úhel „roll“. Úhel rotace okolo osy x.
ψ	Eulerův úhel „pitch“. Úhel rotace okolo osy y.
ϑ	Eulerův úhel „yaw“. Úhel rotace okolo osy z.
3D	Tří dimenzionální
×	Hamiltonův součin
MEMS	Micro-electro-mechanical system. Mikro-elektro-mechanický systém
DOF	Degrees of freedom – stupňů volnosti
IMU	Inetrial motion unit
AHRS	Atitude and heading reference system
F [N]	Síla
ω [$\frac{deg}{s}$]	Úhlová rychlost
v [$\frac{m}{s}$]	Translační rychlost
m [kg]	Hmotnost
Θ [$°C$]	Teplota
a [$\frac{m}{s^2}$]	Zrychlení
GMP	Geomagnetické pole

Seznam obrázků

Úvod

Předkládaná práce je zaměřena na využití moderních senzorů pro ovládání robotické ruky. Součástí této práce je tedy upravit stávající senzorickou rukavici *(angl. Wired glove)* a rozšířit ji na senzorický systém, který snímá polohu a pohyby celé paže. Dále pak vytvoření robotické ruky, která bude řízena.

Práce je rozdělena do čtyř hlavních kapitol. V první kapitole se zaměřuje na problematiku popisu a vyčíslení rotace objektů v 3D prostoru. Druhá nejrozsáhlejší kapitola této práce se věnuje senzorům pro snímání polohy a pohybů ruky. Z velké části se tato kapitola věnuje teorii MARG senzorů, ale zmíněny jsou zde i Flex senzory. Třetí kapitola pojednává o realizaci senzorického systému, komunikaci s ním a testovací aplikaci pro MARG senzory. Poslední čtvrtá kapitola popisuje 3D tisk robotické ruky a její řízení pomocí servomotorů.

Při tvorbě práce byl kladen důraz na jednoduchost popisování problematiky. Některé složitější informace o MARG senzorech zde pro potřeby práce nebyly popisovány do hloubky, ale je možné je dohledat v uvedené literatuře.

1 Natočení v inerciální soustavě

Mezi základní teoretické poznatky, ze kterých práce vychází, patří problematika vyjádření natočení v prostoru. Při znalosti natočení všech částí kostry a délku kostí je možné z natočení určit relativní polohu všech částí kostry bez potřeby znalosti absolutní polohy jejich částí v prostoru.

Toto natočení lze určit přesně ze znalosti přímé Kosinovy matice *(angl. direkt Cosine matrix - DCM)* [1. 1.], která transformuje natočení bázových vektorů v kartézských souřadnicích. Rotace pomocí DCM se často využívá v počítačových 3D software.

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R * \begin{bmatrix} x' \\ x' \\ x' \end{bmatrix} \qquad [1.\ 1.]$$

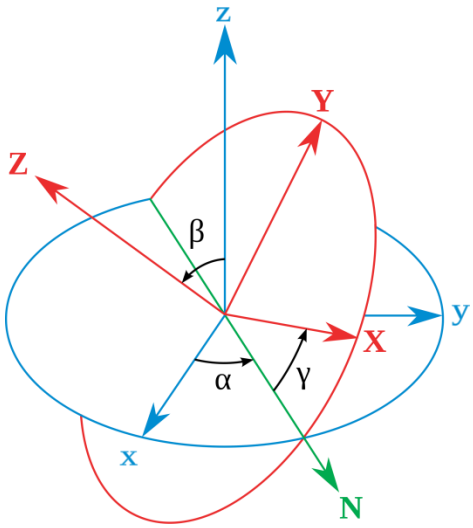
K vyjádření otočení objektu v inerciální soustavě lze přistupovat jak absolutně, tak diferenčně (teoreticky diferenciálně) tedy jako ke změně natočení. Diferenční přístup se může zdát jako méně vhodný, protože zde zavádíme integrační chybu při výsledném vyjádření rotace. Ve většině případů pro aplikaci MARG senzorů nebo jim podobným senzorům se chyba navíc nepřidává, neboť diferenční vyjádření rotace je mezikrokem při výpočtu výsledné rotace fúzním algoritmem. Záleží tedy na aplikaci, jestli bude výsledek integrován při zpracování fúzního algoritmu nebo až ve výsledné aplikaci.

Ve většině případů lze pro vyjádření rotace objektu pomocí MARG senzorů přenášet následující informace - Eulerovy úhly, kvaterniony nebo prvky Eulerových rotačních matic. V následujících podkapitolách budou tyto varianty blíže představeny a shrnuty jejich výhody a nevýhody pro využití ve spolupráci s IMU senzorem.

1.1.1 Eulerovy úhly

V R^3 prostoru lze rotaci objektu vyjádřit třemi úhly otočení oproti inerciální soustavě (Obr. 1). Historicky tuto vlastnost objevil a popsal již v polovině 18. století Leonard Euler a proto se příslušné úhly nazývají Eulerovy úhly. Tyto úhly rotací kolem inerciálních kartézských souřadnic jsou pojmenovány následovně:

- φ (α) úhel rotace kolem osy x – úhel vlastní rotace *(angl. Roll)*
- ψ (β) úhel rotace kolem osy y – precesní úhel *(angl. Pitch)*
- ϑ (γ) úhel rotace kolem osy z – nutační úhel *(angl. Yaw)*



Obr. 1 Rotace pomocí Eulerových úhlů
(http://en.wikipedia.org/wiki/Euler_angles)

Přímý přechod mezi těmito úhly a DCM neexistuje, ale je možné DCM vyjádřit skrze rotační matice. Tyto úhly je výhodné používat spíše pro lehce pochopitelný popis člověkem.

1.1.2 Eulerovy rotační matice

Rotace o Eulerovy úhly jsou ve 3D prostoru vzájemně závislé na pořadí. Rotace se tedy skládá ze tří dílčích rotací o Eulerovy úhly ve 2D. Teoreticky by nebyla tato závislost na pořadí problematická, ale reálně při zavedení chyb roste chyba s pořadím závislosti. Dalším problémem může být nelinearita této chyby v oblastech blížících se k singularitám nebo problém zvaný „gimbal lock“.

Gimbal’s označuje závěsy, na kterých visí u původních mechanických gyroskopů rotující objekt. Problém „gimbal lock“ tak nastává při vyrovnání více rotačních rovin do jedné (pro lepší pochopení doporučuji video ukázky na youtube.com).

Převod do DCM

Jak již bylo zmíněno, celková rotace je vytvořena ze tří dílčích rotací okolo hlavních os. Následující rotační matice popisují jednotlivé rotace.

$$\boldsymbol{R_x} = \begin{bmatrix} \mathbf{1} & \mathbf{0} & \mathbf{0} \\ \mathbf{0} & \boldsymbol{\cos\varphi} & \boldsymbol{\sin\varphi} \\ \mathbf{0} & \boldsymbol{-\sin\varphi} & \boldsymbol{\cos\varphi} \end{bmatrix} \boldsymbol{R_y} = \begin{bmatrix} \boldsymbol{\cos\psi} & \mathbf{0} & \boldsymbol{\sin\psi} \\ \mathbf{0} & \mathbf{1} & \mathbf{0} \\ \boldsymbol{-\sin\psi} & \mathbf{0} & \boldsymbol{\cos\psi} \end{bmatrix} \boldsymbol{R_z} = \begin{bmatrix} \boldsymbol{\cos\vartheta} & \boldsymbol{\sin\vartheta} & \mathbf{0} \\ \boldsymbol{-\sin\vartheta} & \boldsymbol{\cos\vartheta} & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \mathbf{1} \end{bmatrix} \quad [1.\ 2.]$$

Součinem těchto rotačních matic získáváme celkovou rotaci ekvivalentní k DCM. Pro ukázku je uveden výsledek pouze pro jeden možný sled rotací, tím je posloupnost podle osy Z, Y a následně X [1. 3.].

$$\boldsymbol{R = R_z * R_y * R_x =}$$
$$= \begin{bmatrix} \boldsymbol{\cos\psi * \cos\vartheta} & \boldsymbol{\cos\varphi * \sin\vartheta + \cos\vartheta * \sin\varphi * \sin\psi} & \boldsymbol{\sin\varphi * \sin\vartheta + \cos\varphi * \cos\vartheta * s} \\ \boldsymbol{-\cos\psi * \sin\vartheta} & \boldsymbol{\cos\varphi * \cos\vartheta - \sin\varphi * \sin\psi * \sin\vartheta} & \boldsymbol{\cos\vartheta * \sin\varphi - \cos\varphi * \sin\psi * s} \\ \boldsymbol{-\sin\psi} & \boldsymbol{\cos\psi * \sin\varphi} & \boldsymbol{\cos\varphi * \cos\psi} \end{bmatrix} \quad [1.\ 3.]$$

1.1.3 Kvaterniony

Kvaterniony matematicky představují rozšířená komplexní čísla (tzv. hyperkomplexní) do trojrozměrného prostoru. Skládají se z jedné reálné složky a tří imaginárních

$$a + b * i + c * j + d * k$$

, kde proměnné a, b, c a d jsou reálná čísla.

U zrodu kvaternionů *(ang. quaternions)* stál Sir William Rowan Hamilton (1805–1865). Tento muž definoval základní vazbu mezi imaginárními složkami [1. 4.].

$$i^2 = j^2 = k^2 = ijk = -1 \quad [1.\ 4.]$$

Pro úplnost matematické definice je oproti standardním komplexním číslům násobení kvaternionů nekomutativní.

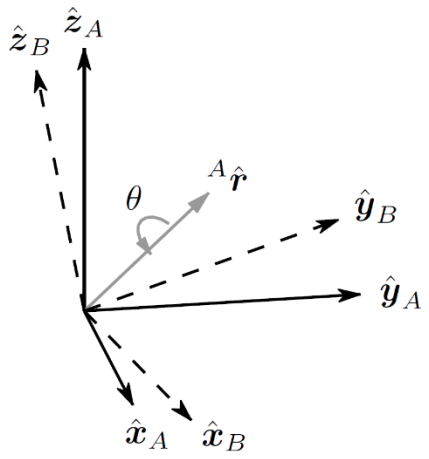
Po zrodu kvaternionů se na přelomu 19. a 20. století zvedla aktivita ve vymýšlení nových aplikací této algebry. Tento obor zpočátku slavil velký úspěch, protože se pomocí této algebry dalo vypočítat mnoho věcí, které s pomocí dosavadních metod nebylo možné řešit. Posléze se nicméně ukázalo, že pro praktické využití v reálném světě se složitost a nekomutativnost kvaternionů ukázala jako nevhodná a kvaterniony se aplikačně využívají jen v hrstce oborů.

Obecně známé obory praktického nasazení jsou pouze tři. Elektromechanika (Maxwellova kniha „*A Treatise on Electricity and Magnetism“)*, kvantová mechanika a 3D animace a geometrie (rotace) [9].

Využití kvaternionů v geometrii pro rotaci

V geometrii se kvaterniony využívají obecně k rotaci objektu v trojrozměrném prostoru. Zní to vše velmi složitě, ale stačí pouze chápat základní teorii a několik vztahů. Základní vztah pro rotaci objektu v kartézském souřadnicovém 3D prostoru [1. 5.] a její ilustrace v prostoru (Obr. 2). Vztah lze využít také pro určení kvaternionu mezi dvěma normovanými vektory ve tvaru [0 x y z].

$${}^{A}_{B}q = [q_1, q_2, q_3, q_4] = [-cos\frac{\theta}{2}, r_x * sin\frac{\theta}{2}, r_y * sin\frac{\theta}{2}, r_z * sin\frac{\theta}{2}] \quad [1.\ 5.]$$



Obr. 2 Rotace pomocí kvaternionu

V podstatě je v původním kartézském systému vytvořen pomocný vektor r (imaginární část kvaternionu) a okolo něj rotován objekt o úhel θ.

Z Obr. 2 je patrné, že vyjádření rotace pomocí kvaternionů je nejjednodušší možné a není zde žádná redundantní část v porovnání s rotací pomocí Eulerových matic.

V předešlých odstavcích byl definován vznik kvaternionu pro využití v 3D geometrii. Při práci s MARG senzory může však být nevýhodné pracovat s goniometrickými funkcemi. Kvaternion z akcelerometru a kompasu lze určit minimalizací chybové funkce.

Součin kvaternionu spojuje rotaci obou činitelů podle následujícího vztahu

$$ {}^A_C q = {}^B_C q \times {}^A_B q \qquad [1.\ 6.] $$

$\times$ zde označuje Hamiltonův součin [1. 7.]

$$ \begin{aligned} a \times b = [a_1, a_2, a_3, a_4] \times [b_1, b_2, b_3, b_4] = \\ = \begin{bmatrix} a_1 b_1 - a_2 b_2 - a_3 b_3 - a_4 b_4 \\ a_1 b_2 + a_2 b_1 + a_3 b_4 - a_4 b_3 \\ a_1 b_3 - a_2 b_4 + a_3 b_1 + a4b2 \\ a_1 b_4 + a_2 b_3 - a_3 b_2 + a_4 b_1 \end{bmatrix}^T \end{aligned} \qquad [1.\ 7.] $$

Pro opačný případ, tj. rotace vektoru v kartézském prostoru pomocí kvaternionu, poslouží vztah [1. 8.], kde jsou oba figurující kvaterniony jednotkové (odmocnina součtu kvadrátů všech členů je rovna jedné).

$$ v' = q \times v \times q^* \qquad [1.\ 8.] $$

q^* je komplexně sdružený kvaternion s q.

Pro komplexně sdružený tvar kvaternionu platí obdobný vztah jako ve standardních komplexních číslech (změna polarity imaginární části)

$$ {}^{A}_{B}q^{*} = {}^{B}_{A}q = [\,{}^{A}_{B}q_1\,, -{}^{A}_{B}q_2\,, -{}^{A}_{B}q_3\,, -{}^{A}_{B}q_4\,] \qquad [1.\ 9.] $$

Převod do DCM

Kvaternionem lze vektor v kartézských souřadnicích rotovat pomocí vztahu [1. 8.]. Postačí pouze dosadit osy vektoru obecně a vztah upravit [1. 10.].

$$ v = [0, \boldsymbol{x}, \boldsymbol{y}, \boldsymbol{z}] $$

$$ q = [q0, q1, q2, q3] $$

$$ q^{*} = [q0, -q1, -q2, -q3] $$

$$ v' = q \times v \times q^{*} = \qquad [1.\ 10.] $$

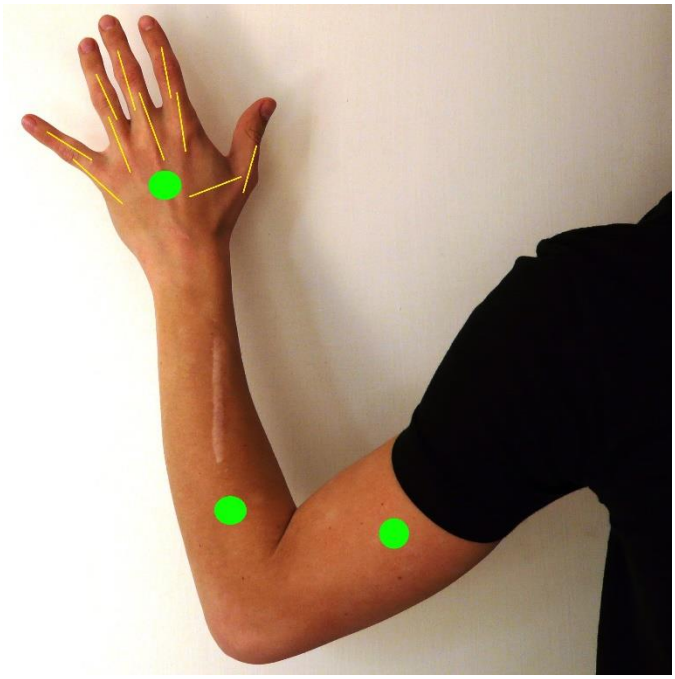
$$ \begin{bmatrix} \mathbf{0} \\ (\boldsymbol{q0^2 + q1^2 - q2^2 - q3^2})\boldsymbol{x} + \boldsymbol{2(q1 * q2 - q0 * q3)y} + \boldsymbol{2(q1 * q3 + q0 * q2)z} \\ \boldsymbol{2(q0 * q3 + q1 * q2)x} + (\boldsymbol{q0^2 - q1^2 + q2^2 - q3^2})\boldsymbol{y} + \boldsymbol{2(q2 * q3 - q0 * q1)z} \\ \boldsymbol{2(q1 * q3 + q0 * q2)x} + \boldsymbol{2(q0 * q1 - q2 * q3)y} + (\boldsymbol{q0^2 - q1^2 - q2^2 + q3^2})\boldsymbol{z} \end{bmatrix}^{\boldsymbol{T}} $$

Úpravou na vztah DCM platí

$$ \boldsymbol{R} = \begin{bmatrix} (\boldsymbol{q0^2 + q1^2 - q2^2 - q3^2}) & \boldsymbol{2(q0 * q3 + q1 * q2)} & \boldsymbol{2(q1 * q3 + q0 * q2)} \\ \boldsymbol{2(q1 * q2 - q0 * q3)} & (\boldsymbol{q0^2 - q1^2 + q2^2 - q3^2}) & \boldsymbol{2(q0 * q1 - q2 * q3)} \\ \boldsymbol{2(q1 * q3 + q0 * q2)} & \boldsymbol{2(q2 * q3 - q0 * q1)} & (\boldsymbol{q0^2 - q1^2 - q2^2 + q3^2}) \end{bmatrix} \qquad [1.\ 11.] $$

2 Senzory

Pro měření a vyhodnocení polohy a pohybů ruky bylo využito více druhů senzorů. Pro ohyby prstů byly vzhledem k jednoduchosti vyhodnocení měřených dat zvoleny flex senzory. Na Obr. 3 je jejich umístění naznačeno žlutými čarami. Pro vyhodnocení natočení částí dlaně, předloktí a pažní kosti byl vybrán senzor typu MARG. Jeho umístění je na Obr. 3 vyznačeno světle zelenými kolečky. Senzory jsou rozmístěny tak, aby bylo možné s jejich malým počtem snímat co nejlépe polohu a pohyby ruky. Díky znalostem délky kostí a vhodnému umístění senzorů lze také určit vzájemnou polohu a pohyby jednotlivých částí ruky.



Obr. 3 Znázornění umístění senzorů

2.1 MARG senzor

MARG *(angl. Magnetic, Angular Rate and Gravity)* senzor je nejčastěji se vyskytující zkratkou označení MEMS senzoru, který v sobě ukrývá nejméně tři dílčí tříosé senzory -akcelerometr, kompas a gyroskop. Tento senzor je konstruován především pro určení relativně přesného natočení ve 3D prostoru. Pojem „relativní“ je zde umocněn i tím, že nelze jednoduše objektivně posoudit, zda jsou výstupní data senzoru přesná. Většinou se za dostatečně objektivní považuje zpětná vazba okem člověka nebo porovnání dat vyhodnocených různými metodami pro určení natočení. Ze své podstaty pro určení natočení

teoreticky postačí pouze gyroskop. Pro řešení takového problému by byl potřeba ideální gyroskop. Levné MEMS gyroskopy založené na principu vibrující struktury se v přesnosti ani zdaleka ideálnímu gyroskopu nepřibližují. Pro kompenzaci chyb gyroskopu v MARG senzorech se využívá akcelerometru a kompasu. Akcelerometr a kompas mají samozřejmě také své vlastní nevýhody pro určení natočení v prostoru. Není proto jednoduché najít optimální způsob vzájemné kompenzace chyby při měření, aby určení natočení bylo co nejpřesnější. Tato kompenzace obvykle probíhá skrze tzv. fúzní algoritmus *(angl. fusion algorithm).*

Cena těchto senzorů se pohybuje v řádu jednotek euro především díky hromadnému nasazení v mobilních zařízeních typu tablet nebo telefon. Díky MEMS technologii dosahují rozměry pouzdra těchto senzorů jednotek milimetrů.

Často se lze také setkat v obdobné problematice s pojmy 9DOF senzor, IMU nebo AHRS. 9DOF senzor (*angl. degrees of freedom)* je podmnožinou MARG senzoru. Jedná se o senzor s devíti nezávislými osami snímání. IMU *(inertial motion unit)* je v podstatě elektronická obdoba libely s kompenzovanou boční akcelerací gyroskopem a určuje relativně přesné natočení Eulerových úhlů „roll" a „pitch". IMU představuje 6DOF senzor a nejčastěji se skládá z tří osého gyroskopu a akcelerometru. Zkratka AHRS *(attitude and heading reference system)* označuje systém určený pro poskytování informace o natočení letadel. Skládá se ze tříosých senzorů a představuje obecnější formu MARG.

2.1.1 Gyroskop

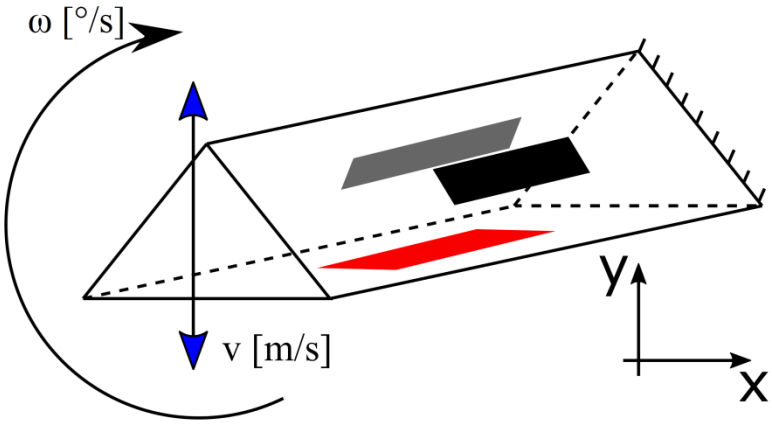
Gyroskop je obecně zařízení využívané pro navigaci. Bližší specifikace je velice obtížná, jelikož druhy gyroskopů se liší jak principem, tak výstupem. Výstupem gyroskopu mohou být hodnoty rotační rychlosti nebo Eulerovy úhly rotace objektu v prostoru. Obecně jsou známy gyroskopy založené na Coriolisově jevu nebo optické gyroskopy založené na Sagnacově jevu.

V praxi nezanedbatelným parametry tohoto zařízení je jejich cena a velikost. Při výběru senzoru pro aplikaci snímání pohybů ruky byla snaha oba tyto parametry minimalizovat. Z tohoto důvodu byl pro aplikaci zvolen již zmíněný MARG MEMS senzor.

Princip funkce „micro vibrating structure" gyroskopu

Na Obr. 4 je zobrazen trojboký hranol, který má na každém boku připevněn piezoelektrický materiál. Přivedením náboje do tohoto materiálu lze změnit jeho objem a tím

změnit i jeho rozložení rozměrů (např. délku). Díky tomu při přivedení střídavého napětí o rezonanční frekvenci hranolu na jednu z jeho stěn s připevněným piezoelektrickým materiálem lze hranol rozkmitat. Piezoelektrický materiál označuje na Obr. 4 červený obdélník. Pohyb hranolu pak v ostatních dvou piezoelektrických prvcích generuje náboj, který je stejný při antisériovém zapojení na obou prvcích. Na Obr. 4 jsou tyto měřící piezoelektrické prvky označeny šedou a černou barvou. Z vnějšího pohledu (bráno jako napěťový výstup) nenaměříme žádnou změnu napětí. Při rotaci zobrazené na Obr. 4 působí na rozkmitaný hranol Coriolisova síla.



Obr. 4 Teoretický princip "micro vibrating structure gyroskopu"

$$F_c = 2 * m * v \times \omega \qquad [2.\ 1.]$$

$\times$ v tomto případě vektorový součin

Corrielisova síla má za následek nevyvážení nápoje na měřicích piezoelektrických prvcích a změnu napětí na antisériovém zapojení. Velikost tohoto napětí je úměrná vychýlení hranolu ve směru osy x (Obr. 4). Vychýlení hranolu je díky vazbě skrze Coriolisovu sílu úměrné rotační rychlosti celého objektu.

Jak bylo již uvedeno, gyroskop neměří úhlovou rychlost zcela přesně. Základní chyby gyroskopu popisuje následující obecný vztah [2. 2.]

$$\omega = \omega_{TRUE} + \omega_{ER} \qquad [2.\ 2.]$$

$$\omega_{ER} = \omega_{AVR}(\Theta) + \omega_{NOICE}(t, \Theta) + \omega_{NONLIN}(\omega)$$

Pro určení rotace je problematická především chyba se stejnosměrnou složkou, která po integraci způsobuje drift natočení. Dále jsou zde nemalým problémem chyby při překročení mechanického rozsahu nebo dosažení maximálního rozsahu měření. Všechny

tyto nedokonalosti se v praxi pro MARG senzory řeší zvýšením váhy akcelerometru a kompasu.

Matematicky lze obtížněji popsat změnu chyb mechanickým šokem, takže i pokud gyroskop zkalibrujeme, po mechanickém šoku pravděpodobně vzroste míra chyby především stejnosměrné složky. (vylepšené komplementární filtry např. Madgwick během měření pomocí výstupní rotace průběžně kalibrují gyroskop a tím snižují vliv stejnosměrné složky chyby gyroskopu).

2.1.2 Akcelerometr

Akcelerometr představuje senzor využívaný pro měření zrychlení. Pracuje na principu setrvačné síly. V MEMS provedení akcelerometru probíhá vyhodnocení setrvačné síly standardně přes změnu kapacity objektu vychýleného vnějším zrychlením vůči pevně umístěným elektrodám v senzoru. Problematická je především nelinearita výstupu. Tento základní princip se označuje jako první generace MEMS akcelerometru.

Přibližně v devadesátých letech minulého století se začal využívat vylepšený způsob uvedeného principu založený na zpětnovazební smyčce, který je označovaný jako druhá generace. První část smyčky je stejná jako v první generaci, ale vychýlení pohyblivé části akcelerometru je vyvažováno vnější silou, elektrickým polem nebo piezoelektrickým principem, aby pohyblivá část setrvávala ve středu pohybového rozsahu. Vyhodnocení výstupu spočívá v úměrnosti síly, kterou se pohyblivá část udržuje na místě, a díky tomu se výstup chová lineárněji v rozsahu funkce.

2.1.3 Magnetometr

Magnetometr je přístroj (senzor), který měří intenzitu i směr magnetického pole. Tímto se liší od kompasu, který je určený pro měření pouze směru magnetického pole. V problematice MARG snímání natočení pro nás není informace o celkové intenzitě magnetického pole příliš důležitá, ale lze jí využít například pro zdokonalení fúzních algoritmů. Například při násobně větší intenzitě magnetického pole než je pole geomagnetické umožňuje tato informace snížit váhu kompasu. V této situaci lze předpokládat rušení cizím zdrojem mg. pole.

Pro snímání magnetického pole existuje velké množství principů. Senzory mohou být založené na Hallovu jevu, Lorentzově síle nebo na využití kvantových jevů v důsledku působení magnetického pole. Tyto principy ve svém článku dobře popisuje např. Cai et

al. [5]. Jelikož výrobci v zájmu obchodního tajemství často informace o principu funkce magnetometrů neuvádějí, nebylo jednoduché určit, na kterém principu magnetometry využívané pro MARG senzory pracují. Nízká cena a relativně velká spotřeba proudu vůči akcelerometru a gyroskopu nasvědčují tomu, že se v nich pravděpodobně využívá princip Hallova jevu.

2.1.4 Vyhodnocení dat ze senzorů

V současné době se lze na trhu setkat s moderními MARG senzory, které přímo v pouzdře obsahují procesor pro zpracování dat. Vzorkem takového senzoru může být např. senzor společnosti Bosch – BNO055, který interně pracuje s fúzním algoritmem na principu Kalmanova filtru. Dnes již tedy existuje cesta, jak se relativně složitému způsobu vyhodnocení dat ze senzoru za mírný příplatek vyhnout. Vyhodnocení dat přímo u senzorů má také výhodu ve frekvenci aktualizace dat přímo ze senzorů bez nutnosti rychlé komunikace po sběrnici. Další výhodou je potřeba menšího výpočetního výkonu na straně vyhodnocení dat ze senzorů. Společnost Bosch uvádí u již zmíněného senzoru 10krát efektivnější vyhodnocení dat než u standardních MARG senzorů. Výstupem tohoto senzoru může být přímo informace o natočení senzoru v 3D prostoru. Tato kapitola se však bude dále věnovat zpracování signálů z hrubých dat dílčích senzorů.

Jak již bylo zmíněno v úvodu o MARG senzorech, rotaci lze určit ze senzorů akcelerometru a magnetometru. Setkáváme se zde s obdobou problému, který stanovila v roce 1965 anglická profesorka Grace Wahba v matematice. Tzv. Wahba's problém se zabývá stanovením rotace ve 3D prostoru pomocí dvou nezávislých vektorů.

Pro stanovení rotace je výhodné využívat kvaterniony (viz kap. 1.1.3). Vstupem jsou data z akcelerometru a magnetometru, požadovaný výstup pak tvoří kvaternion určující natočení v prostoru.

V prvním kroku je stanoven počáteční kvaternion [2. 3.]. Algoritmy výpočtu rotace pro řešení Wahba's problému pro kvaterniony jsou rekurzivní a z tohoto počátečního bodu se vychází. V dalších cyklech je tento krok nahrazen předáním minulého kvaternionu do funkce.

$$q_0 = [1\ 0\ 0\ 0] \qquad [2.\ 3.]$$

Při práci s kvaterniony je výhodné vektory normovat. Normování tedy bude druhým krokem při určení tohoto natočení. Pro normování lze využít vztah pro Pythagorovu větu

rozšířený do tří dimenzí. Norma pak odpovídá tělesové úhlopříčce kvádru vytvořeného ze tří hodnot senzorů.

Pro akcelerometr:

$$\begin{aligned} norm_a &= \sqrt{{a_x}^2 + {a_y}^2 + {a_z}^2}\ \left[\frac{m}{s^2}\right] \\ a_x &= \frac{a_x}{norm_a}\left[\frac{\mathrm{m}}{\mathrm{s}^2}\right] \\ a_y &= \frac{a_y}{norm_a}\left[\frac{\mathrm{m}}{\mathrm{s}^2}\right] \\ a_z &= \frac{a_z}{norm_a}\left[\frac{\mathrm{m}}{\mathrm{s}^2}\right] \end{aligned} \qquad [2.\ 4.]$$

Analogicky pro magnetometr:

$$\begin{aligned} norm_m &= \sqrt{{m_x}^2 + {m_y}^2 + {m_z}^2}\ [T] \\ m_x &= \frac{m_x}{norm_m}[T] \\ m_y &= \frac{m_y}{norm_m}[T] \\ m_z &= \frac{m_z}{norm_m}[T] \end{aligned} \qquad [2.\ 5.]$$

Následně jsou tyto hodnoty seřazeny do vektoru [2. 6.]

$$Z_s = [a_x, a_y, a_z, m_x, m_y, m_z]^T \qquad [2.\ 6.]$$

a stanoven referenční vektor senzorů [2. 7.]

$$Z_0 = [a_{x0}, a_{y0}, a_{z0}, m_{x0}, m_{y0}, m_{z0}]^T \qquad [2.\ 7.]$$

U akcelerometru je předpokládáno nulové natočení vektoru od osy z. U magnetometru je stanovení referenčního vektoru díky inklinaci více problematické. Referenční vektor lze dopočítat přímo podle znalosti inklinace a osu roviny země vybrat volitelně. Možné stanovení referenčního vektoru magnetometru popisují následující vztahy

$$m_{x0} = cos(\alpha_{inkl})\ [T] \qquad [2.\ 8.]$$

$$m_{z0} = sin(\alpha_{inkl})\ [T] \qquad [2.\ 9.]$$

Výsledný referenční vektor je tedy konstantní

$$Z_0 = [0{,}0{,}1, cos(\alpha_{inkl})\ , 0, sin(\alpha_{inkl})\ ]^T \qquad [2.\ 10.]$$

Další variantou je neurčit referenční vektor pro magnetometr přímo, ale dopočítávat jej během každého cyklu oproti aktuálnímu natočení [2. 11.]. Nemalou výhodou tohoto přístupu je nepotřebnost znalosti inklinace pro aktuální polohu a možnost algoritmu správně pracovat i v umělém statickém magnetickém poli. Nevýhodou změny referenčního směru magnetického pole v každém cyklu může být nestabilita výsledku. Při tomto přístupu k problému není ani pomocí metody gradient-descent zaručena konvergence. Vektor magnetometru je možné určit pouze obecně a směr referenčního magnetického pole dopočítat v každém cyklu např. pomocí kvaternionu podle dat z magnetometru. Tato metoda bude označována v této práci jako dynamická metoda určení referenčního vektoru pro magnetometr.

$$[0, b_x, b_y, b_z] = q \times [0, m_x, m_y, m_z] \times q^* \qquad [2.\ 11.]$$

$$Z_0 = [0,0,1, \sqrt{{b_x}^2 + {b_y}^2}, 0, b_z]^T \qquad [2.\ 12.]$$

Ze stanovení vektorů Z_0 *a* Z_s lze určit chybový vektor

$$\epsilon = Z_0 - M * Z_s \qquad [2.\ 13.]$$

$$M = \begin{bmatrix} R & 0 \\ 0 & R \end{bmatrix} \qquad [2.\ 14.]$$

Matice M [2. 14.] o rozměru 6x6 je složená ze dvou dílčích DCM pro každý senzor. Umístění nul znamená nulovou submatici 3x3. DCM matici R popisuje vztah [1. 11.].

Z chybového vektoru lze vypozorovat, že pokud by chyba byla nulová, matice R by obsahovala přesně rotaci, kterou se snažíme určit.

Dalším krokem k určení natočení pomocí akcelerometru a magnetometru je tedy minimalizovat chybovou funkci [2. 13.]. K této minimalizaci mohou posloužit rekurzivní minimalizační algoritmy jako např. metoda gradient-descent nebo Gauss-Newtonova metoda.

Metoda gradient-descent je méně výpočetně náročná, ale zaručuje nalezení alespoň jednoho lokálního minima. Její nevýhodou je však pomalejší konvergence. Gauss-Newtonova metoda nezaručuje konvergenci k minimu a je výpočetně náročnější, ale v oblastech blížících se k minimu konverguje velmi rychle.

Pro porovnání těchto metod byl vytvořen skript v softwaru MATLAB. Rovnice pro výpočty metod byly převzaty z [2] a [4]. Metody byly testovány statisticky. Výsledky odpovídají teoretickým předpokladům. Označení „bez konvergence“ má v následujícím přehledu význam nedosažení rozdílové zastavovací podmínky během uvedených cyklů.

rychlost konvergence při statickém určení referenčního vektoru magnetometru

Zastavovací podmínka: změna všech částí kvaternionu mezi předchozí a současnou hodnotou menší než 0,00005 nebo počet kroků vyšší než 200.

Výsledkem je aritmetický průměr na 10 000 pokusů s pseudonáhodnými vstupními vektory akcelerometru a magnetometru:

gradient-descent	15,22 kroku	
Gauss-Newton	21,51 kroku	956krát bez konvergence (9,6 %)

rychlost konvergence při dynamickém určení referenčního vektoru magnetometru

Zastavovací podmínka: změna všech částí kvaternionu mezi předchozí a současnou hodnotou menší než 0,001, nebo počet kroků vyšší než 500.

Výsledkem je aritmetický průměr na 10 000 pokusů s pseudonáhodnými vstupními vektory akcelerometru a magnetometru:

gradient-descent	25,6 kroku	232krát bez konvergence (2,3 %)
Gauss-Newton	33,4 kroku	809krát bez konvergence (8,1 %)

Rychlost výpočtu

Výpočet proveden v software MATLAB R2012B na středně výkonném PC.

Přímá doba výpočtu rovnice:

gradient-descent	18 μs
Gauss-Newton	48 μs

Doba výpočtu jednoho kroku se statickým určením referenčního vektoru magnetometru:

gradient-descent	70 μs
Gauss-Newton	93 μs

Doba výpočtu jednoho kroku s dynamickým určením referenčního vektoru magnetometru:

gradient-descent	105 μs
Gauss-Newton	137 μs

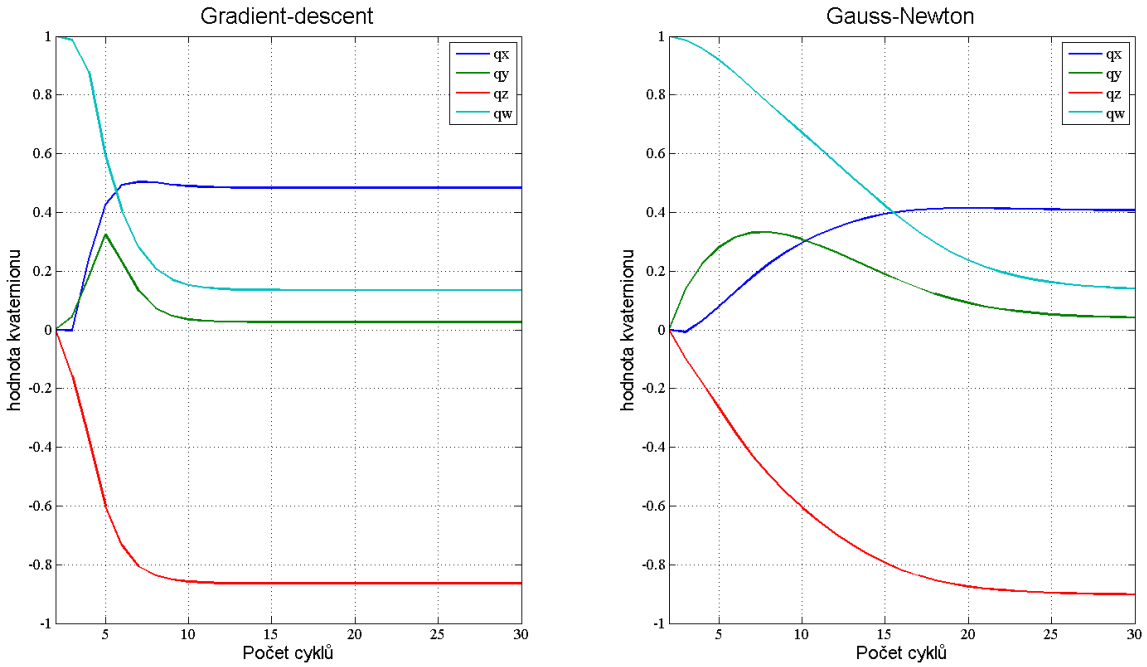
Z testovaných dat lze pozorovat, že podle předpokladů konvergují metody znatelně pomaleji s dynamickým určením referenčního vektoru magnetometru. Lze taky pozorovat,

že Gauss-Newtonova metoda s testovanými náhodnými vektory konverguje hůře než metoda Gradient-descent (přibližně o 60 %). Tento test ale není nastaven jako optimální pro Gauss-Newtonovu metodu. Při reálných datech se kromě počátku nemění hodnota kvaternionu o tak vysokou hodnotu mezi jednotlivými vzorky dat. Podle předpokladů má metoda Gradient-descent také menší problémy s konvergencí.

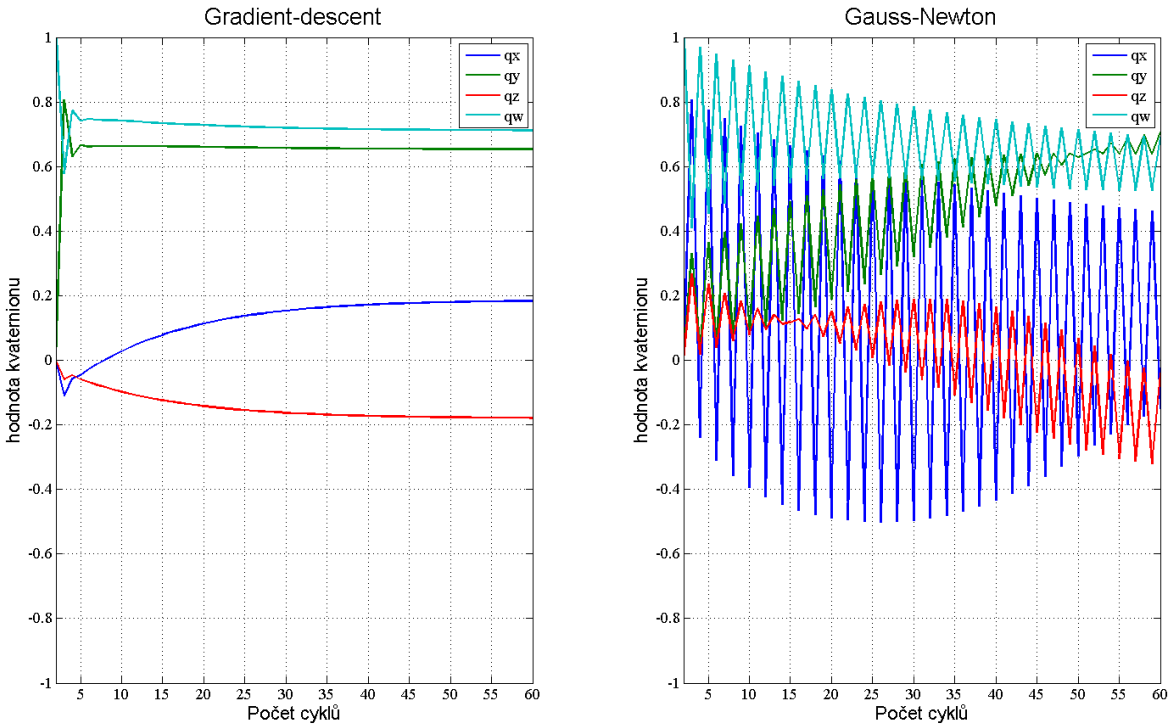
Čas výpočtu samotné rovnice, která je pro tyto metody rozdílná, nemá pro získání nového kvaternionu příliš vysokou výpovědní hodnotu, protože v každém kroku je nutné provést přepočet chybové funkce a Jacobiho matice. U dynamické metody je nutný také přepočet referenčního magnetického vektoru. Podle předpokladů lze také sledovat větší potřebu výpočetního výkonu pro Gauss-Newtonovu metodu.

Testování s daným nastavením tedy dopadlo jednoznačně pro jednoduchý, ale robustní algoritmus metody Gradient-descent.

Na Obr. 5 je znázorněn průběh konvergence metod. Na Obr. 6 je zobrazena možná nestabilita Gauss-Newtonovy metody. Obr. 6 také zachycuje začátek minimalizace chybového vektoru, která teoreticky dál konvergovat může, ale při testování jsem narazil na vektory, u kterých se Gauss-Newtonova metoda ustálila v rozkmitaném stavu.



Obr. 5 Ukázka příkladu konvergence metod



Obr. 6 Možnost nekonvergence Gauss-Newtonovy metody

Výsledné natočení pomocí akcelerometru a magnetometru stačí po vhodné zastavovací podmínce odečíst, čímž lze získat základní AHRS, ale s nevýhodami uvedenými pro akcelerometr a magnetometr v teoretických kapitolách.

U gyroskopu je nejprve stanoven diferenciální kvaternion [2. 16.] a z něho je posléze numerickou integrací určeno natočení [2. 17.].

$$^{S}\omega = [0, \omega_x, \omega_y, \omega_z] \qquad [2.\ 15.]$$

$$\dot{{}^{S}_{E}q}_t = \frac{1}{2} {}^{S}_{E}q_{t-1} \times {}^{S}\omega \qquad [2.\ 16.]$$

$${}^{S}_{E}q_t = {}^{S}_{E}q_{t-1} + \dot{{}^{S}_{E}q}_t * dt \qquad [2.\ 17.]$$

K fúzi gyroskopu a akcelerometru s magnetometrem existují dva základní přístupy. Prvním z nich je stochastický přístup založený na Kalmanovu filtru a algoritmu prediktor-korektor. V této práci jsem se ale pro jednoduchost rozhodl pracovat s filtrem deterministickým založeným na využití frekvenčních vlastností senzorů. Tento filtr spadá mezi filtry komplementární. Pro vyšší frekvence se využívá gyroskop a pro nízké frekvence

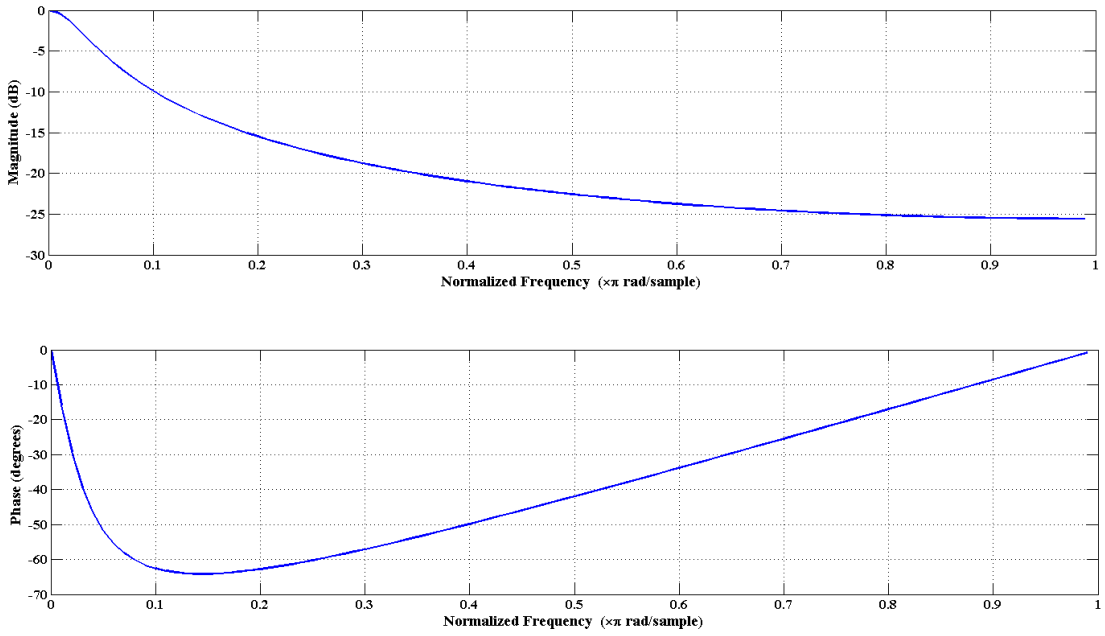
akcelerometr s magnetometrem. Driftová chyba gyroskopu se tímto principem vykompenzuje a dynamické akcelerace se silně utlumí.

Základní vztah pro fúzi senzorů komplementárním filtrem

$$q = (1 - \alpha) * q_g + \alpha * q_{am} \ \{\alpha \in (0,1)\} \qquad [2.\ 18.]$$

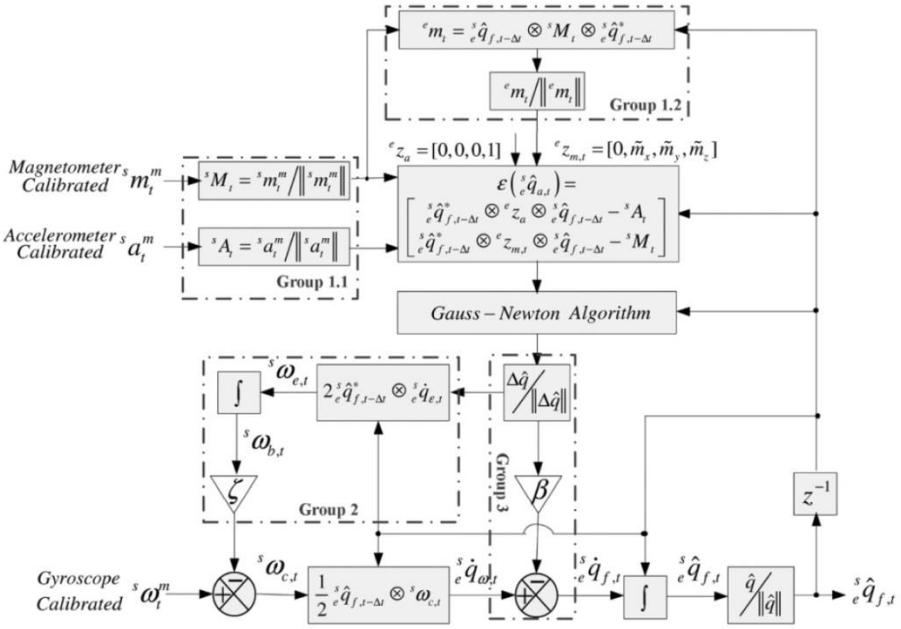
Koeficient α určuje oddělovací frekvenci komplementárního filtru. Snižování koeficientu přikládá větší váhu gyroskopu a naopak. U běžných levných MARG senzorů se využívá hodnota koeficientu okolo 0,02. Tento vztah vychází z předpokladu, že data z gyroskopu byla již dříve filtrována horní propustí pro odstranění driftové složky gyroskopu, tedy i kvaternion z těchto dat je již filtrovaný.

U nízkých koeficientů α lze zanedbat mírnou nepřesnost z podstaty tohoto komplementárního filtru a váhu gyroskopu $(1 - \alpha)$ porovnat jedné. Na Obr. 7 je zobrazena frekvenční charakteristika filtru při nulovém vstupním vektoru gyroskopu a nastavení konstanty α na hodnotu 0,1. Z grafů lze vyčíst, že se podle teoretických předpokladů komplementární filtr pro rotaci určenou z akcelerometru a kompasu chová jako dolní propust. Dále je zde patrné, že fázová charakteristika je nelineární a díky tomu lze předpokládat, že i skupinové zpoždění filtru nebude konstantní.



Obr. 7 Frekvenční charakteristika komplementárního filtru s nulovou hodnotou gyroskopu

Na obrázku vidíme blokové schéma vylepšeného komplementárního filtru, jež navrhly Ya Tian et al. [6].



Obr. 8 Blokové schéma komplementárního filtru [6]

2.1.5 Flex senzor

Flex senzor představuje jednoduchý moderní senzor ve formě plastového pásku s aktivní vrstvou určený ke snímání ohybu. Jeho výstupem je změna odporu. Měření ohybu tímto senzorem jsem se věnoval již v bakalářské práci [1], zde bych chtěl doplnit spíše princip funkce těchto senzorů.

Změna odporu vzniká díky změně rozložení rozdílně vodivých částí v aktivní vrstvě. Aktivní vrstva senzoru je tedy kompozitní materiál, kde vodivou část zajišťují drobné částečky uhlíku nebo stříbra, které jsou vhodně rozloženy v elastickém hůře elektricky vodivém materiálu. Typicky tuto hůře vodivou část aktivní vrstvy tvoří polymerní materiál, např. pryž.

Při ohybu plastového pásku se elastický materiál napíná do stran a drobné částečky vodivého materiálu se od sebe oddalují. Díky této změně vzdálenosti se změní i celkový odpor flex senzoru. Díky vhodnému rozložením vodivých částí v nevodivém materiálu ve flex senzoru lze změnu odporu upravit tak, aby byla téměř lineárně závislá na ohybu.

2.2 Statické fyzikální veličiny snímané senzory

Tato kapitola popisuje statická pole snímaná MARG senzorem pro získání natočení ekvivalentnímu k natočení z gyroskopu. Dále jsou zde zmíněny i problémy s využíváním těchto polí pro určení natočení v prostoru.

2.2.1 Geomagnetické pole

Geomagnetické pole (dále jen GMP) je magnetické pole vytvořené procesy země. Toto pole se dělí na pole vzniklé pod povrchem země a pole vzniklé v atmosféře. S prvním jmenovaným se obecně můžeme setkat pod pojmem vnitřní GMP, které tvoří cca 99 % z celkové síly magnetického pole. S polem, které vzniká v horní atmosféře, se můžeme setkat pod označením vnější GMP. Přestože toto pole oproti vnitřnímu GMP nemá na celkový magnetický tok velký vliv, je z hlediska výzkumu země velmi důležité.

I v dnešní době stále není zcela jasné, jak přesně vnitřní GMP vzniká. Hlavní hypotéza je založena na tom, že jádro země je relativně dobře elektricky vodivé. Vlivem pohybu jádra ve slabším magnetickém poli se vytváří proudy v jádře a důsledkem těchto proudů je vznik vnitřního GMP země.

Vnitřnímu GMP se z dlouhodobějšího hlediska mění směr i intenzita. V delším časovém horizontu dokonce se dochází i k prohození pólů. Jedná se však o náhodný jev, ke kterému dochází i několikrát za milion let. Procentní změny intenzity se však pohybují v řádu desítek let a změna polohy pólu činí cca 3 km za rok. Vnější GMP se oproti tomu mění vlivem slunečního větru a rotace země každý den. Po superpozici s vnitřním GMP se tak poloha pólů mění denně po elipsovité křivce s hlavní poloosou o maximální délce okolo 80 km. Změna polohy pólu je závislá na aktuální síle slunečního větru.

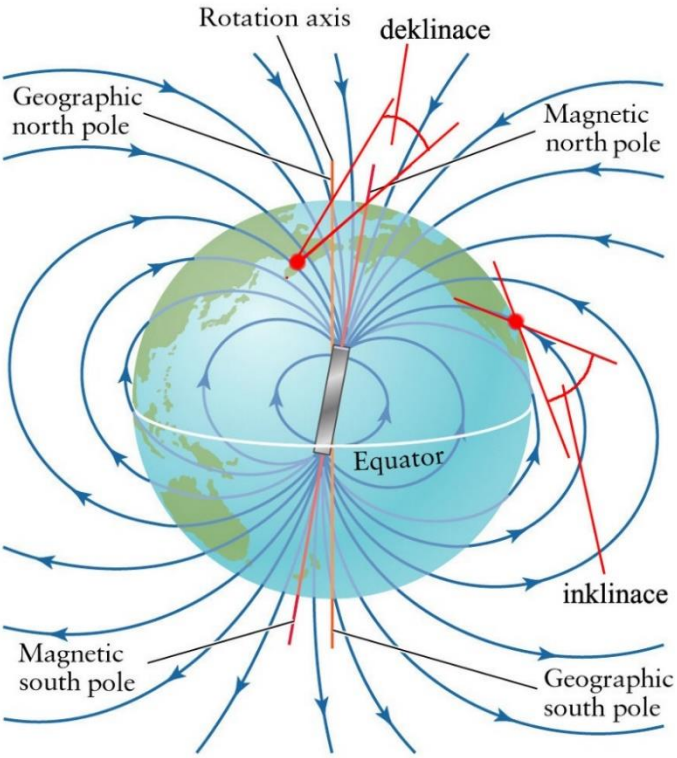
K tomuto tématu se vztahuje ještě jedna zajímavost. Severní magnetický pól se aktuálně nachází v blízkosti severního geografického pólu, nicméně je to pouze norma, protože z fyzikálního hlediska je to magnetický pól jižní.

Velikost a směr GMP

Tato podkapitola se více zaměřuje na využití magnetického pole pro určení přesného směru na povrchu země.

Velikost GMP není oproti uměle vytvořeným magnetickým polím nikterak velká (neodymové magnety dosahují indukce i v řádech Tesla), ale většina těchto uměle

vytvořených polí působí jen ve velmi malé oblasti v okolí. V rovníkové oblasti se lze setkat s hodnotou GMP okolo 30 μT, v polárních oblastech pak okolo 60 - 70 μT.



Obr. 9 Zobrazení deklinace a inklinace geomagnetického pole

Velmi důležitá informace, se kterou jsem se před realizací této práce nikdy nesetkal, se týká směru GMP. Jak bylo již dříve v obecnějších informacích zmíněno, jižní magnetický pól zcela přesně neodpovídá poloze severního geografického pólu. Pokud se tedy nacházíme kdekoliv jinde než v ploše vytyčené osou rotace země a magnetickou osou země, nebude pomyslná ručička kompasu ukazovat na geografický pól, ale na pól magnetický. Tato odchylka vyjádřená úhlem se nazývá magnetická deklinace, která však není velkou překážkou pro aplikaci MARG senzorů v rámci praktické části práce. Pro Plzeň činí hodnota magnetické deklinace v roce 2015 03°18'. Nicméně cílem práce není stanovit přesný sever, nýbrž zastavit drift gyroskopu. Větším problémem je méně známý termín magnetická inklinace. Ta popisuje úhel mezi pomyslnou rovinou země v bodě měření a směrem magnetického pole. Hodnota inklinace je nejvíce závislá na magnetické zeměpisné šířce a na rovníku se blíží k nule. Značně větší hodnota inklinace však přísluší již poloze ČR. Pro rok 2015 činí hodnota magnetické inklinace pro Plzeň 65°41'09". S ohledem na nepřesnosti měření gravitačního zrychlení akcelerometrem se kompenzace driftu úhlu „yaw“ v oblastech

jen vzdáleně se blížících magnetickým pólům jeví jako nepoužitelná. Hodnoty magnetické inklinace a deklinace pro určitou lokalitu lze nalézt na webových stránkách NOAA [8]. Pro větší názornost jsou obě výše zmíněné veličiny znázorněny na Obr. 9.

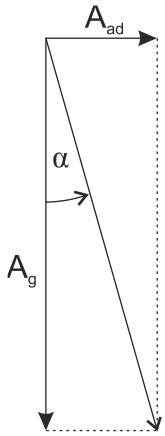
Dalším dosti nemalým problémem je pro danou aplikaci mírná změna směru při superpozici s umělým zdrojem magnetismu nebo v blízkosti magneticky dobře vodivých objektů. Po praktických zkušenostech s měřením GMP pomocí MEMS senzoru bylo ovlivnění měření směru umělým zdrojem magnetismu prokazatelné až při přiblížení nástěnného magnetu na přibližně 20 cm. U dobře vodivých magnetických látek je ovlivnění ještě nižší, ale závisí na velikosti objektu, který ovlivňuje pole.

2.2.2 Akcelerace

Akcelerace neboli zrychlení je pojem, který v této práci nemá smysl hluboce rozvádět, ale bude zde uvedeno pouze několik informací týkajících se řešené problematiky.

Základem pro stanovení inerciální soustavy je stanovení směru gravitačního zrychlení. Pro tento problém se využívá tří vzájemně nezávislých akcelerometrů. Ve stanovení směru gravitačního zrychlení brání především přidání dalších tzv. dynamických zrychlení. Výsledným zrychlením je vektorový součet všech dílčích zrychlení (Obr. 10). U tohoto jevu je určitou komplikací, že při sčítání vektorů se úhel směru výsledného zrychlení nemění lineárně, ale s funkcí arkus tangens, tedy nejcitlivěji právě s přidáním i malých zrychlení.

$$\alpha = tan^{-1}\frac{A_{ad}}{A_g}\left[^{\circ},\frac{m}{s^2},\frac{m}{s^2}\right] \qquad [2.\ 19.]$$



Obr. 10 Vliv dynamické akcelerace na snímání statické akcelerace

Dále se lze setkat s pojmy statické a dynamické zrychlení. Za statické zrychlení se považuje gravitační zrychlení bez dalších přidaných zrychlení. Analogicky dynamické zrychlení je zrychlení působící na objekt bez působení gravitačního zrychlení. Pokud se vyskytujeme na povrchu země, je pro získání zrychlení dynamického nutné vždy odstranit statickou složku zrychlení od celkového zrychlení.

Teoreticky je možné se setkat i se změnou směru gravitačního zrychlení. Jedná se například o změnu přílivu a odlivu v blízkosti pobřeží nebo vliv astronomických těles, ale pro potřeby práce je pro stanovení inerciální soustavy pro měření přesnost a stálost směru gravitace dostačující.

3 Implementace senzorického systému

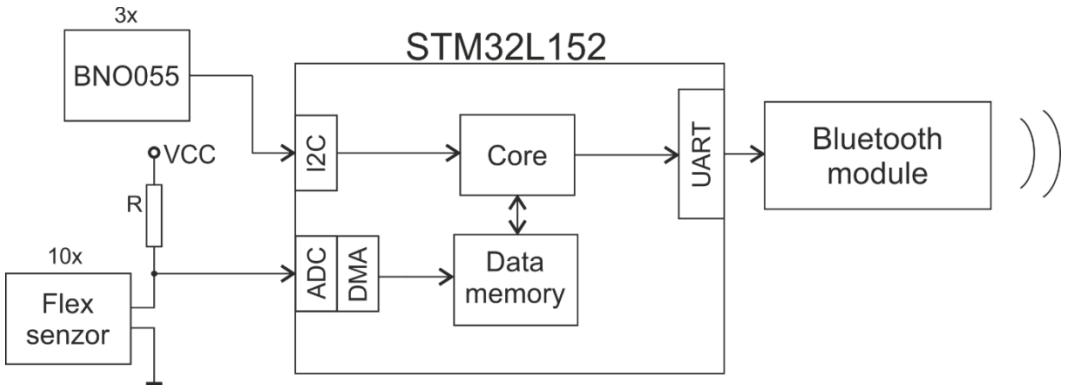
V předchozích kapitolách byla shrnuta práce s jednotlivými senzory. Práci s flex senzory byla věnována část bakalářské práce [1]. Tato kapitola se bude věnovat především jejich implementaci pro snímání polohy a pohybů ruky, v podobě zvolené autorem práce.

Flex senzory jsou umístěny na upravené zahradnické rukavici tak, aby snímaly polohu prstů [1]. Na rukavici je na svrchní straně připevněna malá deska plošných spojů s mikroprocesorem určená pro řízení celého systému, na které je umístěn jeden MARG senzor pro určení natočení. Další dva senzory jsou umístěny na dalších menších DPS. Ty jsou umístěny v plastových pouzdrech pro možnost připevnění na paži elastickými popruhy.

Jako senzor natočení byl zvolen senzor společnosti BOSCH – BNO055, který i za příznivé ceny levných MEMS senzorů obsahuje vlastní procesorovou jednotku na zpracování dílčích senzorů. Oproti stejnému MEMS senzoru bez procesorové jednotky je nutné počítat s příplatkem cca 80 %. Jako fúzní algoritmus využívá interně Kalmanův filtr a výstupní rotaci lze z registrů vyčítat přímo ve formě kvaternionu.

Celý systém je napájen z malé Li-Pol baterie (180 mAh, 7,4 V). Pro napájecí zdroj je problematické napájení BNO055 senzorů. Každý z těchto senzorů ve fúzním režimu odebírá proud cca 10 mA. Celý systém v provozu pak dohromady odebírá proud z baterie 74,6 mA. Ideální doba provozu na baterii je tedy 2,4 hodiny.

Na následujícím Obr. 11 je zobrazeno principiální blokové schéma, které znázorňuje tok informace v senzorickém systému.



Obr. 11 Blokové schéma senzorického systému

Jak je zobrazeno na Obr.11, mikroprocesor komunikuje se senzorem BNO055 po sběrnici I2C a změnu odporu flex senzoru vyhodnocuje pomocí děliče napětí A/D převodníkem. Data z A/D převodníku jsou ukládána do paměti skrze jednotku DMA a dále

jsou normována na hodnotu od nuly do sta. Toto normování vychází z kalibrace flex senzorů, kterou lze na začátku programové smyčky provést. K odskoku do kalibrační smyčky programu se využívá dynamické akcelerace senzoru BNO055. Uživatel má po resetu programu 6 sekund na prudký pohyb rukou doleva a doprava pro vsup do kalibrace. Následně má uživatel 6 sekund na ohnutí prstů do maximálních poloh, které jsou zaznamenány (7.2).

Následně jsou data seřazena do string řetězce a odeslána skrze bluetooth. Formát kvaternionů ve stringu je „quat[číslo kvaternionu]: [hodnota kvaternionu] \t “. Formátování dat z Flex senzorů je „AD[číslo flex senzoru]: [kalibrovaná hodnota AD převodníku] \t “.

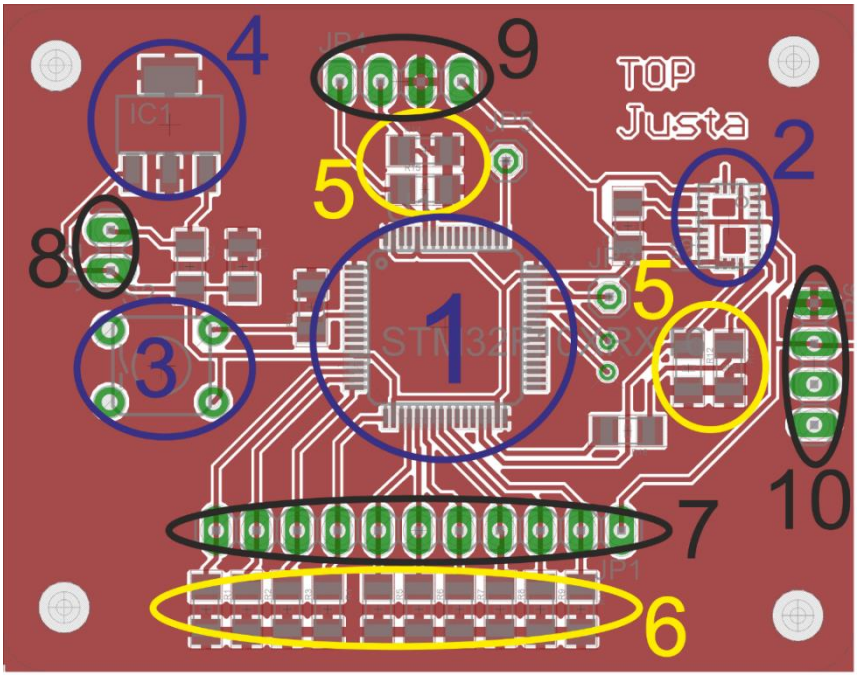
3.1 Desky plošných spojů

Jak již bylo zmíněno obecně, systém je ovládaný pomocí mikroprocesoru STM32L152. Tento mikroprocesor byl vybrán podle parametrů nízké spotřeby, $\geq 10x$ ADC a $\geq 2x$ I2C. Potřeba dvou nezávislých sběrnic I2C vychází z nemožnosti změnit adresu senzoru BNO055 na více než dvě adresy.

Mikroprocesor je umístěn na řídící DPS (4,1x5,4 cm) a zpracovává data ze všech senzorů, které posílá skrze bluetooth modul do PC. Na řídící DPS se dále nachází lineární sériový stabilizátor, odpory vytvářející napěťový dělič s flex senzorem, součástky podporující funkci zařízení a jeden senzor pro určení natočení. Další dva senzory pro určení natočení umístěné na malých DPS jsou propojeny vodiči po sběrnici I2C, paralelně je vedeno i napájení pro senzory. Na Obr. 12 a Obr. 13 je zobrazen návrh řídící desky a desky pro senzor.

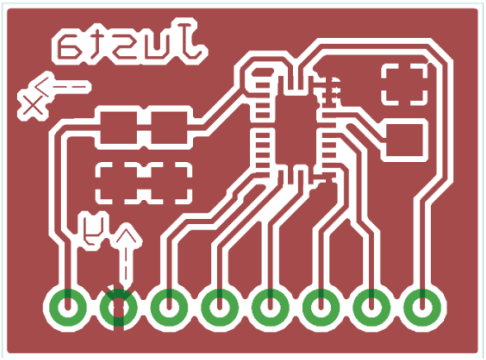
Na Obr. 12 čísla označují:

1) Mikroprocesor - STM32L152
2) MARG senzor BNO055
3) Tlačítko reset
4) Lineární stabilizátor napětí- TLV1117-33CDCYR
5) Pull-up rezistory pro I2C
6) Rezistory vytvářející dělič s Flex senzorem
7) Pájecí plochy pro Flex senzory
8) Pájecí plochy pro přívody z baterie
9) Pájecí plochy pro dva BNO055 senzory ležící mimo tuto DPS
10) Pájecí plochy pro bluetooth modul



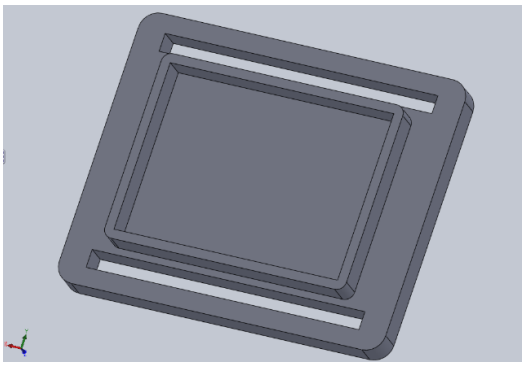
Obr. 12 Řídící DPS senzorického systému

DPS na následujícím Obr. 13 je určená pouze pro rozvod základních propojení a vyvedení pinů z pouzdra senzoru BNO055.



Obr. 13 DPS pro rozvod pinů z BNO055

Tato deska je umístěna v pouzdru vytvořeném pro stabilní upevnění desky se senzorem (Obr. 14). Pouzdro má v sobě díry k přichycení elastických popruhů, které jsou určeny pro přichycení DPS se senzorem k ruce. Pouzdra byla vymodelována v programu Solidworks a vytištěna na 3D tiskárně.



Obr. 14 Pouzdro pro senzory na paži

Celý systém na paži v reálném provedení je zobrazen na následujícím Obr. 15.

Obr. 15 Senzorický systém v reálné podobě

3.2 Aplikace pro testování MARG senzoru

Při získání kvaternionu z MARG senzorů lze z hrubých dat jen obtížně zjistit, jestli jsou správná. Pro kontrolu a testování byla data vizualizována v XNA aplikaci s kostkou. XNA jsou knihovny C# původně určené především pro tvorbu her na Xbox360 [6]. V softwaru Blender bylo vymodelován model DPS s texturou.

Fragmenty zdrojového kódu aplikace:

Data ze senzorů jsou přijímány ze senzorického systému přes COM port vytvořený zařízením bluetooth. Příjem dat vyvolá událost a skočí do funkce pro příjem dat (analogie interruptu v mikroprocesorech).

```
private void serialPort1_DataReceived(object sender,
System.IO.Ports.SerialDataReceivedEventArgs e)
        {
            while (port.BytesToRead > 0) //je počet bytů pro příjem větší než 0?
            {
                port.Read(r_buf, i, 1);  //čti jeden byte a ulož ho do
                                         //vektorového pole s posunem o i
                if (r_buf[i] == '\r')    //přišel speciální znak konce řádku?
                {
                    zobraz();            //zavolej funkci pro zpracování přijatého
                                         //řetězce
                    i = 0;               //vynuluj posun
                }
                i++;
                if (i > 499)             //ochrana přesahu paměti
                {
                    i = 0;
                }
            }
        }
```

Pro převod pole znaků na číselné hodnoty byla vytvořena funkce zobraz().

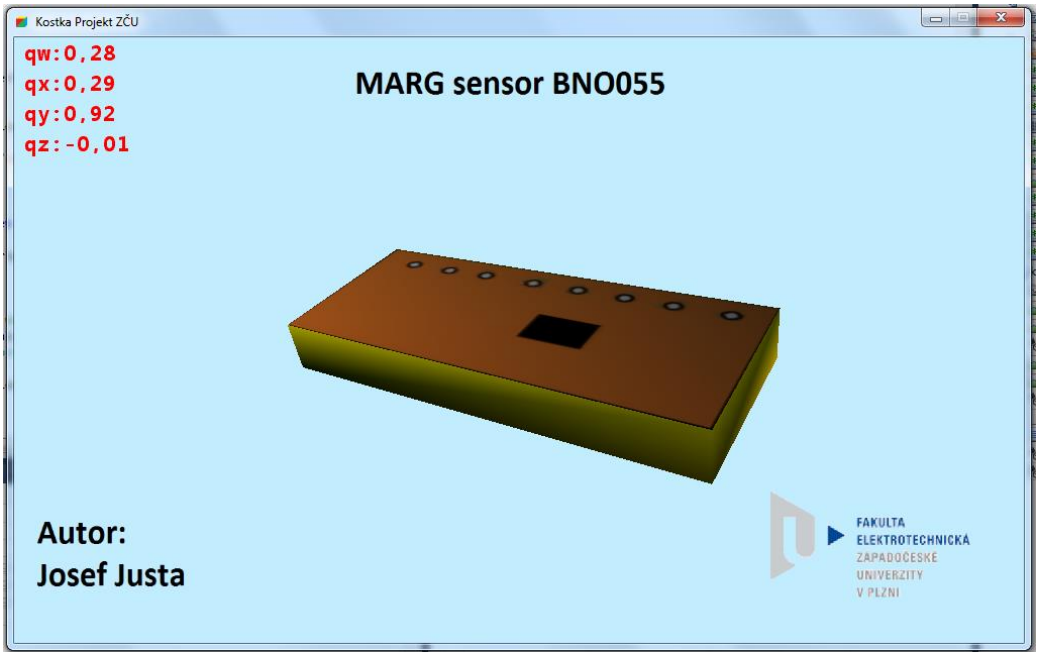
```
    private void zobraz()
        {
            string q;
            long del = 16384;
            int quat_num, j;
            string s = new string(r_buf);   //převeď znakové pole do formátu string

            for (quat_num = 1; quat_num < 13; quat_num++)//v cyklu postupně vyber
                                                         //všechny hodnoty
                                                    quaternionu
            {
                q = "quat" + quat_num + ":";    //vytvoř string řetězec analogický s
                                                //vysílanými daty
                if (s.Contains(q))              //obsahuje přijatý řetězec string q?
                {
                    int i = s.IndexOf(q) + q.Length;  //ulož následující hodnotu do
                                                      //proměnné i a j
                    j = i;

                    while (r_buf[i] != '\t' && r_buf[i] != '\r' && (i - j) < 100)
                                                //dokud nenarazíš na speciální znak cykly
                    {
                        rollc[i - j] = r_buf[i];      //ulož znak s posunem o i do
                                                      //řetězce
                        i++;                          //inkrementuj i
                    }
                    rolls = string.Join("", rollc);   //vytvoř ze znaků řetězec
                                                      //string

                    try
                    {
                        switch (quat_num)
                        {
                            case 1:
                                quat1.W = (float)Convert.ToDouble(rolls) / del;
                                                //převeď řetězec na hodnotu a normuj
                                break;
                            case 2:
                                quat1.Z = (float)Convert.ToDouble(rolls) / del;
```

```
                                                    //převeď řetězec na hodnotu a normuj
                                    break;
                                case 3: …

                            }
                            catch (FormatException)     //převod selhal
                            {
                             Console.WriteLine("Unable to convert '{0}' to a Double.",
                            rolls);
                            }
                            Array.Clear(rollc, 0, rollc.Length); //uvolni řetězec
                        }

                    }
                }
```

Za zmínění stojí ještě funkce z knihovny XNA, která převádí kvaternion na DCM matici pro natočení objektu (v XNA se standardně otáčí okolní svět a ne objekt samotný). Tímto jednoduchým zápisem v XNA je vytvořeno natočení objektu pomocí kvaternionu.

```
World = Matrix.CreateFromQuaternion(quat1);
```

Snímání dat v reálném čase a jejich zobrazení v aplikaci je uvedeno na Obr. 16. K diplomové práci je pro lepší názornost přiložen také videozáznam běhu této aplikace (viz 7.5).



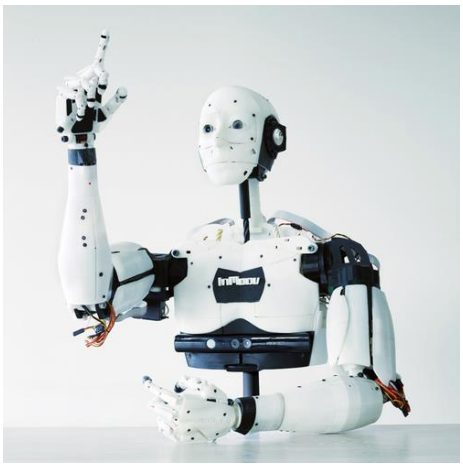
Obr. 16 Aplikace pro testování MARG senzoru

4 Robotická ruka

V rámci praktické části práce byla vytvořena robotická ruka, přesněji mechanická ruka poháněná servomotory. Označení „robotická“ je spíše méně vhodným termínem z hlediska přesnosti (zejména po prostudování několika definic robota), nicméně pro představivost běžných lidí je z praktické zkušenosti výhodný. Vytvořená ruka působí při provozu velmi efektně, a díky tomu může zaujmout oko mladých techniků, ve kterém tento nebo podobné projekty mohou být skrytou motivací pro jejich další studium a vlastní projekty.

4.1 Projekt InMoov

Pro realizaci při porovnání dostupných zdrojů jsem se rozhodl pro tisk ruky na 3D tiskárně. Z hlediska náročnosti jak finanční, tak i časové jsem po dohodě s vedoucím práce vybral tisk ruky s předloktím podle open source projektu „InMoov“. Tento projekt s tvorbou humanoida vytvořil Francouz Gael Langevin [3] a aktuálně je v rozsahu celé horní poloviny těla (viz Obr. 17).

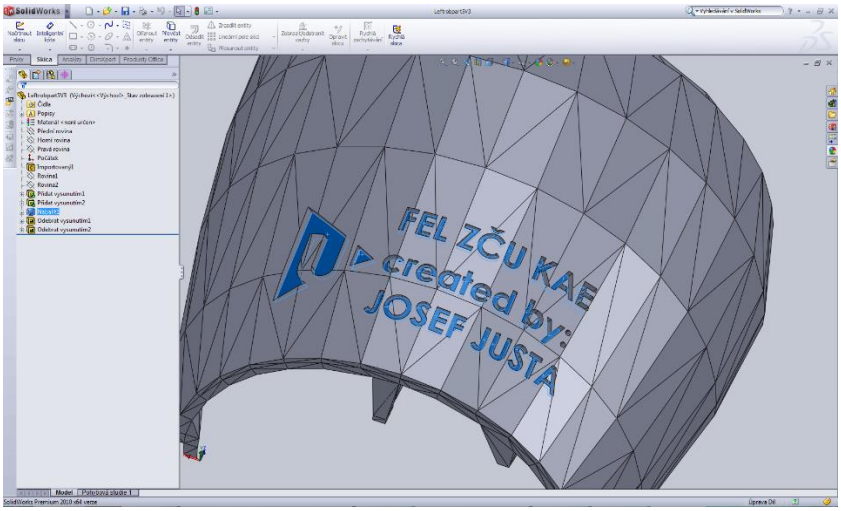


Obr. 17 Projekt InMoov
(http://static1.squarespace.com/)

Části těla jsou vymodelovány pro možnost vytištění i na levnějších typech 3D tiskáren s možností tisku 12x12x12 cm. Vytisknuté díly lze sestavit s pomocí běžně dostupných nástrojů, lepidla a několika šroubů. Návod na sestavení lze dohledat přímo na stránkách projektu [3]. Při sestavování dílu se někdy vyskytli mírné problémy s kompatibilitou dílů různých verzí. Po vytištění hrubých dílů ve 3D tiskárně je potřeba počítat někdy s nemalou úpravou dílů nožem. Tyto úpravy závisí na vhodnosti nastavení 3D

tisku a složitosti dílů. Pro představu tisk ruky s předloktím trval na aditivní 3D tiskárně dostupné na katedře přibližně mezi 40 a 50 hodinami čistého tisku.

Jeden díl byl upraven v programu Solidworks pro potřeby katedry (viz Obr. 18).



Obr. 18 Úprava dílu pro katedru

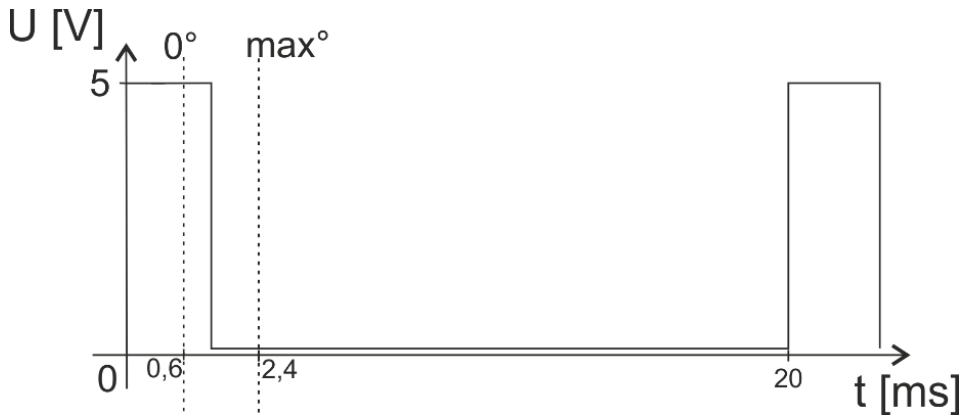
Ruka s předloktím je ovládána pomocí šesti servomotorů, díky nimž pohyby robotické ruky napodobují pouze základní pohyby ruky lidské. Pět servomotorů umístěných v předloktí ovládá pohyby jednotlivých prstů pomocí umělých šlach ve formě rybářských pletených vlasců. Poslední servomotor umístěný v předloktí u zápěstí otáčí zápěstím přes převody vytištěné také na 3D tiskárně.

4.2 Ovládání servomotorů

Pro pohyby robotické ruky doporučují v projektu „InMoov“ servomotory MG946r. Vzhledem k ceně a dostupnosti ale byly vybrány servomotory s analogickými parametry HK15288A.

Oba uvedené servomotory patří do skupiny servomotorů s analogovým řízením. Pro jejich funkci jsou vyvedeny tři vodiče do známého JR konektoru, kde standardně hnědý označuje napájecí zem, červený kladný potenciál napájecího napětí a oranžový řídící signál.

Řídící signál servomotoru je standardizován na signál s periodou 20 ms (Obr. 19). Signál lze obecně považovat za diskrétní v hodnotě („0“ nebo „1“) a spojitý v čase. Délka trvání kladného impulzu nese informaci o požadovaném natočení servomotoru. Kladný impulz délky 600 µs odpovídá natočení 0°, délky 2400 µs natočení maxima. Lineární interpolací mezi těmito hodnotami lze získat celý rozsah servomotoru.



Obr. 19 Řídící signál analogového servomotoru

Kvůli absenci více akčních členů na ovládání jednoho prstu a snímání ohybu jednoho prstu dvěma flex senzory jsem rozdělil požadovaný ohyb prstu s 50% váhou pro každou kalibrovanou hodnotu z flex senzoru. Výsledné otočení je tedy spočteno ekvivalentně k následujícímu vztahu

$$\alpha_{serv} = 0{,}5 * flex_1 + 0{,}5 * flex_2 \qquad [4.\ 1.]$$

Natočení servomotoru pro zápěstí je vypočteno z kvaternionu na dlani a kvaternionu na předloktí podle základního vztahu uvedeného v teoretické kapitole o kvaternionech [1. 6.] (viz 7.4). Výsledný kvaternion je pro získání úhlu otočení servomotoru převeden na Eulerovy úhly. Tento přepočet vychází z ekvivalentnosti DCM matic vyjádřených z rotační Eulerovy matice a z kvaternionů. Převod z kvaternionů na Eulerovy úhly ale není bezproblémový a v praxi je nutné ještě řešit problém se singularitami a normováním úhlu. Teoreticky fungující výpočet Eulerova úhlu „Yaw“ ukazují následující vztahy

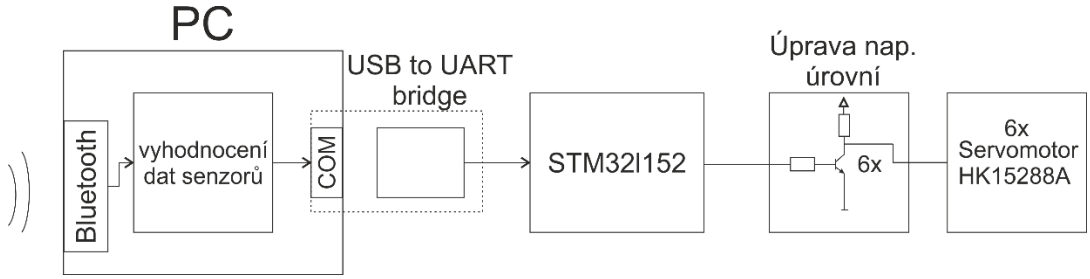
$$\overset{s1}{s2}\boldsymbol{q} = \overset{s2}{eart}\boldsymbol{q}^{*} \times \overset{s1}{eart}\boldsymbol{q} \qquad [4.\ 2.]$$

$$\boldsymbol{\vartheta} = \mathbf{arctan2}\left\{\mathbf{2} * \left(\overset{s1}{s2}\boldsymbol{q_x} * \overset{s1}{s2}\boldsymbol{q_y} + \overset{s1}{s2}\boldsymbol{q_z} * \overset{s1}{s2}\boldsymbol{q_w}\right), \mathbf{1} - \mathbf{2} * \left(\overset{s1}{s2}\boldsymbol{q_y} * \overset{s1}{s2}\boldsymbol{q_y} + \overset{s1}{s2}\boldsymbol{q_z} * \overset{s1}{s2}\boldsymbol{q_z}\right)\right\} \qquad [4.\ 3.]$$

Funkce arkus tangens 2 je zde využita, protože umí na rozdíl od standardního arkus tangens rozeznat 1. a 3. kvadrant při vykreslení vstupních hodnot do kartézského souřadnicového systému.

Celý systém toku informace a jejího zpracování zobrazuje blokové schéma na Obr. 20. Data jsou ze senzorického systému přijímána přes bluetooth a v aplikaci .NET s pomocnou knihovnou XNA pro práci s kvaterniony jsou data ze senzorů přepočtena na požadované natočení všech servomotorů. Tato aplikace dále posílá informace o natočení přes virtuální sériovou linku do modulu se zařízením UART. Data jsou přijata do mikroprocesoru

STM32L152 na discovery kitu a zde jsou převedena na inverzní signály pro servomotory (viz 7.1). Dále jsou signály napěťově upraveny na signál vhodný pro servomotory zapojením tranzistorů se společným emitorem.



Obr. 20 Blokové schéma řízení servomotorů robotické ruky

5 Závěr

Práce se z větší části zabývala návrhem a realizací senzorického systému pro snímání polohy a pohybů ruky. Součástí práce byla vlastní konstrukce systému, jeho návrh a úprava senzorické rukavice z mé předešlé bakalářské práce. V písemné části byly popsány jednotlivé součásti senzorického systému, komunikace mezi senzorickým systémem a řízeným systémem. Dále pak konstrukcí a ovládání robotické ruky podle projektu InMoov.

V první kapitole bylo pojednáváno o možných způsobech vyjádření rotace ve 3D prostoru. Byly zde vysvětleny klady a zápory využívání různých metod pro vyhodnocení natočení orientované na MARG senzory.

Druhá nejrozsáhlejší kapitola této diplomové práce pojednává o senzorech využitých pro snímání polohy a pohybů ruky. Především je zaměřena na senzory typu MARG a uvedeny jsou zde principy práce těchto senzorů, veličiny snímané těmito senzory a jejich výhody i nevýhody pro určení natočení. Je zde i uvedeno několik informací týkajících se Flex senzorů.

O implementaci, praktické realizaci a programování souvisejícím se senzorickým systémem pojednává třetí kapitola. Je zde popsána komunikace se senzory, návrh desek plošných spojů a je zde uvedeno několik informací o návrhu pouzdra pro přichycení senzoru BNO055 k ruce.

V poslední kapitole je popsána práce na robotické ruce vytištěné na 3D tiskárně podle open source projektu InMoov. V této kapitole se také věnuji základní teorii řízení analogových servomotorů použitých v projektu.

Senzorický systém pro snímání polohy a pohybů ruky byl realizován. Robotická ruka byla podle projektu InMoov vytvořena. Komunikace mezi tímto systémem a robotickou rukou byla otestována a všechny praktické části lze považovat za funkční.

Návrhem na další vylepšení praktické části práce (vytvořené robotické ruky) může být implementace modulu přijímače bluetooth do systému a zpracování dat v mikroprocesoru, díky čemuž by bylo možné vyřadit z řetězce nutnost počítače s převodníkem na sériovou linku.

6 Literatura

[1] Justa, Josef. Aplikace polohových snímačů pro řízení, Bakalářská práce, Plzeň: Západočeská univerzita. 2013

[2] Madgwick, Sebastian. [online]. An efficient orientation filter for inertial and inertial/magnetic sensor arrays. Internal report. Duben 30, 2010.

[3] Langevin, Gael. [online web]. InMoov Humanoid project. http://www.inmoov.fr/

[4] Comotti, Daniele. [online]. Orientation estimation based on Gauss-Newton method and implementation of a quaternion. https://code.google.com/p/9dof-orientation-estimation/

[5] By Yongyao Cai, Yang Zhao, etc. [online]. Magnetometer basics for mobile phone applications. http://www.memsic.com/

[6] Ya Tian, Hongxing Wei, Jindong Tan. [online]. An Adaptive-Gain Complementary Filter for Real-Time Human Motion Tracking With MARG Sensors in Free-Living Environments, 2012, ISSN: 1534-4320

[7] Visan, Adrian-Florin. [online]. XNA tutorial: Drawing a simple cube. http://iloveshaders.blogspot.cz/2011/04/drawing-simple-cube.html

[8] National centers for enviromental information. Magnetic Field Calculators. [online]. http://www.ngdc.noaa.gov/

[9] White, Steve. [online]. Applications of Quaternions. http://www.zipcon.net/~swhite/docs/math/quaternions/applications.html

7 Přílohy

7.1 Program STM32l152 pro ovládání robotické ruky

```
#include <stm32l1xx.h>
#include "stm32l1xx_conf.h"
#include <stdio.h>
#include "uart.h"
#include "init.h"

#define min_1 60
#define max_1 236

char s[200];
volatile long time=0;
unsigned int stat_calib=0,time_calib=0;
int time_s1,time_s2,time_s3,time_s4,time_s5,time_s6;
int s1=100,s2=100,s3=100,s4=100,s5=100,s6=100;  // min 59=3B max 237=ED

void send_uart(char data[])     //funkce pro odeslání znaku
{
        int i=0;
        while(data[i])
        {
                fputchar(data[i]);
                i++;
        }
}

void delay(int tim)     //čeká tim*10us
{
        long t=time;
        while((tim+t)>time);
}

void TIM6_IRQHandler(void) // vyvolaný po 10us
{
        time_calib++;           //inkrementace proměnných pro časování
        time++;
        time_s1++;
        time_s2++;
        time_s3++;
        time_s4++;
        time_s5++;
        time_s6++;

        if(time_calib>600000) //odpočet doby pro vstup do kalibrace
        {
                stat_calib=1;
        }

        if(time_s1>1999) //tvorba signálu pro servomotor 1
        {
        time_s1=0;
        GPIO_ResetBits(GPIOA, GPIO_Pin_0);
        }
        if(time_s1>s1)
        GPIO_SetBits(GPIOA, GPIO_Pin_0);

        if(time_s2>1999) //tvorba signálu pro servomotor 2
        {
        time_s2=0;
        GPIO_ResetBits(GPIOA, GPIO_Pin_1);
        }
        if(time_s2>s2)
        GPIO_SetBits(GPIOA, GPIO_Pin_1);
```

```
        if(time_s3>1999) //tvorba signálu pro servomotor 3
        {
        time_s3=0;
        GPIO_ResetBits(GPIOA, GPIO_Pin_2);
        }
        if(time_s3>s3)
        GPIO_SetBits(GPIOA, GPIO_Pin_2);

        if(time_s4>1999) //tvorba signálu pro servomotor 4
        {
        time_s4=0;
        GPIO_ResetBits(GPIOA, GPIO_Pin_3);
        }
        if(time_s4>s4)
        GPIO_SetBits(GPIOA, GPIO_Pin_3);

        if(time_s5>1999) //tvorba signálu pro servomotor 5
        {
        time_s5=0;
        GPIO_ResetBits(GPIOA, GPIO_Pin_4);
        }
        if(time_s5>s5)
        GPIO_SetBits(GPIOA, GPIO_Pin_4);

        if(time_s6>1999) //tvorba signálu pro servomotor 6
        {
        time_s6=0;
        GPIO_ResetBits(GPIOA, GPIO_Pin_5);
        }
        if(time_s6>s6)
        GPIO_SetBits(GPIOA, GPIO_Pin_5);

        TIM_ClearFlag(TIM6, TIM_FLAG_Update);   //uvolnění flagu časovače
}

void USART1_IRQHandler(void){                   //přerušení vyvolané příjmem dat

        static uint8_t cnt = 0;

        if( USART_GetITStatus(USART1, USART_IT_RXNE) ){

                char t = USART1->DR;  //čtení dat do proměnné


                if( (t != (char)0x65) && (cnt < 200) ){
                        s[cnt] = t;     //vloží data do pole char na hodnotu posuvu
                        cnt++;
                }
                else{
                        cnt = 0;//při příchodu speciálního znaku nuluje posuv
                }
        }
}

int main(void)
{

        GPIO_pin_conf();        //inicializace výstupu
        InitializeTimer_interr();       //inicializace timeru a jeho přerušení
        USART_Config();         //inicializace příjmu dat

        send_uart("Wello world!!");     //řetězec pro kontrolu správné funkce UART

        while(1)
        {
                delay(10);
                s1=(int)((float)(max_1-min_1)*s[1]/100)+min_1;  //upravení dat pro
                                                                //natočení servo1

                s2=(int)((float)(max_1-min_1)*s[2]/100)+min_1;  //upravení dat pro
                                                                //natočení servo2
```

```
                s3=(int)((float)(max_1-min_1)*s[3]/100)+min_1;   //upravení dat pro
                                                                 //natočení servo3

                s4=(int)((float)(max_1-min_1)*s[4]/100)+min_1;   //upravení dat pro
                                                                 //natočení servo4

                s5=(int)((float)(max_1-min_1)*s[5]/100)+min_1;   //upravení dat pro
                                                                 //natočení servo5

                s6=(int)((float)(max_1-min_1)*s[0]/100)+min_1;   //upravení dat pro
                                                                 //natočení servo6

        }
}
```

7.2 Program pro STM32l152 v senzorickém systému

```
#include <stm32l1xx.h>
#include "stm32l1xx_conf.h" //includy stdp
#include <stdio.h>
#include "uart.h"
#include "i2c.h"
#include "init.h"
#include "BNO055.h"

#define ARRAYSIZE 8*4

volatile long time=0;
unsigned int stat_calib=0,time_calib=0;
volatile uint16_t ADC_values[ARRAYSIZE];
__IO uint16_t ADC_C_Val[10];
__IO uint16_t ADC_C_Val_l[10];
__IO uint16_t ADC_C_Val_h[10];
float ADC_C_Val_cal[10];



void send_uart(char data[])     // fukce pro odesílání dat
{
        int i=0;
        while(data[i])
        {
                fputchar(data[i]);
                i++;
        }
}

float invSqrt(float x) {        // funkce 1/sqrt() nízko výpočtově náročná
        float halfx = 0.5f * x;
        float y = x;
        long i = *(long*)&y;
        i = 0x5f3759df - (i>>1);
        y = *(float*)&i;
        y = y * (1.5f - (halfx * y * y));
        return y;
}

void delay(int tim)             // funkce pro spoždění tim*10us
{
        long t=time;
        while((tim+t)>time);
}

void TIM6_IRQHandler(void) // vyvolaný po 10us
{
        time_calib++;
        time++;
```

```
        if(time_calib>500000)
        {
                stat_calib=1;
        }

        TIM_ClearFlag(TIM6, TIM_FLAG_Update);
}

void calib(void)        // kalibrační část programu pro flex senzory
{
        int iter;
        stat_calib=1;
        for(iter=0;iter<10;iter++)
        {
                ADC_C_Val_l[iter]=ADC_C_Val[iter];
                ADC_C_Val_h[iter]=ADC_C_Val[iter];
        }

        time=0;
        send_uart("6 sekund calibrace");
        while(time<600000)      // po dobu 6 sekund hledá nejnižší a nejvyšší hodnotu
        {
                for(iter=0;iter<10;iter++)
                {
                        if(ADC_C_Val[iter]<ADC_C_Val_l[iter])
                                ADC_C_Val_l[iter]=ADC_C_Val[iter];
                        if(ADC_C_Val[iter]>ADC_C_Val_h[iter])
                                ADC_C_Val_h[iter]=ADC_C_Val[iter];
                }
        }
}

int main(void)
{
        char s[200];
        float qw,qx,qy,qz,qw2,qx2,qy2,qz2,qw3,qx3,qy3,qz3;
        float lacc_x,lacc_y,lacc_z,lacc_low,lacc_high,lacc_1;
        int i;

        InitializeTimer_interr();       // inicializace timeru
        USART_Config();         // inicializace UART

        send_uart("Hello world!!");     // kontrolní text

        I2C1_init();                    // inicializace I2C1
        init_imu(SLAVE_ADDRESS1);       // inicializace MARG 1
        init_imu(SLAVE_ADDRESS2);       // inicializace MARG 2
        I2C2_init();                    // inicializace I2C2
        init_imu2(SLAVE_ADDRESS1);      // inicializace MARG 3

        ADC_DMA_Config((uint32_t*)&ADC_C_Val[0]);       // inicializace DMA od ADC

        cti_lin_acc(&lacc_x,&lacc_y,&lacc_z,SLAVE_ADDRESS1);
        lacc_1=lacc_y;


        while(1)
        {
        cti_quat(&qw, &qx,&qy,&qz,SLAVE_ADDRESS1);      // čtení hodnot kvaternionu
                                                                // senzorů BNO055
        cti_quat(&qw2, &qx2,&qy2,&qz2,SLAVE_ADDRESS2);
        cti_quat2(&qw3, &qx3,&qy3,&qz3,SLAVE_ADDRESS1);

        if(stat_calib==0)               // prvních 6 sekund možnost pohybem ruky pro
                                        // odskok na kalibraci
        {
        cti_lin_acc(&lacc_x,&lacc_y,&lacc_z,SLAVE_ADDRESS1);
                if(lacc_y<lacc_low)
                        lacc_low=lacc_y;
                if(lacc_y>lacc_high)
                        lacc_high=lacc_y;
                if((lacc_low<(lacc_1-350))&&(lacc_high>(lacc_1+350)))
```

```
                        calib();
        }

        for(i=0;i<10;i++)               // úprava přečtených hodnot adc na 0-100
        {
                if((ADC_C_Val_h[i]-ADC_C_Val_l[i])!=0)
                ADC_C_Val_cal[i]=((float)(ADC_C_Val[i]-
                ADC_C_Val_l[i])*100)/(ADC_C_Val_h[i]-ADC_C_Val_l[i]);
                else
                ADC_C_Val_cal[i]=100;
                if(ADC_C_Val_cal[i]>100)
                        ADC_C_Val_cal[i]=100;
                if(ADC_C_Val_cal[i]<0)
                        ADC_C_Val_cal[i]=0;
        }

        if(stat_calib)          /// odesílaní hodnot po kalibraci
        {
        sprintf(s,"quat1:%d\tquat2:%d\tquat3:%d\tquat4:%d\tquat5:%d\tquat6:%d\tquat7:%
        d\tquat8:%d\tquat9:%d\tquat10:%d\tquat11:%d\tquat12:%d\tad1:%d\tad2:%d\tad3:%d\tad4:%d\tad5:%
        d\tad6:%d\tad7:%d\tad8:%d\tad9:%d\tad10:%d\r",
        (int)(qw),(int)(qx),(int)(qy),(int)(qz),(int)(qw2),(int)(qx2),(int)(qy2),(int)
        (qz2),(int)(qw3),(int)(qx3),(int)(qy3),(int)(qz3),(int)ADC_C_Val_cal[0],(int)ADC_C_Val_cal[1]
        ,(int)ADC_C_Val_cal[2],(int)ADC_C_Val_cal[3],(int)ADC_C_Val_cal[4],(int)ADC_C_Val_cal[5],(int
        )ADC_C_Val_cal[6],(int)ADC_C_Val_cal[7],(int)ADC_C_Val_cal[8],(int)ADC_C_Val_cal[9]);
                send_uart(s);
        }
        }
}
```

7.3 Testovací program pro MARG v XNA

```
using System;
using System.Collections.Generic;
using System.Linq;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Media;
using System.IO.Ports;
using System.IO;
using System.Media;
using AngleEstimationApp_BetaRelease;//.Windows.Media.Media3D;

namespace ILSCubeColorsEnd
{

    public partial class Game1 : Microsoft.Xna.Framework.Game
    {
        char[] x = new char[1000];
        char[] cut = new char[1000];
        int stat = 0,i=0;
        string rolls;
        char[] rollc = new char[500];
        private SerialPort port = new SerialPort("COM6", 115200, Parity.None, 8, StopBits.One);

        Model model;
        GraphicsDeviceManager graphics;
        KeyboardState currentKeys;
        Quaternion quat,quat1, quat2, q;
        Effect simpleColorEffect;
        Matrix[] bonetr;
        Matrix World, View, Projection;
        Vector3 radAngles = new Vector3();
        Texture2D backgroundTexture;
```

```
        Filter filter;
        SpriteFont Font1;
        Vector2 FontPos;
        SpriteBatch napis;
        SpriteBatch poz;
        Rectangle screenRectangle;

        public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = "Content";
            filter = new ComplementaryFilter();
        }

        protected override void Initialize()
        {
            base.Initialize();
            this.IsMouseVisible = true;
            graphics.PreferredBackBufferWidth = 1000;
            graphics.PreferredBackBufferHeight = 600;
            screenRectangle = new Rectangle(0, 0, graphics.PreferredBackBufferWidth,
graphics.PreferredBackBufferHeight);
            graphics.ApplyChanges();

            this.port.DataReceived += new
System.IO.Ports.SerialDataReceivedEventHandler(this.serialPort1_DataReceived);
            port.Open();

        }

        protected override void LoadContent()
        {

            simpleColorEffect = Content.Load<Effect>("SimpleColor");

            model = Content.Load<Model>("untitled2");

            World = Matrix.Identity;

            View = Matrix.CreateLookAt(new Vector3(0, 0, 8), Vector3.Zero, Vector3.Up);

            Projection = Matrix.CreatePerspectiveFieldOfView(MathHelper.PiOver4,
GraphicsDevice.Viewport.AspectRatio, 1, 100);

            // Create a new SpriteBatch, which can be used to draw textures.
            napis = new SpriteBatch(GraphicsDevice);
            Font1 = Content.Load<SpriteFont>("napis1");
            backgroundTexture = Content.Load<Texture2D>("back");
        }

        protected override void UnloadContent()
        {
        }


        protected override void Update(GameTime gameTime)
        {
            World = Matrix.CreateFromQuaternion(quat1);
            base.Update(gameTime);
        }

        protected override void Draw(GameTime gameTime)
        {

            GraphicsDevice.Clear(Color.CornflowerBlue);

            napis.Begin();
            napis.Draw(backgroundTexture, screenRectangle, Color.White);

            if (stat == 1)
            {
                string output = "qw:" +  quat1.W.ToString("0.00");
```

```
                napis.DrawString(Font1, output, new Vector2(10, 0), Color.Red);
                output = "qx:" + quat1.X.ToString("0.00");
                napis.DrawString(Font1, output, new Vector2(10, 30), Color.Red);
                output = "qy:" + quat1.Y.ToString("0.00");
                napis.DrawString(Font1, output, new Vector2(10, 60), Color.Red);
                output = "qz:" + quat1.Z.ToString("0.00");
                napis.DrawString(Font1, output, new Vector2(10, 90), Color.Red);
            }
            napis.End();

            GraphicsDevice.SetVertexBuffer(vertices);
            GraphicsDevice.Indices = indices;

            simpleColorEffect.Parameters["WVP"].SetValue(World * View * Projection);
            simpleColorEffect.CurrentTechnique.Passes[0].Apply();

            bonetr = new Matrix[model.Bones.Count];
            model.CopyAbsoluteBoneTransformsTo(bonetr);

            foreach (ModelMesh mesh in model.Meshes)
            {
                foreach (BasicEffect eff in mesh.Effects)
                {
                    eff.World = World;
                    eff.View = View;
                    eff.Projection = Projection;
                    eff.EnableDefaultLighting();
                }
                mesh.Draw();
            }

            base.Draw(gameTime);
        }



        private void serialPort1_DataReceived(object sender,
System.IO.Ports.SerialDataReceivedEventArgs e)
        {
            while (port.BytesToRead > 0)
            {

                port.Read(x, i, 1);
                if (x[i] == '\r')
                {
                    zobraz();
                    i = 0;
                }
                i++;
                if (i > 499)
                {
                    i = 0;
                }
            }
        }



        private void zobraz()
        {


            string q, f = new string(x);
            int quat_num, j, it, it2;
            long del = 16384;

            string s = new string(x);
            for (quat_num = 1; quat_num < 5; quat_num++)
            {
                q = "quat" + quat_num + ":";
                if (s.Contains(q))
                {
```

```
                    int i = s.IndexOf(q) + q.Length;
                    j = i;

                    while (x[i] != '\t' && x[i] != '\r' && (i - j) < 100)
                    {
                        rollc[i - j] = x[i];
                        i++;
                    }
                    q = rollc[0].ToString();

                    rolls = string.Join("", rollc);

                    try
                    {
                        switch (quat_num)
                        {
                            case 1:
                                quat1.W = (float)Convert.ToDouble(rolls) / del;
                                break;
                            case 2:
                                quat1.Z = -(float)Convert.ToDouble(rolls) / del;
                                break;
                            case 3:
                                quat1.X = -(float)Convert.ToDouble(rolls) / del;
                                break;
                            case 4:
                                quat1.Y = (float)Convert.ToDouble(rolls) / del;
                                break;


                            default: break;
                        }
                    }
                    catch (FormatException)
                    {
                        Console.WriteLine("Unable to convert '{0}' to a Double.", rolls);
                    }
                    Array.Clear(rollc, 0, rollc.Length);
                }

            }

            stat = 1;
        }

    }

}
```

7.4 Aplikace pro příjem dat a převod pro servomotory

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Linq;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;
using System.IO;
using Microsoft.Xna.Framework;


namespace prevadec
{
    public partial class Form1 : Form
    {
```

```
        char[] x = new char[500];
        char[] x1 = new char[500];
        char[] cut = new char[500];
        char[] rollc = new char[500];
        int[] ad = new int[10];
        Quaternion quat,quat1, quat2, quat_out;
        int i = 0;
        string rolls;
        private SerialPort port = new SerialPort("COM3", 115200, Parity.None, 8, StopBits.One);
        private SerialPort port2 = new SerialPort("COM1", 115200, Parity.None, 8, StopBits.One);

        public Form1()
        {
            InitializeComponent();
        }

        private void button1_Click(object sender, EventArgs e)
        {
            this.port.DataReceived += new
System.IO.Ports.SerialDataReceivedEventHandler(this.serialPort1_DataReceived);
            if (!port.IsOpen)
            {
                port.Open();
            }
            else
            {
                port.Close();
            }

        }

        private void button2_Click(object sender, EventArgs e)
        {
            if (!port2.IsOpen)
            {
                port2.Open();
            }
            else
            {
                port2.Close();
            }
        }

        private void serialPort1_DataReceived(object sender,
System.IO.Ports.SerialDataReceivedEventArgs e)
        {

            while (port.BytesToRead > 0)
            {

                    port.Read(x, i, 1);
                    if (x[i] == '\r')
                    {
                        zobraz();
                        i = 0;
                    }
                    i++;
                    if (i > 499)
                    {
                        i = 0;
                    }
            }

        }


        private void zobraz()
        {
            int ad_num,quat_num, j;
            string q, s = new string(x);
            long del = 16384;
```

```
    for (ad_num = 1; ad_num < 11; ad_num++) //vyber postupně všechny data ad převodníků z
                                            //řetězce
    {
    q = "ad" + ad_num + ":";            //formát dat v řetězci

    if (s.Contains(q))
    {
        int i = s.IndexOf(q) + q.Length; //do paměti následující znak obsahující první
                                         //číslici
        j = i;


        while (x[i] != '\t' && x[i] != '\r' && (i - j) < 100) //vyber znaky ke
                                                             //speciálnímu znaku
        {
            rollc[i - j] = x[i];                      //ulož do char[] pole
            i++;
        }

        rolls = string.Join("", rollc);                //převeď char[] pole na string

        try
        {
            ad[ad_num - 1] = (int)Convert.ToDouble(rolls);  //konverze stringu
                                                    //obsahujícího číslice na číslo int
        }
        catch (FormatException)                         //vyjímka převodu
        {
            System.Console.WriteLine("Unable to convert '{0}' to a Int.", rolls);
        }
        Array.Clear(rollc, 0, rollc.Length);
    }
}

for (quat_num = 1; quat_num < 13; quat_num++)
{
    q = "quat" + quat_num + ":";
    if (s.Contains(q))
    {
        int i = s.IndexOf(q) + q.Length;
        j = i;

        while (x[i] != '\t' && x[i] != '\r' && (i - j) < 100)
        {
            rollc[i - j] = x[i];
            i++;
        }
        q = rollc[0].ToString();

        rolls = string.Join("", rollc);

        try
        {
            switch (quat_num)
            {
                case 1:
                    quat1.W = (float)Convert.ToDouble(rolls) / del;
                    break;
                case 2:
                    quat1.Z = -(float)Convert.ToDouble(rolls) / del;
                    break;
                case 3:
                    quat1.X = -(float)Convert.ToDouble(rolls) / del;
                    break;
                case 4:
                    quat1.Y = (float)Convert.ToDouble(rolls) / del;
                    break;

                case 9:
                    quat2.W = (float)Convert.ToDouble(rolls) / del;
                    break;
```

```
                    case 10:
                        quat2.Z = -(float)Convert.ToDouble(rolls) / del;
                        break;
                    case 11:
                        quat2.X = -(float)Convert.ToDouble(rolls) / del;
                        break;
                    case 12:
                        quat2.Y = (float)Convert.ToDouble(rolls) / del;
                        break;

                    default: break;
                }
            }
            catch (FormatException)
            {
                Console.WriteLine("Unable to convert '{0}' to a Double.", rolls);
            }
            Array.Clear(rollc, 0, rollc.Length);
        }

    }


}


private void timer1_Tick(object sender, EventArgs e)
{
    if (port2.IsOpen)
    {
        pictureBox1.BackColor = System.Drawing.Color.SpringGreen;
    }
    else
    {
        pictureBox1.BackColor = System.Drawing.Color.Red;
    }
    if (port.IsOpen)
    {
        pictureBox2.BackColor = System.Drawing.Color.SpringGreen;
    }
    else
    {
        pictureBox2.BackColor = System.Drawing.Color.Red;
    }
    if (port2.IsOpen)
    {
        int pom;
        char[] ch = new char[8];
        int[] serv = new int[6];
        Vector3 eul;

        serv[0] = (int)(ad[0] * 0.5 + ad[1] * 0.5);
        serv[1] = (int)(ad[2] * 0.5 + ad[3] * 0.5);
        serv[2] = (int)(ad[4] * 0.5 + ad[5] * 0.5);
        serv[3] = (int)(ad[6] * 0.5 + ad[7] * 0.5);
        serv[4] = (int)(ad[8] * 0.5 + ad[9] * 0.5);

        quat_out = Quaternion.Multiply(Quaternion.Conjugate(quat2), quat1);

        eul = FromQ2(quat_out);
        if (eul.Z > 200)
        {
            serv[5] = (int)(-(eul.Z - 360) / (double)1.0);
            if (serv[5] < 0)
            { serv[5] = 0; }
            if (serv[5] > 100)
            { serv[5] = 100; }
        }

        ch[0] = (char)101;
        for (pom = 1; pom < 7; pom++)
        {
```

```
                    ch[pom] = Convert.ToChar(serv[(pom - 1)]);
                }
                ch[7] = (char)101;

                port2.Write(ch, 0, 8);
                richTextBox1.Text = serv[5].ToString();
            }

        private void Form1_FormClosed(object sender, FormClosedEventArgs e)
        {
            port.Close();
            port2.Close();
        }

    }
}
```

7.5 Video funkce testovací aplikace pro MARG

Příloha na CD