

Západočeská univerzita v Plzni

Fakulta aplikovaných věd

Katedra kybernetiky

Diplomová práce

Plzeň, 2016

Bc. Jaroslav Růžička

ZADÁNÍ DIPLOMOVÉ PRÁCE
(PROJEKTU, UMĚLECKÉHO DÍLA, UMĚLECKÉHO VÝKONU)

Jméno a příjmení: **Bc. Jaroslav RŮŽIČKA**
Osobní číslo: **A14N0158P**
Studijní program: **N3902 Inženýrská informatika**
Studijní obor: **Řídicí a rozhodovací systémy**
Název tématu: **Návrh řídicího algoritmu pro Stewartovu platformu**
Zadávací katedra: **Katedra kybernetiky**

Z á s a d y p r o v y p r a c o v á n í :

1. Navrhněte řídicí elektroniku pro Stewartovu platformu.
2. Realizujte systém pro určení orientace koncového efektoru.
3. Navrhněte a ověřte algoritmus řízení orientace koncového efektoru.

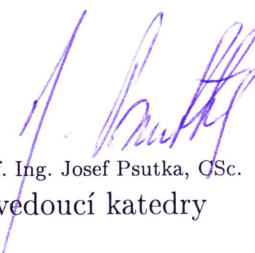
Rozsah grafických prací: **dle potřeby**
Rozsah kvalifikační práce: **40-50 stránek A4**
Forma zpracování diplomové práce: **tištěná**
Seznam odborné literatury:
Dodá vedoucí diplomové práce.

Vedoucí diplomové práce: **Ing. Miroslav Flídr, Ph.D.**
Katedra kybernetiky

Datum zadání diplomové práce: **1. října 2015**
Termín odevzdání diplomové práce: **13. května 2016**



Doc. RNDr. Miroslav Lávička, Ph.D.
děkan



Prof. Ing. Josef Psutka, CSc.
vedoucí katedry

V Plzni dne 1. října 2015

PROHLÁŠENÍ

Předkládám tímto k posouzení a obhajobě diplomovou práci zpracovanou na závěr studia na Fakultě aplikovaných věd Západočeské univerzity v Plzni.

Prohlašuji, že jsem diplomovou práci vypracoval samostatně a výhradně s použitím odborné literatury a pramenů, jejichž úplný seznam je její součástí.

V Plzni dne 13. května 2016

.....
podpis

PODĚKOVÁNÍ

Chtěl bych poděkovat panu Ing. Miroslavu Flídřovi, Ph.D., za odborné vedení, za pomoc a rady při zpracování této práce. Dále bych chtěl poděkovat panu Ing. Martinu Švejdovi za poskytnuté rady a panu Ing. Arnoldu Jágerovi za pomoc s realizací modelu Stewartovy platformy. V neposlední řadě bych chtěl poděkovat rodině za podporu během celého mého studia.

ANOTACE

Anotace:

Cíl této práce je rozšířit již existující model Stewartovy platformy o zpětno-vazební řízení koncového efektoru na základně již známe kinematiky daného modelu. Pro tuto práci je nutné vhodně rozšířit existující model a měření polohy koncového efektoru, dále je nutné navrhnout vhodnou elektroniku pro řízení plošiny. V poslední části této práce se autor bude zabývat návrhem a ověřením algoritmu řízení orientace koncového efektoru dle požadovaného zadání.

Abstract:

The main goal of this thesis is extend existing model of Stewart platform about feedback control ending effector on the base of know kinematics this model. For this thesis is necessary appropriately expand existing model and measuring position ending effector. It is also necessary design electronics for control platform. In the last part this thesis author will design and verification of algorithm of control orientation ending effector according to required task.

Klíčová slova:

Arduino Due, Raspberry Pi, Rex, regulátor, Stewartova platforma

Key words:

Arduino Due, Raspberry Pi, Rex, regulator, Stewart platform

Obsah

Obsah	IV
1 Úvod	1
2 Stewartova platforma	2
2.1 Model Stewartovy platformy	3
2.1.1 Popis rozdílů mezi ideální a realizovaným modelem Stewartovy platformy	3
2.1.2 Realizace modelu Stewartovy platformy	4
2.2 Matematický model Stewartovy plošiny	7
2.2.1 Popis matematického modelu	7
2.2.2 Popis modelu s lineárními pohoby	7
2.2.3 Převedení lineárního ramene na rotační rameno	12
2.2.4 Ověření algoritmu	13
2.3 Řídicí elektronika	13
2.3.1 Krokové motory	13
2.3.2 Řadiče krokových motorů	15
2.3.3 Ochranný obvod řadičů	20
2.3.4 Hlavní řídicí jednotka	22
2.4 Přímovazební řízení polohy koncového efektoru Stewartovy platformy	23
3 Vybraná stabilizační úloha na modelu Stewartovy platformy	28
3.1 Úloha kulička na nakloněné rovině	28
3.1.1 Matematický popis úlohy	28
3.2 Návrh regulátoru	31
4 Implementace stabilizační úlohy	35
4.1 Hardwarové potřeby k implementaci stabilizační úlohy	35
4.1.1 Řídicí hardware	36
4.1.2 Hardware pro měření polohy	36

4.1.3	Řadič dotykového panelu	38
4.2	Implementace řídicího algoritmu pomocí systému Rex	38
4.2.1	Implementace snímání polohy kuličky	42
4.2.2	Implementace PD regulátorů	46
4.2.3	Komunikace s řídicí jednotkou modelu	47
4.2.4	Vizualizace dat	53
5	Ověření implementace	54
5.1	Realizace úlohy	55
6	Závěr	59
	Příloha A	60
	Příloha B	92
	Literatura	101

Kapitola 1

Úvod

Tato diplomová práce se věnuje tématu řízení modelu Stewartovy plošiny. Tato práce navazuje na bakalářskou práci, která se věnovala návrhu a realizaci Stewartovy platformy. Stewartova platforma je jeden z mnoha druhů paralelního manipulátoru. Paralelní manipulátor, proto že koncový efektor není se základnou spojen pouze jedním spojem, ale několika v tomto konkrétním případě šesti rameny. Tradičně jsou Stewartovy platformy realizovány s využitím lineárních pohonů. V tomto konkrétním případě je lineární pohon nahrazen rotačním pohonem, jehož pohyb je převáděn na posuvný díky ramenu, které je připevněno na každý motor. Dále je odlišné uchycení k vrchní desce, které je v tomto případě otočeno tak, že každý motor je uchycen k vrchní desce ve stejném bodě, ve kterém je osa otáčení rotačního pohonu.

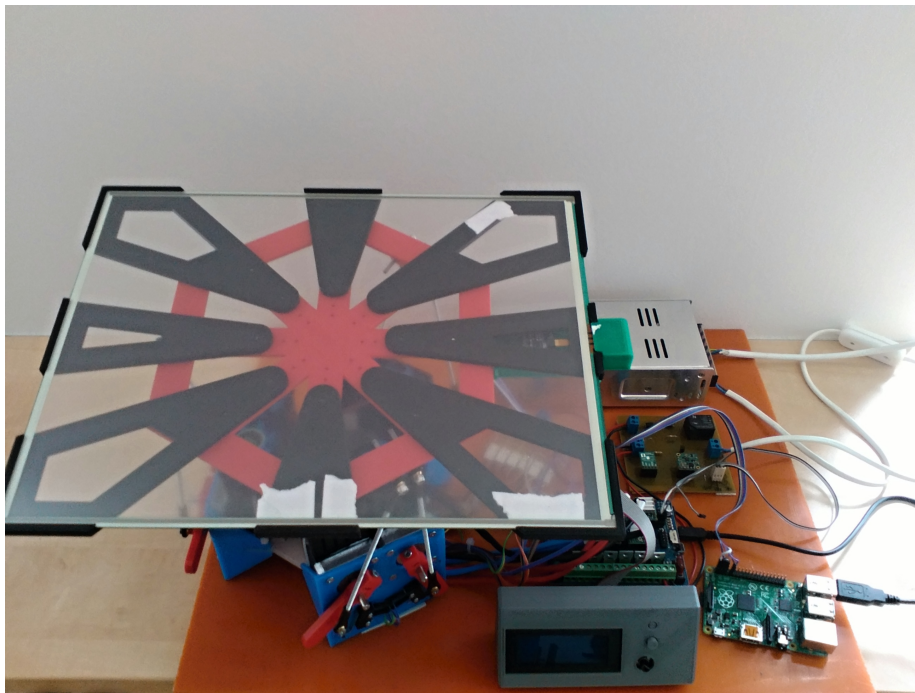
V rámci diplomové práce byl model z velké části předělán, byl nahrazen koncový efektor za takový, který umožňuje připevnit držák dotykového panelu pro stabilizační úlohu, dále umožňuje přichycení inerciální měřící jednotky. Dále byla nahrazena původní elektronika, která nevyhovovala požadavkům na uspořádání jednotlivých částí modelu. Řídicí elektronika byla navíc doplněna o ochranný obvod, který zabraňuje nechtěnému poškození řídicí elektroniky.

V rámci diplomové práce byla vybrána jedna stabilizační úloha, která byla na modelu realizována. Pro vybranou stabilizační úlohu bylo potřeba najít matematický model. Na základě matematického modelu byly určeny přenosové funkce systému, které sloužily pro návrh regulátoru v programu Matlab/ Simulink, kde byly tyto regulátory i ověřeny. Následně došlo k ověření na reálném modelu Stewartovy platformy. Aby bylo možné tuto úlohu realizovat, bylo nutné doplnit model o další hardware a software, ve kterém byl implementován řídicí algoritmus.

Kapitola 2

Stewartova platforma

Tato kapitola se bude věnovat reálnému modelu Stewartovy platformy. Nejprve bude popsán konkrétní model včetně úprav, které byly provedeny oproti předchozí práci [13]. Další část bude věnována popisu použité řídicí elektroniky, která je nutná pro fungování celého modelu a jeho přímovazebního řízení. Poslední část této kapitoly bude věnována popisu přímovazebního řízení. Bude navržen algoritmus a vysvětleny nejdůležitější části řídicího programu.



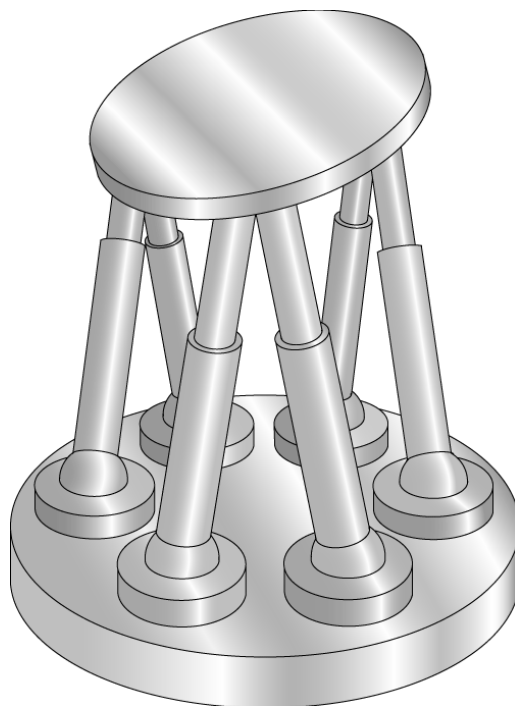
Obrázek 2.1: Model Stewartovy platformy boční pohled

2.1 Model Stewartovy platformy

Tato část práce se bude věnovat popisu modelu Stewartovy platformy. Nejprve bude popsán ideální model, jehož autoři jsou označováni jako Gough a Stewart [16], kteří jako první nezávisle na sobě popsali tento typ paralelního manipulátoru.

2.1.1 Popis rozdílů mezi ideální a realizovaným modelem Stewartovy platformy

Jak se lze dočíst v [16] původní návrh Stewartovy plošiny je realizován, jako paralelní manipulátor se šesti lineárními pohony, často hydraulickými nebo elektrickými. Tyto motory jsou připevněné v páru ve třech pozicích na základně a křížem jsou spojeny s vrchní deskou, neboli koncovým efektem. Toto spojení lze také nazvat, jako spojení do trojúhelníku. Tento model se vyznačuje tím, že má šest stupňů volnosti tj. umožňuje lineární pohyb v osách x, y a z , dále umožňuje v každé ose rotaci kolem osy. Příklad, jak takový původní model vypadá je na obrázku (obrázek 2.2).



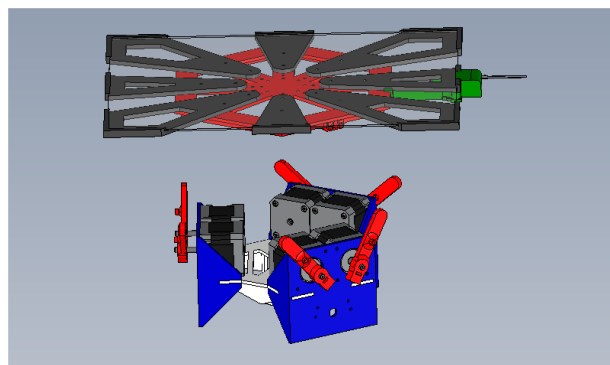
Obrázek 2.2: Ukázka lineárního modelu Stewartovy platformy

Tento ideální model nemohl být realizován, jelikož lineární aktuátory hydrau-

lické, nebo elektrické jsou příliš drahé. Proto tento pohon musel být nahrazen něčím dostupnějším a to je pohyb rotační. Rotačních motorů se vyrábí celé řada např. servo pohony, krokové motory a podobně. Aby bylo docíleno pohybu koncového efektoru je nutné rotační pohyb převádět na lineární, k tomuto účelu slouží rameno, které je na jednom konci spojeno s osou otáčení rotačního aktuátoru a na druhém konci je spojeno přes čep s tyčí, která je spojena s koncovým efektořem. Toto řešení umožňuje lineární pohyb, jaký by vykonaly lineární aktuatory. Tento lineární pohyb není v celém rozsahu, ale jen pouze v části a závisí na délce ramene a tyči, které spojuje pohon s koncovým efektořem. Stejně jako předloha i tento model má na základně umístěny motory v páru. Celkový počet párů je tři, což umožňuje zachování vlastnosti stupňů volnosti Stewartovy platformy. Oproti klasickému návrhu je jinak uspořádané uchycení vrchní desky, jak již bylo zmíněno v úvodu práce.

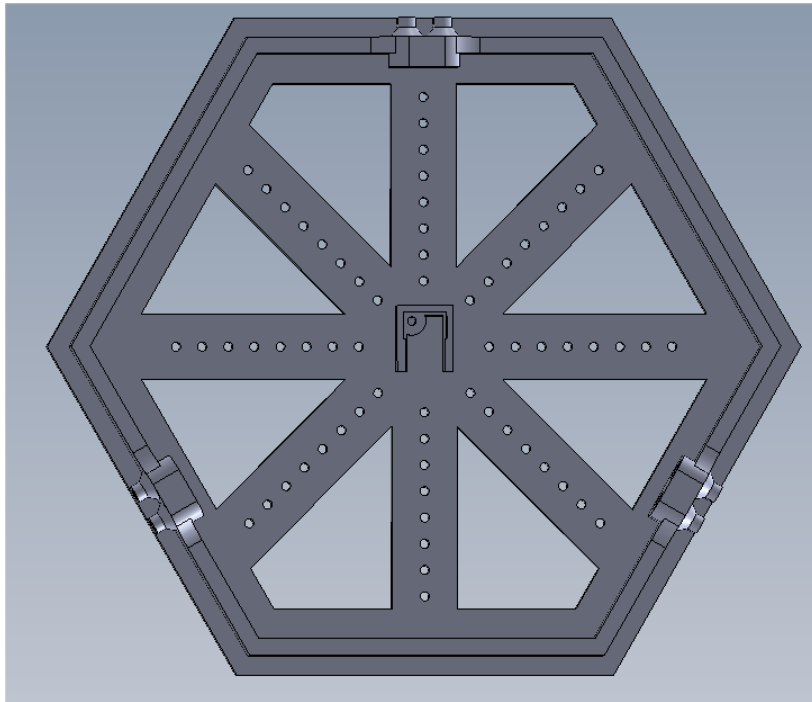
2.1.2 Realizace modelu Stewartovy platformy

Tento model Stewartovy platformy byl realizován již v předešlé práci, kde se jednalo o prototyp modelu. Jak se později ukázalo, bylo nutné tento model přepracovat, jelikož nevyhovoval požadavkům na další použití. Původní model byl převzat z projektu [4], který má volně dostupný technický výkres modelu. Tento výkres byl využit jako předloha pro tento model. Jelikož jsou používány rozdílné krokové motory, tak se musel tento model upravit, tak aby se do základny motory vešly. Dále byl upraven koncový efektoř. Původní návrh nepočítal s možností připevnění dalších komponent, proto byl přepracován tak, aby umožňoval ze zdola umístit inerciální měřící jednotku, které by určovala polohu koncového efektoru a zároveň umožňovala uchycení dotykového panelu, který je využit ve zpětnovazební úloze, jako senzor měření polohy kuličky na ploše koncového efektoru. Vizualizaci návrhu lze vidět na následujícím obrázku obrázek 2.3.



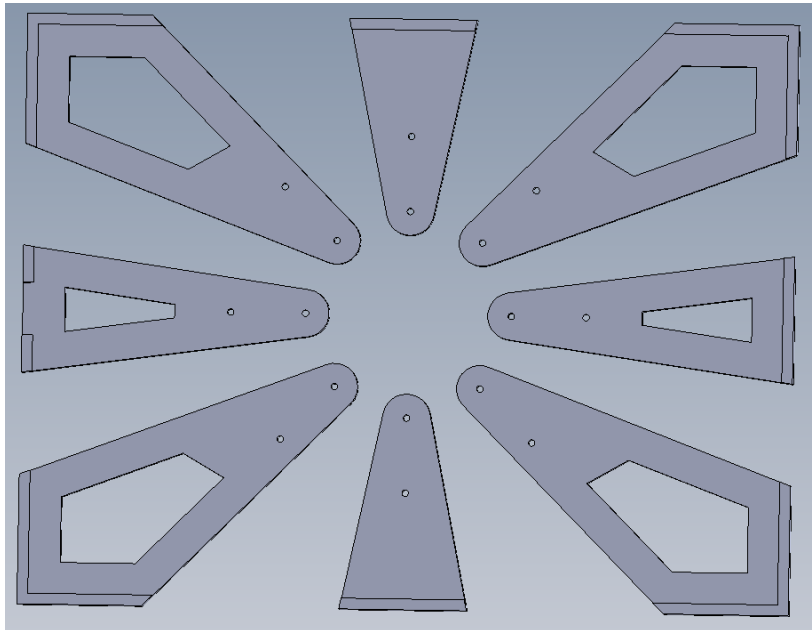
Obrázek 2.3: 3D návrh modelu Stewartovy platformy

Na 3D návrhu není vidět spojení mezi jednotlivými krokovými motory a vrchním dílem, jelikož pro spojení byla použita obyčejná závitová tyč a modelářské čepy, které jsou běžně dostupné. Na obrázku jsou vidět všechny části, které byly vytištěny na 3D tiskárně a nyní budou detailněji popsány. Základna se neliší od té, které byla použita už na prototypu, proto ji není potřeba detailně popisovat. Vrchní díl se skládá ze třech částí. První část je samotný koncový efektor na obrázku vidět červeně.



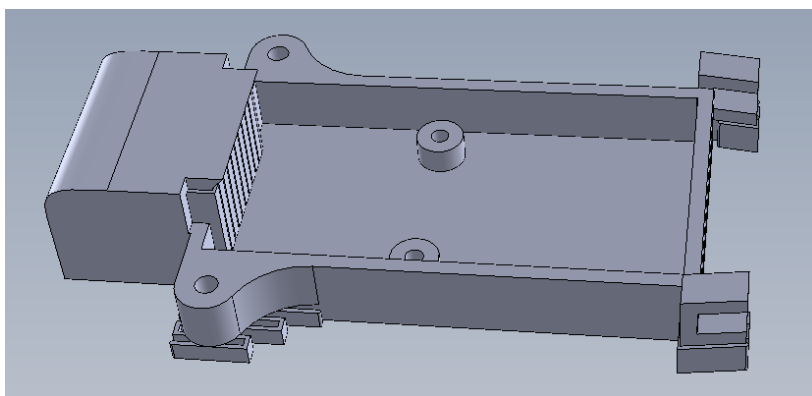
Obrázek 2.4: Vrchní díl Stewartovy platformy

Z tohoto pohledu je vidět, jako je rozvrženo spojení vrchní desky s krokovými motory a kde je plánováno umístění inerciální měřicí jednotky, které je ve středu desky. Další novou částí, která je přidána na model Stewartovy platformy, je držák dotykového panelu, jehož 3D návrh je vidět na obrázku 2.5.



Obrázek 2.5: 3D návrh držáku dotykového panelu

Samotný držák se skládá z osmi dílů, které mají připravené otvory, pomocí kterých lze držák připevnit ke koncovému efektoru Stewartovy platformy. Dále má držák na každém dílu udělanou drážku, ve které je zafixován dotykový panel. Na jednom dílu je vidět i drážka, která je určena pro kabel, který je vyveden z dotykového panelu. Kabel dotykového panelu spolu s řadičem dotykového panelu je schován v krytu, který je na obrázku 2.3 vidět zelenou barvou. Detailnější návrh je vidět na následujícím obrázku 2.6.



Obrázek 2.6: 3D návrh držáku dotykového panelu

Tento držák byl též vytištěn na 3D tiskárně. Uvnitř držáku je umístěn řadič do-

tykového panelu, který zpracovává údaje o místě dotyku předmětu na dotykovém panelu. Kryt je dvěma samo řeznými šrouby spojen s držákem dotykového panelu. Tento kryt má též ochranou funkci, jelikož dotykový kabel je osazen páskovým kabelem, se kterým se musí zacházet velmi obezřetně, aby nedošlo k jeho poškození, proto je lepší, když je kabel zakrytý a není k němu přístup.

2.2 Matematický model Stewartovy plošiny

V této části bude popsán matematický popis Stewartovy plošiny. Matematický popis byl odvozen v bakalářské práci [13], proto se zde autor nebude věnovat celému odvození. Odvození z bakalářské práce bylo znovu ověřeno a bylo dosaženo stejných výsledků.

2.2.1 Popis matematického modelu

V této části budou popsány vztahy, které popisují pohyb plošiny. Tyto vztahy lze v obecném případě odvozovat dvěma způsoby: přímou geometrickou úlohou, kdy se podle kloubových souřadnic vypočítává pozice koncového efektoru, nebo pomocí inverzní geometrické úlohy, kdy je známa poloha koncového efektoru a dopočítávají se kloubové souřadnice. Tuto úlohu lze řešit pomocí inverzní geometrické úlohy. Pomocí přímé geometrické úlohy ji nelze řešit, jelikož je to tak komplikovaný systém, že nelze najít analytické řešení.

Tato část bude rozdělena na dvě části. V první části budou odvozeny vztahy pro lineární aktuátory Stewartovy plošiny. Druhá část se bude zabývat převodem lineárního pohonu na rotační.

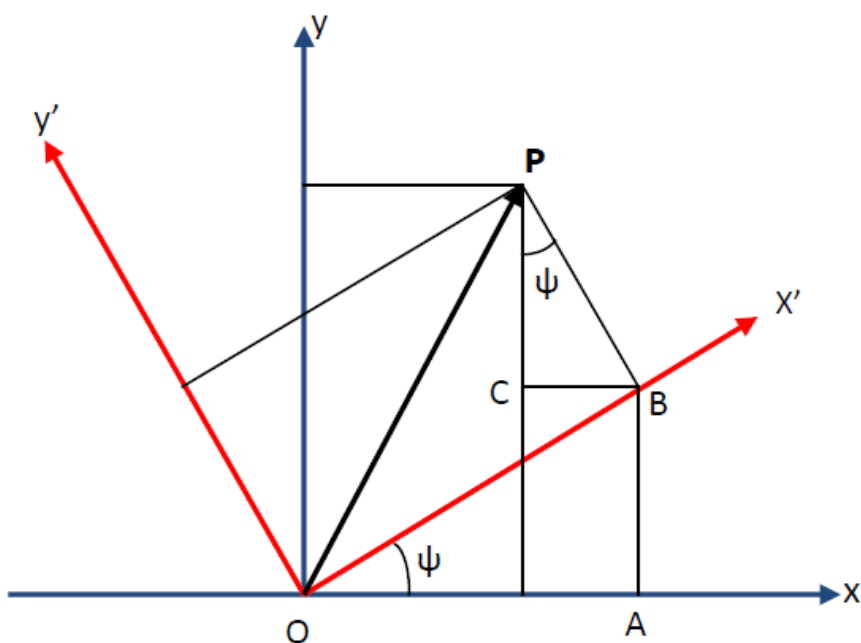
2.2.2 Popis modelu s lineárními pohoby

Pro výpočty jsou zavedeny následující předpoklady:

- plošina je složena ze dvou desek, které jsou spolu spojeny celkem 6 rameny
- základna je popsána referenční soustavou souřadnic x, y a z , která je ve středu plošiny.
- vrchní desce je přiřazena vlastní soustava souřadnic x', y' a z' , která je uprostřed vrchní desky
- původní souřadnice základny lze pomocí 3 transformačních matic vyjádřit v souřadnicích vrchní desky

- rotace okolo třech os vrchní desky vůči základně lze vyjádřit následujícími třemi Eulerovskými úhly:
 - rotace o úhel Ψ okolo osy z se nazývá vybočení
 - rotace o úhel Θ okolo osy y se nazývá stoupání
 - rotace o úhel Φ okolo osy x se nazývá vytočení

Nyní budou popsány rotace, které transformují souřadnice bodu z jednoho souřadného systému do druhého soustavného systému.



Obrázek 2.7: Znáznornění transformace úhlů z jedné soustavy souřadnic z do soustavy souřadnic z' .

Na obrázku jsou vidět dvě soustavy souřadnic, které jsou vůči sobě posunuté o úhel Ψ . Na obrázku jsou vidět body, které slouží pro výpočet transformace z jedné souřadné soustavy do druhé. Bod O je střed soustav, bod P je bod, který má své souřadnice jak v soustavě x, y tak v soustavě x', y' , body C, B, P jsou body trojúhelníku, které popisují bod P v soustavě x', y' , a bod A slouží k určení velikosti natočení soustavy souřadnic x', y' od soustavy x, y . Nyní musíme najít vhodné vztahy pro popis souřadnic bodu P v souřadném systému X', Y' .

$$P = i'x' + j'y' + k'z' = ix + jy + kz \quad (2.1)$$

Tato rovnice říká, že bod P lze vyjádřit v soustavě x, y a x', y' pomocí jejich vektorových složek i, j, k a i', j', k' . Nyní musíme najít vztahy pro x, y a z .

Dále lze vyjádřit x, y a z :

$$x = OA - BC = x' \cos \Psi - y' \sin \Psi \quad (2.2)$$

$$y = AB + PC = x' \sin \Psi + y' \cos \Psi \quad (2.3)$$

$$z = z' \quad (2.4)$$

Nyní pro přehlednost bude zavedena rotační matice $R_z(\Psi)$, používá se toto označení, které říká, že jde o rotaci kolem osy z o úhel Ψ :

$$\begin{bmatrix} x \\ y \\ z \end{bmatrix} = R_z(\Psi) \begin{bmatrix} x' \\ y' \\ z' \end{bmatrix} \quad (2.5)$$

kde

$$R_z(\Psi) = \begin{pmatrix} \cos \Psi & -\sin \Psi & 0 \\ \sin \Psi & \cos \Psi & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad (2.6)$$

Stejným výpočtem lze dojít k rotační matici kolem osy y o úhel Θ , která se označuje $R_y(\Theta)$:

$$R_y(\Theta) = \begin{pmatrix} \cos \Theta & 0 & \sin \Theta \\ 0 & 1 & 0 \\ -\sin \Theta & 0 & \cos \Theta \end{pmatrix} \quad (2.7)$$

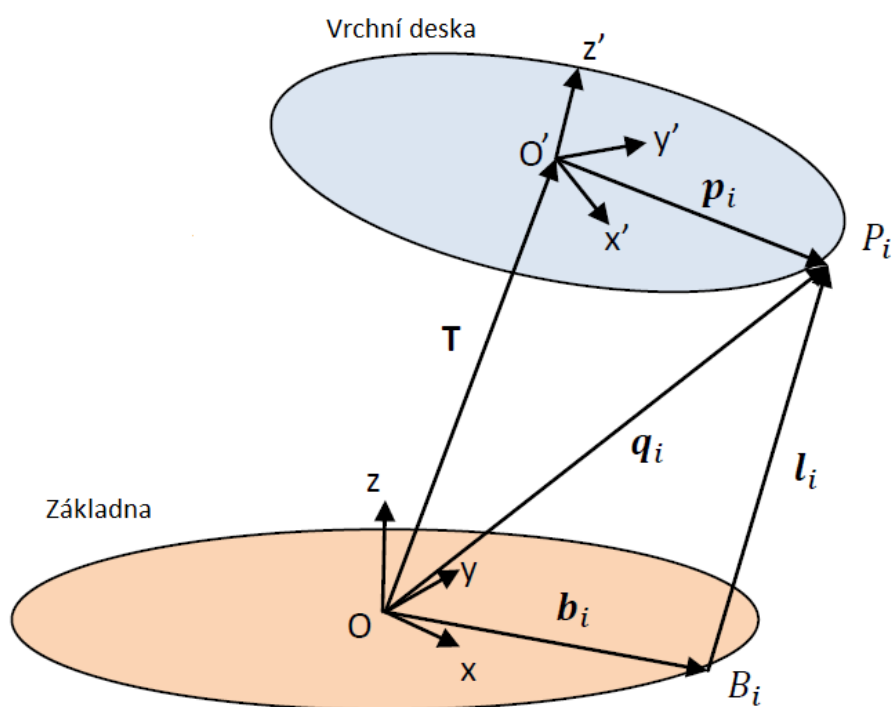
Stejným výpočtem lze dojít k rotační matici kolem osy x o úhel Φ , která se označuje $R_x(\Phi)$:

$$R_x(\Phi) = \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \Phi & -\sin \Phi \\ 0 & \sin \Phi & \cos \Phi \end{pmatrix} \quad (2.8)$$

Nyní pomocí odvozených rotačních matic pro každou osu (2.6) – (2.8) lze určit matici ${}^P R_B$, která popisuje kompletní transformaci z jedné soustavy souřadnic do druhé soustavy souřadnic.

$$\begin{aligned}
{}^P\mathbf{R}_B &= \mathbf{R}_z(\Psi) * \mathbf{R}_y(\Theta) * \mathbf{R}_x(\Phi) = & (2.9) \\
&\begin{pmatrix} \cos \Psi & -\sin \Psi & 0 \\ \sin \Psi & \cos \Psi & 0 \\ 0 & 0 & 1 \end{pmatrix} * \begin{pmatrix} \cos \Theta & 0 & \sin \Theta \\ 0 & 1 & 0 \\ -\sin \Theta & 0 & \cos \Theta \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \Phi & -\sin \Phi \\ 0 & \sin \Phi & \cos \Phi \end{pmatrix} \\
&= \begin{pmatrix} \cos \Psi \cos \Theta & -\sin \Psi \cos \Theta & \sin \Psi \sin \Theta \\ \sin \Psi \cos \Theta & \cos \Psi \cos \Theta & \cos \Psi \sin \Theta \\ -\sin \Theta & 0 & \cos \Theta \end{pmatrix} * \begin{pmatrix} 1 & 0 & 0 \\ 0 & \cos \Phi & -\sin \Phi \\ 0 & \sin \Phi & \cos \Phi \end{pmatrix} \\
&= \begin{pmatrix} \cos \Psi \cos \Theta & -\sin \Psi \cos \Phi + \cos \Psi \sin \Theta \sin \Phi & \sin \Psi \sin \Phi + \cos \Psi \sin \Theta \cos \Phi \\ \sin \Psi \cos \Theta & \cos \Psi \cos \Phi + \sin \Psi \sin \Theta \sin \Phi & -\cos \Psi \sin \Phi + \sin \Psi \sin \Theta \cos \Phi \\ -\sin \Theta & \cos \Theta \sin \Phi & \cos \Theta \cos \Phi \end{pmatrix}
\end{aligned}$$

Nyní, když jsou definovány rotační matice v každé ose z jednoho souřadného systému do druhého souřadného systému, bude představen popis Stewartovy plošiny.



Obrázek 2.8: Popis i-tého ramene Stewartovi plošiny

Na obrázku 2.8 je vidět, jak se vypočítá délka ramene l_i . Na obrázku jsou vidět významné body, které slouží k výpočtu: b_i je vektor vzdálenosti mezi středem soustavy souřadnic základny a přípojného bodu B_i , B_i je vektor přípojného bodu ramene na základně, T je matice posunutí vrchní desky od základny, P_i je přípojný bod ramene na vrchní desce, vektor q_i určuje vzdálenost přípojného bodu P_i od středy soustavy souřadnic základny, vektor p_i je vektor, který určuje vzdálenost mezi přípojným bodem P_i a soustavou souřadnic označenou O' . Vektor q_i je vzdálenost bodu P_i od počátku vztažné soustavy O . Vektor q_i vyjadřuje následující rovnice:

$$q_i = T + {}^P R_B \cdot p_i \quad (2.10)$$

Kde T označuje vektor translace, který udává poziční posunutí vrchní desky vůči základně. Vektor p_i definuje souřadnice od středu soustavy souřadnic vrchní desky a přípojného bodu P_i .

Podobně lze určit vztah pro l_i rameno:

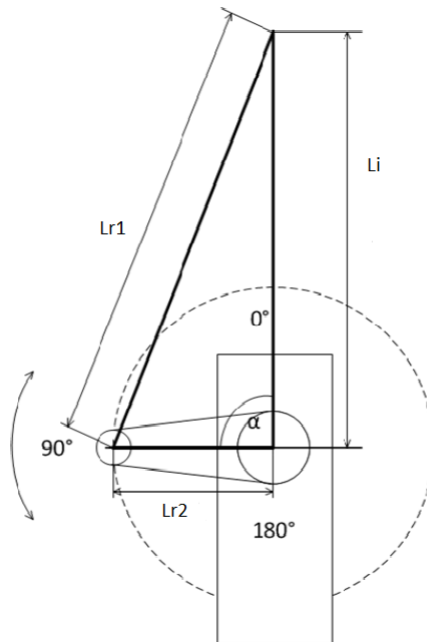
$$l_i = T + {}^P R_B \cdot p_i - b_i \quad (2.11)$$

Kde b_i je vektor, který definuje dolní přípojný bod B_i . Těchto 6 rovnic dohromady dávají délky všech 6 ramen pro požadovanou pozici a natočení vrchní desky. Tyto rovnice byly převzaty ze zdroje [15].

2.2.3 Převod lineárního ramene na rotační rameno

Tato část bude věnována odvození vztahů pro jednotlivá ramena Stewartovi plošiny, kdy je jako aktuátor použit krokový motor, který realizuje rotační pohyb ramene.

Pokud Stewartova plošina má lineární aktuátory, pak se mění délka ramene každého aktuátoru. Jelikož tento model má k dispozici rotační aktuátory, které mají pevnou délku ramen a páky od středu otáčení, probíhá přepočítání z lineární délky na úhel natočení páky krokového motoru. Na následujícím obrázku je uveden postup.



Obrázek 2.9: Převod lineárního pohybu na rotační.

Pohon je tvořen krokovým motorem, který se může v celém rozsahu otáčet o 360° , ale v této aplikaci není využit celý jeho rozsah, jelikož rozsah závisí na možném otáčení ramena. Rameno L_{r2} se může otáčet o úhel α v rozmezí od 0° do 180° a má svoji pevně danou velikost. Délka spojovacího ramene L_{r1} je též pevně dána, délka ramene L_i je známa z inverzní kinematické úlohy. Zbývá určit úhel natočení α rotačního pohonu, který se určí analyticky pomocí kosinové věty, podle následující rovnice:

$$L_{r1}^2 = L_{r2}^2 + L_i^2 - 2L_i L_{r2} \cdot \cos \alpha \quad (2.12)$$

úhel α

$$\alpha = \arccos \frac{-L_{r1}^2 + L_{r2}^2 + L_i^2}{2L_i L_{r2}} \quad (2.13)$$

2.2.4 Ověření algoritmu

Ověření algoritmu probíhalo v bakalářské práci [13] pomocí programu Matlab/Simulink. Nyní je algoritmus implementován přímo v řídicí elektronice o které pojednává následující část. Tato změna umožňuje snadnější obsluhu modelu Stewartovy platformy a přináší uživatelské usnadnění zadávání hodnot do řídicí elektroniky.

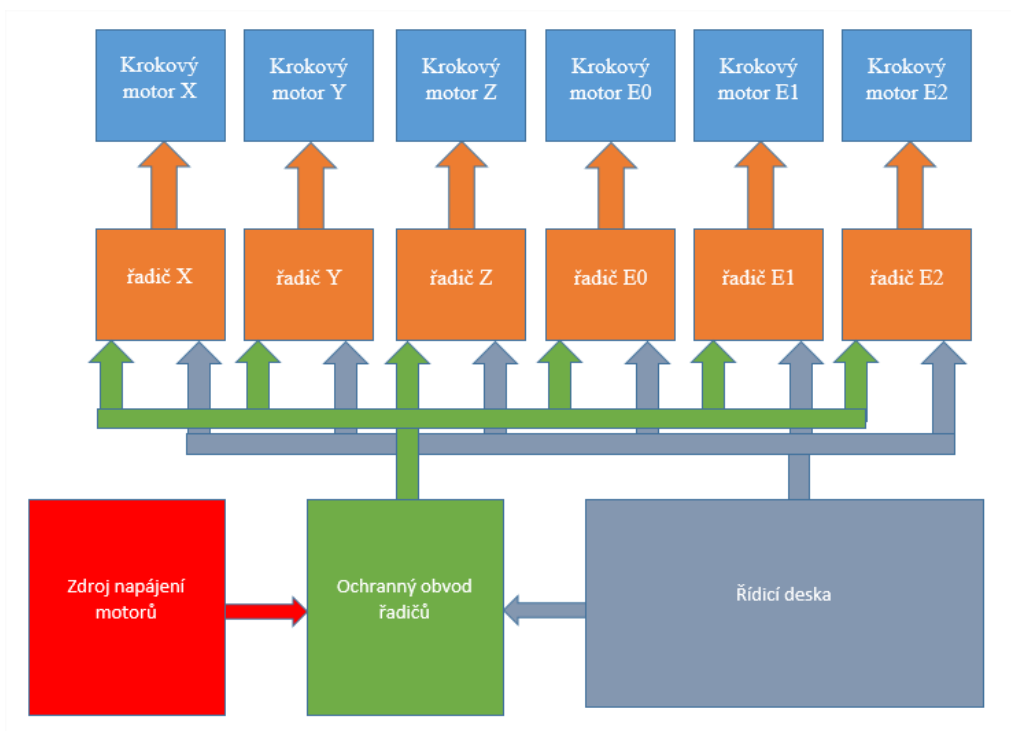
2.3 Řídicí elektronika

Tato část práce se bude věnovat návrhu řídicí elektroniky pro Stewartovu platformu, dále zde bude popsáno, jaké další elektronické komponenty byly použity v tomto modelu. Na úvod bude uvedeno a popsáno schéma řídicí elektroniky. Následně budou popsány jednotlivé části z tohoto schématu. Existující model měl řídicí elektroniku, která nevyhovovala dalšímu použití, proto byla nahrazena jinou, která splňovala požadavky na realizaci přímovazebního řízení.

Na obrázku 2.10 je vidět idealizované schéma zapojení elektroniky pro model Stewartovy elektroniky. Toto idealizované schéma bylo použité při realizaci elektroniky. Celé schéma se skládá z pěti důležitých částí pro fungování modelu Stewartovy platformy. Jednotlivé části budou detailně popsány v příslušných částech, zde bude uveden pouze výčet těchto prvků. Základem modelu je rotační aktuátor v tomto konkrétním případě byly zvolené krokové motory. Aby mohly krokové motory fungovat potřebují řadiče, které generují signály pro řízení krokových motorů. Řadiče krokových motorů mají dvojí napájení, které se musí spouštět v daném pořadí, jinak hrozí poškození řadičů, proto schéma zobrazuje i ochranný obvod řadičů. Nezbytnou součástí je i zdroj napájení, který bude též popsán. Nejdůležitější částí elektroniky Stewartovy platformy je řídicí deska, která zpracovává požadavky od uživatele a předává požadované natočení krokových motorů do příslušných řadičů.

2.3.1 Krokové motory

Jak už bylo napsáno výše v tomto modelu Stewartovy platformy, byly jako rotační aktuátory zvoleny krokové motory. Krokový motor je synchronní točivý stroj většinou napájený impulsem stejnosměrného proudu. Magnetické pole je



Obrázek 2.10: Idealizované schéma zapojení elektroniky modelu

generováno postupným napájením jednotlivých polových dvojic. Pohyb rotoru krokového motoru je při nízkých rychlostech nespojitý, rotor se pohybuje mezi stabilními polohami vždy v určitém úhlu – mluvíme o pohybu v krocích. Počet kroků (stabilních klidových poloh) je dán počtem pólových dvojic, rovněž může být ovlivněn způsobem ovládání. K pohybu tohoto motoru je vždy třeba řídicí elektronika – ovladač krokového motoru. K mechanickému kontaktu a tudíž otěru nedochází u krokových motorů jinde než v ložiscích. Vyznačují se proto velkou mechanickou odolností, dlouhou dobou života a provozem téměř bez údržby. Nevýhodou krokových motorů je tzv. ztráta kroku, která nastává při překročení mezního zatížení a sklon k mechanickému zakmitávání, které může vést k nestabilitě při pohybu. Obě tyto negativní vlastnosti lze předem vyloučit volbou vhodného motoru a ovladače s přihlédnutím k momentovým charakteristikám pohonu [18].

V tomto modelu byly použité krokové motory od prodejce Pololu typu MEMA 17 [7]. Jedná se o hybridní motor tj. může být použit, jako unipolární, nebo bipolární. V této práci je využit, jako bipolární, což je dáno použitým typem řadiče. Krokový motor se v jednom kroku otočí o úhel 1.8° , na celou otočku potřebuje dvě stě kroků. Podrobnosti o použitých řadičích budou uvedeny v následující sekci.



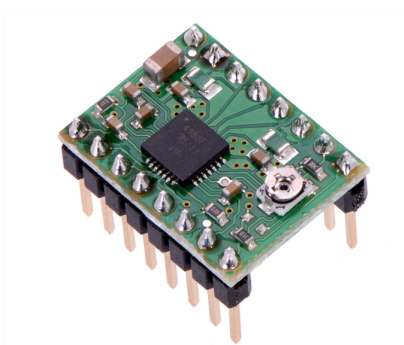
Obrázek 2.11: Krokový motor MEMA 17

V rámci testování Stewartovy platformy se ukázalo, že se motory značně přehřívají, proto byl na každý motor umístěn chladič, který je běžně dostupný v prodejnách počítačových komponentů, kde se využívá na chlazení čipů základní desky počítače. Tyto aktivní chladiče odvádějí teplo, které vzniká během činnosti krokových motorů.

2.3.2 Řadiče krokových motorů

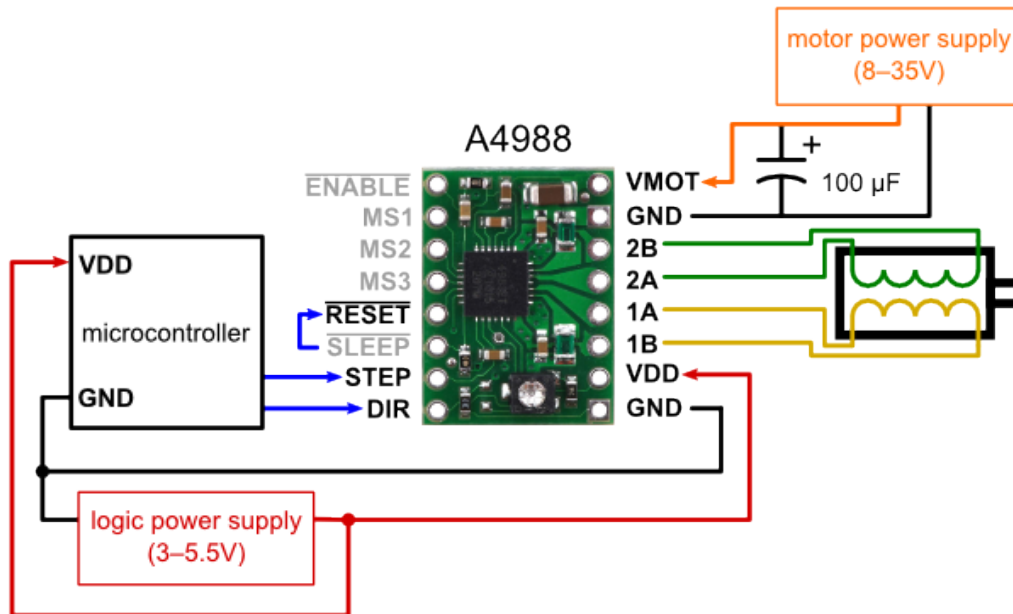
Tato část bude věnována použitým řadičům. Jak už bylo uvedeno v předchozí sekci každý krokový motor vyžaduje řadič, který generuje výstupní signál, tak aby došlo k otočení krokového motoru. Na trhu existuje spousta druhů řadičů krokových motorů ať už se jedná o samotné čipy, které je nutné ještě dovybavit, tak aby se mohly používat, nebo již hotové řadiče pro běžné použití, ale dají se sehnat i průmyslové jednotky. V tomto modelu jsou použity již hotové moduly, které kromě samotného řadiče obsahují i základní obvod, zaručující jeho správnou funkci. V aktuální verzi řídicí elektroniky modelu byly zaměněny řadiče

krokových motorů oproti prototypu. V prototypu se používaly řadiče DRV8825, které byly nahrazeny řadiči A4988 [10]. Tyto řadiče jsou rozměrově stejné mají i stejné zapojení rozdíl mezi nimi je v maximálním proudu, které jsou schopny dodat do krokového motoru. Původní řadiče mohly dodávat proud o velikosti 2.2 A, zatímco řadiče A4988 jsou doporučované používat maximálně při proudu 1 A. Jelikož při větším proudu dochází k většímu zahřívání řadiče, což může vést k jeho poškození. V této aplikaci není toto omezení nijak limitující, proto došlo k záměně používaných řadičů.



Obrázek 2.12: Použitý řadič pololu a4988

Na obrázku 2.12 lze vidět, jak vypadá používaný řadič. Tento řadič byl ještě dodečně vybaven chladičem na čip, jelikož se během testování ukázalo, že při delší době používání dochází k zahřívání čipu, tak se pro jeho ochranu připevnil na jeho tělo chladič, který odvádí vzniklé teplo.



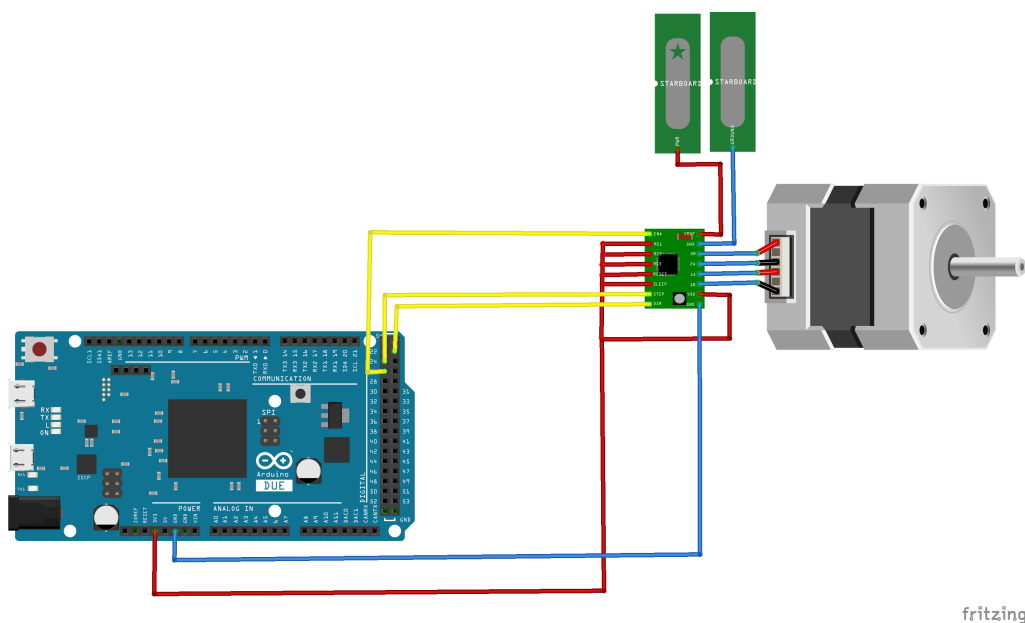
Obrázek 2.13: Zapojení řadiče A4988 dle výrobce.

Na obrázku 2.13 je vidět minimální doporučené zapojení od výrobce. Na pravé straně řadiče je výkonová část tj. nahoře vstupní napájení pro krokový motor v tomto případě +20V. Toto napájení prochází přes kondenzátor o hodnotě $100 \mu F$. Tento kondenzátor má filtrační funkci a odstraňuje špičky, které se mohou vytvářet při zapnutí/vypnutí vstupního napájení. Dále jsou vidět výstupní piny, ke kterým se připojuje krokový motor. Na poslední dva piny je přivedeno logické napájení, dle zvoleného kontroléru v tomto případě +3.3V, dle referenčního napájení hlavní řídicí jednotky. Tato jednotka dále posílá informace o směru točení krokového motoru na vstupní pin DIR, dále počet kroků na vstupní pin STEP a na vstupní pin \overline{ENABLE} informaci o povolení běhu řadiče. Piny \overline{RESET} a \overline{SLEEP} jsou rovnou přivedeny k logické nule, jelikož není žádoucí řadič resetovat a uspávat. Dále jsou na levé straně piny MS1, MS2 a MS3, které slouží k mikrokrokování dle následující tabulky tabulka(2.1). Z této tabulky je zřejmé,

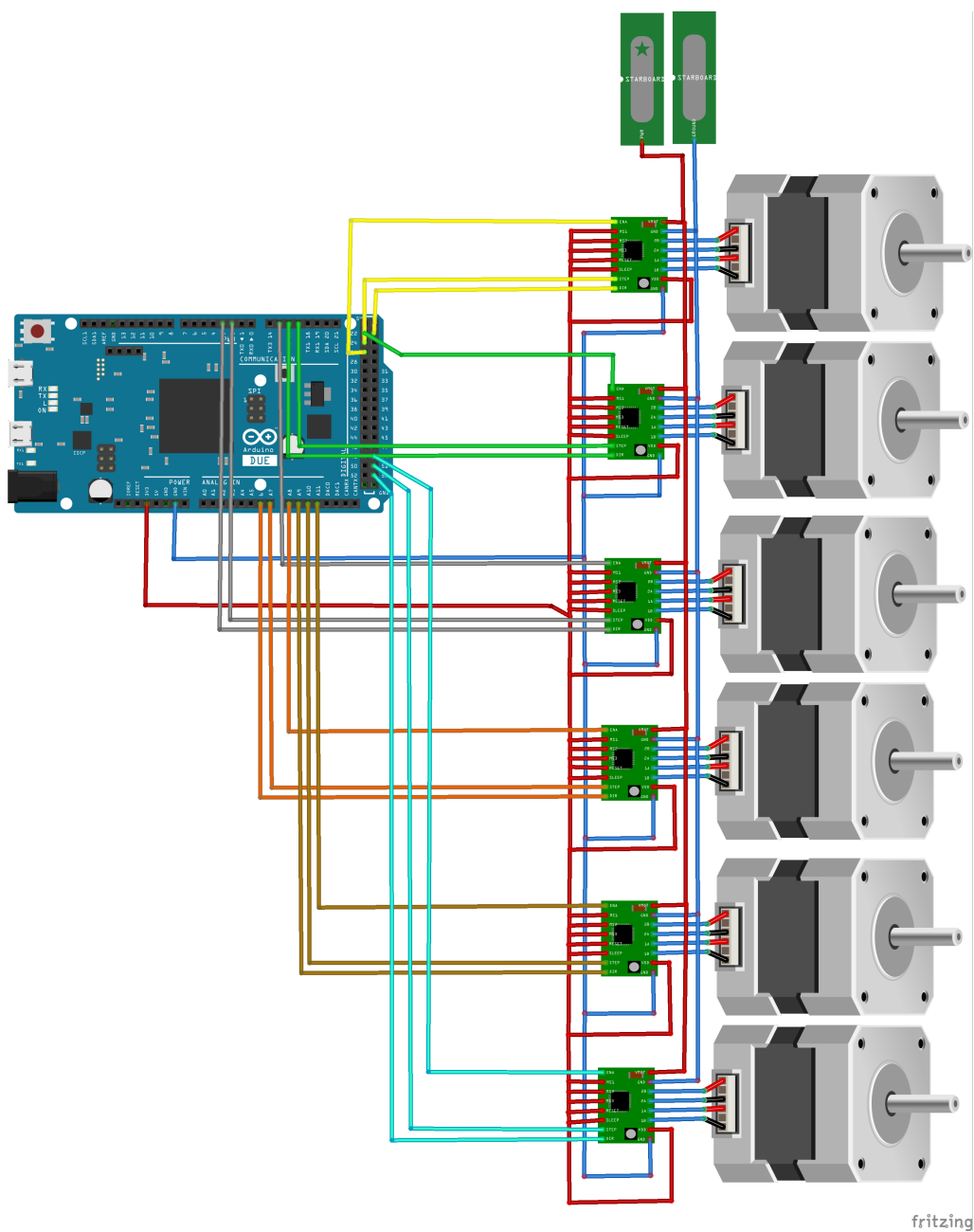
MS1	MS2	MS3	Krok
LOW	LOW	LOW	1 krok
HIGH	LOW	LOW	1/2 kroku
LOW	HIGH	LOW	1/4 kroku
HIGH	HIGH	LOW	1/8 kroku
HIGH	HIGH	HIGH	1/16 kroku

Tabulka 2.1: Nastavení mikrokrokování

že řadič může pracovat s mikrokroky. Tímto lze docílit přesnějšího pohybu motoru, čehož autor využil při návrhu přímovazebního řízení. Konkrétně využil nejpresnějšího nastavení tj. 1/16 kroku.



Obrázek 2.14: Zapojení řadiče, krokového motoru a řídicího členu



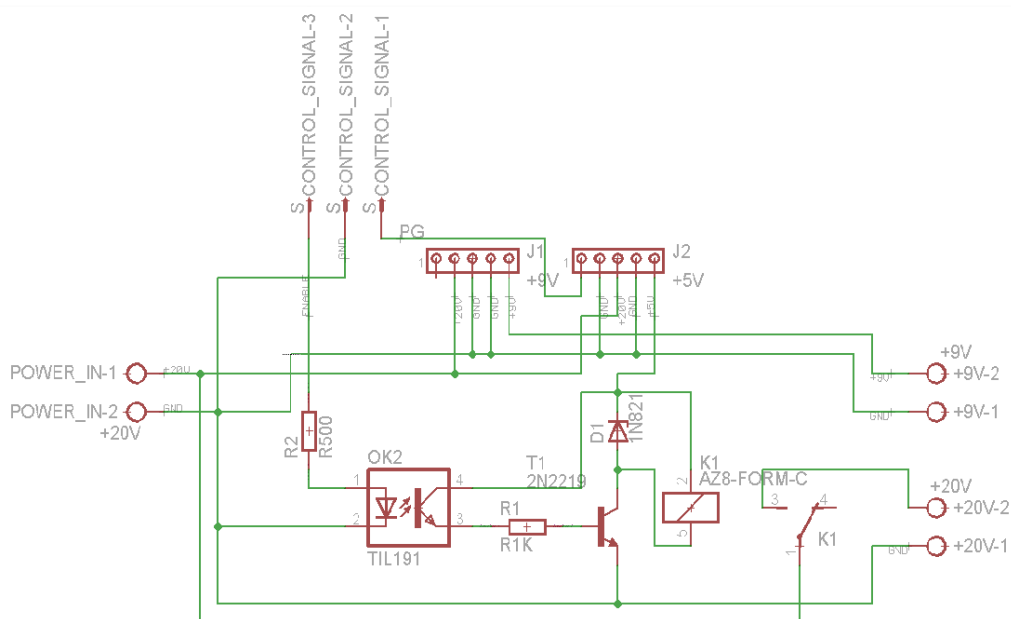
Obrázek 2.15: Kompletní zapojení řadičů krokových motorů modelu.

Na obrázcích 2.14 a 2.15 je vidět zapojení řídicího členu Arduino DUE, řadičů krokových motorů a samotných krokových motorů. Nejprve je zobrazen detail, na kterém je vidět pouze jeden řadič s krokovým motorem a poté je uvedeno celkové zapojení. Na obrázcích je patrné, že je nutné použít velké množství vodičů,

kteřé je nutné použít pro zapojení. Také je vidět, že pokud se zapojení realizuje pomocí kontaktního nepájivého pole, tak musí být uživatel velmi obezřetný při zapojování, jelikož například prohození vodičů plus a mínus může vést ke zničení veškeré elektroniky. Tyto problémy lze odstranit tak, že místo kontaktního nepájivého pole se použije přídatná deska (tzv. shield) pro platformu Arduino, která je určena pro osazení řadičů krokových motorů a usnadňuje jejich připojení k řídicí jednotce. Jedna z mnoha dostupných desek je deska RADD5 [8], která byla použita v této práci.

2.3.3 Ochranný obvod řadičů

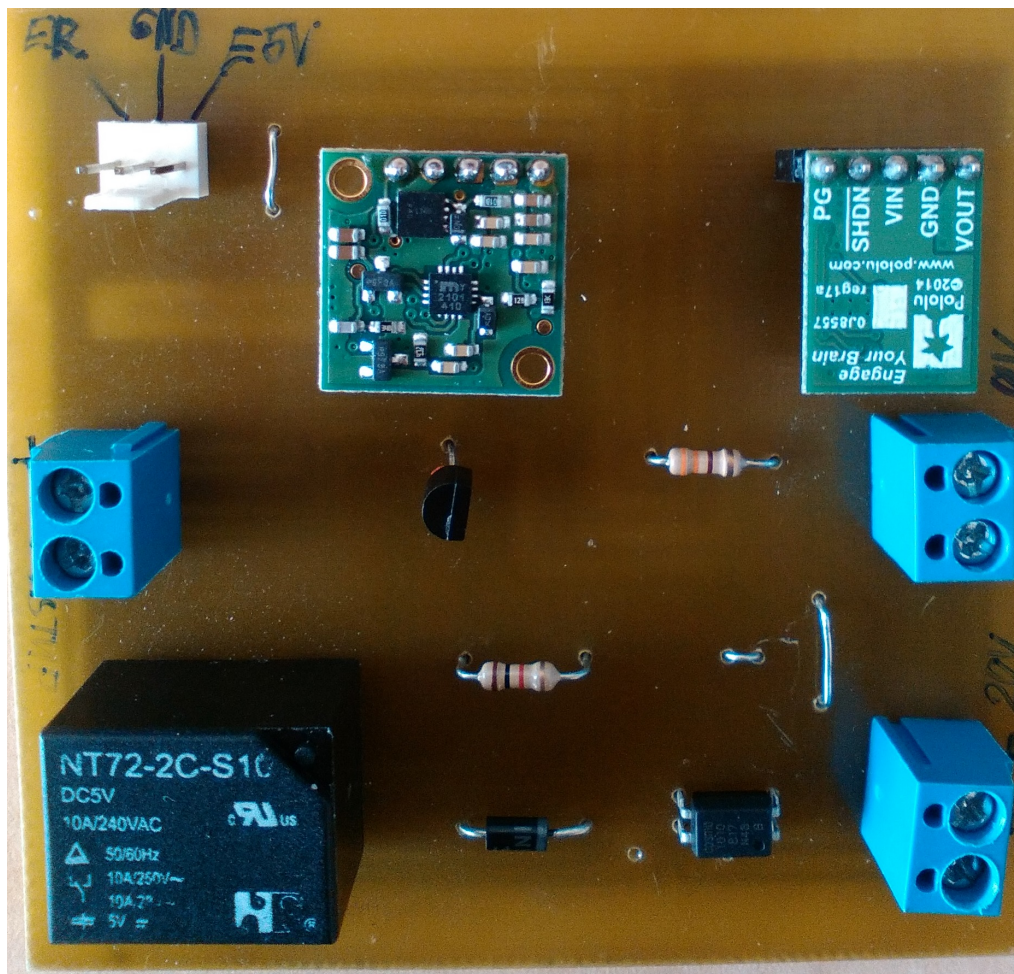
Tento obvod byl do modelu přidán později po zničení první desky pro řadiče krokových motorů. Kdy z neznámých příčin došlo k zničení sady řadičů a desky samotné. Tento spínací obvod má za úkol předcházet podobným incidentům, jelikož spínání řídí Arduino Due, teprve poté co je dokončena inicializace řídicího programu, je softwarově povoleno napájení pro krokové motory. Schéma obvodu spínacího relé je následujícím obrázkem 2.16.



Obrázek 2.16: Schéma spínacího obvodu

Nahoře jsou vidět připravené patice pro regulátory napětí. V této desce jsou použity dva, první je vlevo a převádí napájení +20V na +9V, které slouží pro napájení

chladičů krokových motorů. Vedle je regulátor, který z +20V dělá +5V voltů a slouží pro napájení cívky relé. Spínací svorky relé jsou řízeny z Arduina Due, jelikož pracujeme s jinou logickou úrovní, tak je v obvodu ještě využit optočlen, který galvanicky odděluje tyto dvě napájení. Pro ochranu je ještě použita Zenerova dioda a tranzistor, přes které dochází ke spínání relé obvodu.

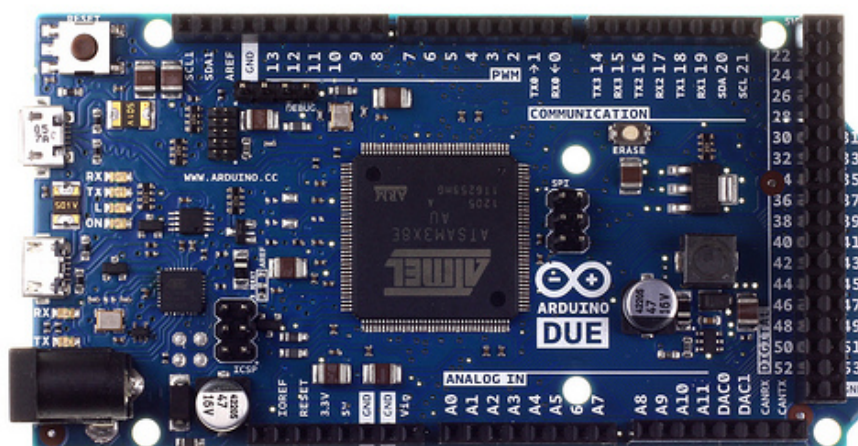


Obrázek 2.17: Realizace spínacího obvodu.

Na obrázku 2.17 je vidět hotový spínací obvod na desce plošných spojů včetně osazení jednotlivými součástkami. Regulátory napětí byly zvoleny výroby firmy Pololu konkrétně se jedná o modely: D24V25F9 a D24V10F5. První jmenovaný je vidět na obrázku vlevo a jedná se o regulátor z +20V na +9V. Druhý regulátor má výstupní napětí +5V a je určen pro napájení cívky relé, jak bylo již zmíněno výše. Použitý optočlen je COSMO K1010 [6].

2.3.4 Hlavní řídicí jednotka

Tato část se bude věnovat popisu nejdůležitější části elektroniky celého modelu a to je hlavní řídicí jednotka. V prototypu modelu Stewartovy platformy bylo využita deska od společnosti Texas Instrumental, konkrétně se jednalo o vývojovou desku Stellaris Launchpad. Tato vývojová deska měla nevýhodu, jelikož je dražší a není tolik rozšířená jako desky od společnosti Arduino, že pro ni nebyla možnost koupit hotovou desku, na kterou by šlo umístit řadiče krokových motorů. Proto byla tato vývojová deska nahrazena vývojovou deskou Arduino DUE [3]. Tato vývojová deska má výhodou oproti dalším deskám založených na platformě Arduino, že její mikrořadič je postaven na 32 bitové architektuře ARM. Konkrétně obsahuje mikrořadič Atmel SAM3EX8E ARM Cortex-M3 CPU, jehož CPU je taktováno s frekvencí 84MHz. Další výhodou oproti ostatním deskám z rodiny Arduino je, že obsahuje velké množství vstupně/ výstupních pinů a není nutné používat různé techniky pro softwarové zvýšení těchto pinů. Zásadní nevýhodou je, že oproti ostatním deskám pracuje na logické úrovni +3.3V což není u těchto desek obvyklé a je potřeba si dávat pozor, na zařízení, které se k této desce připojuje, pokud není určeno pro provoz na +3.3V logice, je potřeba dané zařízení připojit přes takzvaný level shifter. Detailnější porovnání jednotlivých desek z rodiny Arduino lze najít zde [5].



Obrázek 2.18: Vývojová deska Arduino DUE

Jelikož vývojové desky Arduino jsou obecně velmi oblíbené a velmi rozšířené existuje spousta výrobců, kteří vyrábějí nadstavbové desky tzv. shield pro různé vývojové desky Arduino. Jedním takovým shieldem je deska RADDs [8]. Tato

deska lze využít při stavbě hobby 3D tiskárny a umožňuje použití šesti řadičů krokových motorů, což se přesně hodí pro tento model Stewartovy platformy. Díky této desce odpadly problémy se zapojením řadičů na kontaktním nepájivém poli a byla odstraněna možnost nechtěného špatného zapojení. Další výhodou této desky je, že lze na ní připojit koncové tlačítka pro určení koncové polohy ramen modelu. Dále lze k desce připojit LCD display, který lze naprogramovat dle požadavků aplikace. V této úloze je využit pro kontrolu přijímaných zpráv do modelu Stewartovy plošiny.

2.4 Přímovazební řízení polohy koncového efektoru Stewartovy platformy

V této části bude popsáno přímovazební řízení, které slouží k nastavení polohy a orientace koncového efektoru. Jak už bylo napsáno v úvodu této kapitoly došlo ke změně oproti prototypu, kdy přímovazební řízení je realizované přímo v řídicí desce Arduino DUE. Aby bylo možné uživatelsky snadno zadávat požadované polohy koncového efektoru, bylo do modelu implementováno ovládání pomocí tzv. G-kódů. Jedná se o název pro programovací jazyk, který se používá pro řízení NC a CNC obráběcích strojů. Programovací jazyk byl vyvinut počátkem šedesátých let, konečná verze byla schválena v roce 1980 jako RS274D [17]. Pro programování existuje celá řada kódů, které dle normy mají nadefinované významy, jelikož v této práci se využívají G-kódy jen pro pohyb modelu Stewartovy platformy, jsou využity jen některé kódy. Jejich přehled je v následující tabulce 2.2).

Název G-kódu	funkce kódu
G00	lineární pohyb
G01	lineární pohyb
G28	pohyb do domovské pozice
G90	absolutní souřadný systém
G91	Relativní souřadný systém
M17	zapnutí motorů
M18	vypnutí motorů
M100	nápověda
M114	výpis aktuální polohy

Tabulka 2.2: Použití G-kódy v modelu Stewartovy platformy

Nyní budou popsány, jednotlivé příkazy tak, jak musí být zadány, aby jim řídicí deska Arduino DUE rozuměla a poslala správný povel do řadičů krokových motorů.

Pokud uživatel chce pohybovat s koncovým efektem, tak má možnosti pro posuvný pohyb což jsou písmena: X, Y a Z, nebo pro rotaci v X: U, Y: V a Z: W. Příkazy lze zadávat samostatně, pro jednu osu, nebo současně pro více os. Všechny znaky musí být zadané velkými písmeny, aby Arduino DUE správně poznalo požadovaný příkaz. Pro lineární pohyb plošiny lze využít příkazy G00 a G01 následovně:

```
G00 Z2;
```

Tento příkaz pohne s koncovým efektem o 2 cm nahoru. Příkazy pro pohyb lze různě kombinovat například:

```
G00 X2 U10 W5;
```

Tento příkaz pohne s koncovým efektem o 2 cm doleva, otočí s koncovým efektem o deset stupňů kolem osy X a o otočí s koncovým efektem o pět stupňů kolem osy Z.

Další důležitý příkaz je na změnu rychlosti, kterou lze také měnit.

```
G00 F500;
```

Tento příkaz nevykoná žádný pohyb, ale změní nastavení rychlosti pohybu Stewartovy platformy. Rychlost lze měnit v intervalu $< 50; 4000 >$ Rychlost lze měnit buď v klidové poloze a nebo i s vykováním některého pohybu například:

```
G00 Z-2 F800;
```


Dále jsou tu příkazy, které mají informační charakter jedná se příkazy M100 a M114. První zmíněný vypíše krátkou nápovědu, které příkazy lze zadávat a příkaz M114 vypíše aktuální polohu koncového efektoru a zda je používána relativní, nebo absolutní poloha. Pro absolutní polohu je potřeba zadat příkaz:

G90;

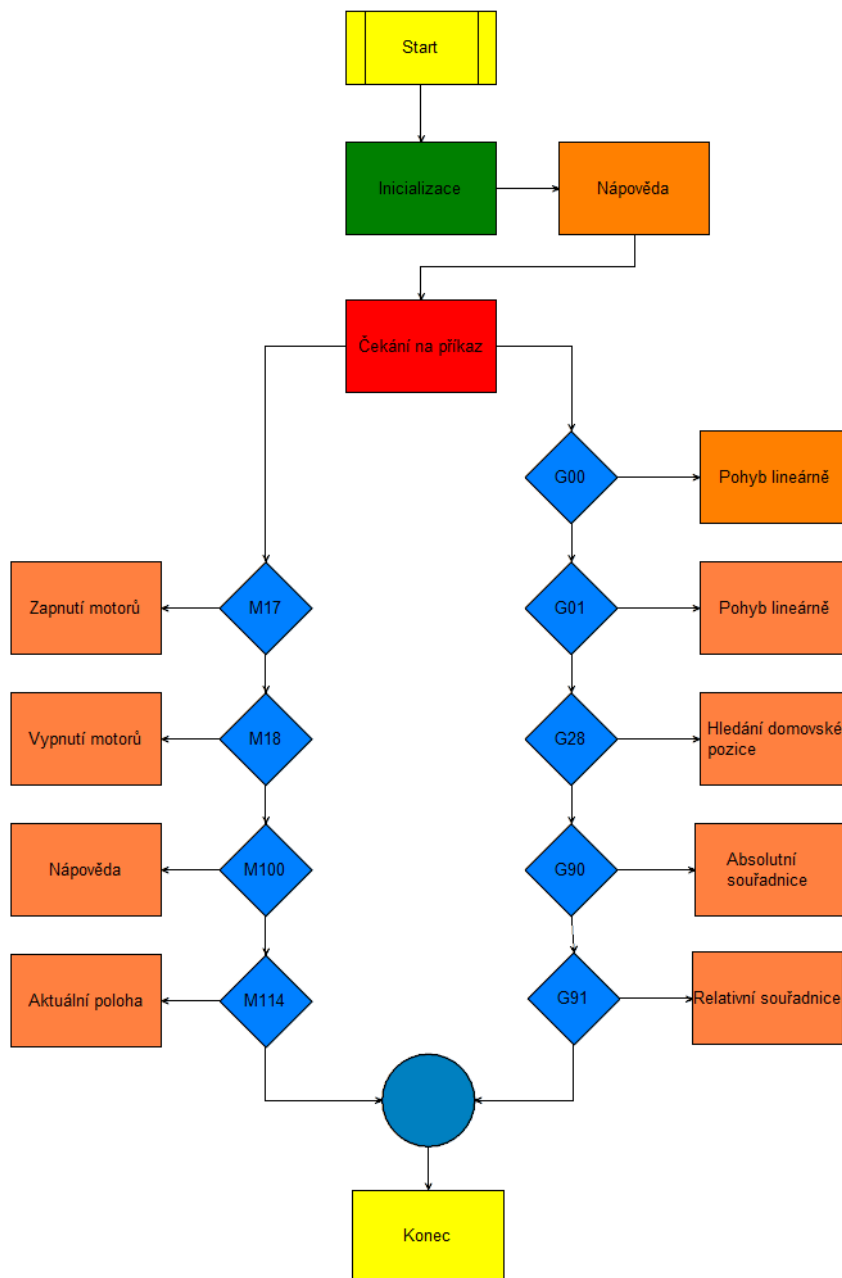
Pro relativní je to příkaz G91;

Další dvojice příkazů slouží pro zapnutí a vypnutí motorů, jedná se o příkazy M17 a M18. Pokud uživatel chce, aby koncový efektor byl v domovské poloze je potřeba zadat příkaz:

G28;

Tento příkaz nejprve pošle všechny ramena Stewartovy platformy do dolní krajní polohy a poté pošle koncový efektor do domovské polohy, kdy jsou nastaveny veškeré hodnoty na hodnotu nula.

Nyní bude popsán řídicí algoritmus, který slouží k polohování koncového efektoru Stewartovy platformy. Poté co se ukázalo, že řízení koncového efektoru kombinací programu Matlab/ Simulink a řídicí desky Stellaris Launchpad nebylo úplně nejlepší volba, jelikož přepis byl velmi složitý. Bylo řízení přepsáno přímo do řídicí desky dle postupu, který použil autor této Stewartovy platformy [14]. Autor tohoto návrhu používal odlišnou řídicí platformu, proto tento kód musel být upraven tak, aby bylo možné ho implementovat do Arduino Due. Zůstala stejná myšlenka běhu programu, která je vidět na obrázku 2.19. Tato idea byla zachována a bude popsána níže. Nedošlo pouze k úpravě parametrů modelu a změně jednotlivých vstupních a výstupních pinů. Řídicí program byl upraven tak, aby fungoval s Arduino Due, proto některé funkce, které nejsou možné realizovat byly odstraněny. Dále bylo upraveno řízení platformy, které autor používá přes virtuální model v počítači, zatímco tato verze je řízená přímo přes G-kódy, tak aby bylo možné model používat pro stabilizační úlohu. Dále bylo nutné upravit rychlost pohybu platformy, která se může měnit, proto bylo nutné do programu připsat časovače, kterým se mění požadovaná rychlost modelu Stewartovy platformy.



Obrázek 2.19: Algoritmus přímovazebního řízení koncového efektoru

Na obrázku 2.19 je vidět algoritmus, kterým se řídí přímovazební řízení koncového efektoru. Skládá se z několika částí, první je Inicializace, ve které se spustí ihned po spuštění Arduino DUE. Nejprve se povolí komunikace po sériové lince, nastaví se všechny vstupní a výstupní piny a nadefinuje se virtuální model Stewartovy platformy. Po tomto kroku se na terminálu vypíše krátká nápověda pro uživatele, aby věděl, jaké příkazy může zadávat. Uživatel zadá příkaz, který se píše velkými písmeny a je zakončen středníkem. Tento příkaz je přijat a pokud splňuje formát zprávy, který je definován pro tuto Stewartovu platformu, tak se příkaz zpracuje. Každý nadefinovaný příkaz má svojí funkci, která se volá, pokud je vyžadována, některé funkce jsou informativní, proto dojde pouze k vypisání informací na terminál uživatele. Funkce, které jsou určeny pro pohyb koncového efektoru se dále zpracovávají. Jak už bylo napsáno, pohyb koncového efektoru je počítán přes inverzní kinematickou úlohu odvozen v 2.2.1. To umožňuje, že poté co uživatel zadá požadovanou polohu, tak se přes vztahy, které jsou uvedeny v matematickém modelu vypočte, kolik kroků a jakým směrem se má na každý motor nastavit. Na algoritmu je vidět, že je používáno lineárního pohybu. Toto vychází opět ze specifikace G-kódu, kde se rozlišují lineární, kruhové a podobně, opět záleží na jednotlivých výrobcích. Zpracování probíhá v několika funkcích, tak aby bylo přehledné. Nejprve se příkaz předá do funkce `hexapod`. V této funkci se dle požadavků spočítají pomocí inverzní kinematické úlohy o jaký úhel se mají krokové motory otočit. Jednotlivé hodnoty kroků jsou uloženy do pole hodnot, které je předáváno do funkce `segment`. V této funkci dochází k vykonávání pohybu. V této funkci je funkce `motory()`, která je volána při přerušování, pokud dojde k požadavku na změnu polohy. Uvnitř funkce se vykoná daný pohyb každého krokového motoru. Kroky se do řadiče přivádí tak, že v cyklu se mění logická hodnota pinu, dle počtu požadovaných kroků. Celý kód je uveden v příloze A.

Kapitola 3

Vybraná stabilizační úloha na modelu Stewartovy platformy

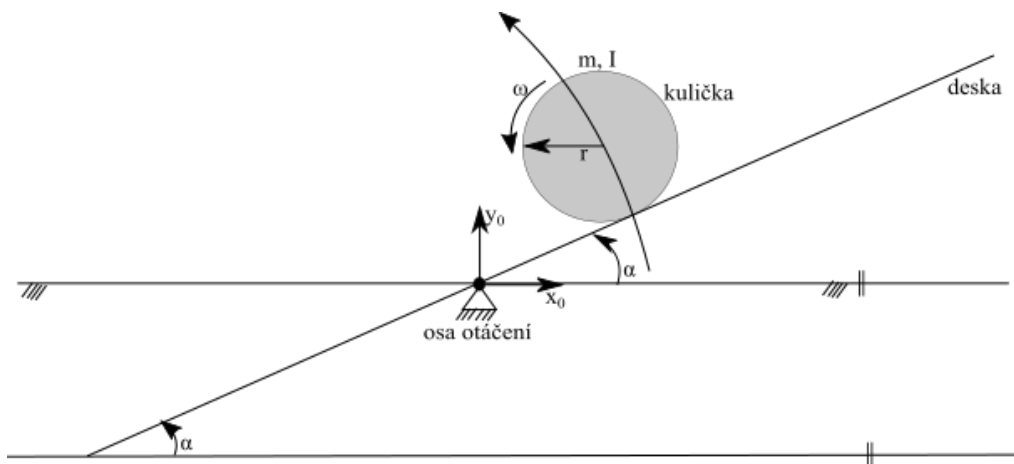
Tato kapitola se bude věnovat popisu vybrané stabilizační úlohy, která bude realizována na modelu Stewartovy platformy. Jako stabilizační úloha byla zvolena stabilizace kuličky na nakloněné rovině. Nejprve bude popsán matematický model nakloněné roviny, který je zjednodušeným modelem plošiny, která je součástí Stewartovy platformy. Následně bude popsán, jaký hardware a software byl použit pro realizaci této úlohy.

3.1 Úloha kulička na nakloněné rovině

Tato část se bude věnovat formulaci úlohy. Budou popsány vztahy, které popisují pohyb kuličky na nakloněné rovině. V závěru této části bude pro takto vzniklý systém představeny přenosové funkce, které budou použity pro návrh regulátoru této úlohy.

3.1.1 Matematický popis úlohy

Tato podkapitola se bude věnovat odvození návrhu zpětnovazebního řízení, které bude ověřeno na Stewartově plošině. Odvození bude provedeno pomocí Euler-Lagrangerovy metody. Odvození bude vycházet z toho, že se jedná o pohyb kuličky po nakloněné rovině viz následující obrázek 3.1. Tento obrázek reprezentuje úlohu v jedné ose. Pro druhou osu by byl obrázek stejný, akorát by se změnil úhel α za úhel β , proto není nutné uvádět druhý obrázek.



Obrázek 3.1: Kulička na desce.

Pro odvození vztahů pro pohyb kuličky na desce zavedeme následující předpoklady:

- Kulička nemá žádná prokluz na desce.
- Kulička je zcela symetrická a homogenní.
- Veškeré tření je zanedbáno.
- Kulička a deska jsou v kontaktu po celou dobu.

Odvození vychází z rovnice pro součet kinetické a potenciální energie.

$$\frac{d}{dt} \frac{\partial L}{\partial \dot{q}_i} - \frac{\partial L}{\partial q_i} = Q_i; 1 \leq i \leq n \quad (3.1)$$

kde $L \triangleq T - V$ je Lagrangian, T je kinetická energie vzhledem k inerciální soustavě, V je potenciální energie vzhledem k inerciální soustavě, n je počet stupňů volnosti, $q_i, i = 1 \dots n$ jsou zobecněné souřadnice a $Q_i, i = 1 \dots n$ jsou zobecněné síly.

Tato úloha má celkem čtyři stupně volnosti, jelikož kulička se pohybuje v souřadnicích x, y a deska se může natáčet v úhlech α, β . Pro zjednodušení výpočtu předpokládáme, že pohyb v souřadnicích x, y je na sobě nezávislý, můžeme úlohu řešit pro jednu osu viz. obrázek 3.1, kdy odvozené vztahy budou stejně platné i pro druhou osu. Začneme s odvozením kinetické energie T kuličky.

$$T = \frac{1}{2} m \dot{x}^2 + \frac{1}{2} I \omega^2 + \frac{1}{2} (\dot{\alpha} x)^2 m \quad (3.2)$$

Kde m je hmotnost kuličky, I je moment setrvačnosti kuličky, ω je úhlová rychlost kuličky a $\dot{\alpha}$ je derivace úhlu, ve kterém je deska natočená. První člen je kinetická energie translace kuličky, druhý člen je kinetická energie rotace kuličky, poslední člen je kinetická energie, kterou kulička vykoná při posunu desky kolem úhlu α .

Úhlovou rychlost ω vyjádříme dle známých veličin jako:

$$\omega = \frac{\dot{x}}{r} \quad (3.3)$$

Dosadíme získaný vztah (3.3) do rovnice (3.2):

$$T = \frac{1}{2}m\dot{x}^2 + \frac{1}{2}\frac{I}{r^2}\dot{x}^2 + \frac{1}{2}(\dot{\alpha}x)^2m \quad (3.4)$$

Nyní vyjádříme potenciální energii kuličky:

$$P = mgx \sin \alpha \quad (3.5)$$

Nyní z rovnic (3.4) a (3.5) určíme Lagrangián L :

$$L = T - P = \frac{1}{2}\left(m + \frac{I}{r^2}\right)\dot{x}^2 + \frac{1}{2}(\dot{\alpha}x)^2m - mgx \sin \alpha \quad (3.6)$$

Následuje výpočet derivací dle rovnice (3.1):

$$\frac{\partial L}{\partial \dot{x}} = \left(m + \frac{I}{r^2}\right)\dot{x} \quad (3.7)$$

$$\frac{\partial L}{\partial x} = -mg \sin \alpha + xm(\dot{\alpha})^2 \quad (3.8)$$

Následně dosadíme získané derivace do rovnice (3.1) a získáme tak rovnici:

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{x}}\right) - \frac{\partial L}{\partial x} = Q_x \Rightarrow \left(m + \frac{I}{r^2}\right)\ddot{x} + mg \sin \alpha - xm(\dot{\alpha})^2 = 0 \quad (3.9)$$

Tato odvozená rovnice (3.9) je nelineární rovnice popisující pohyb kuličky na desce v jedné ose. Stejná nelineární rovnice nám vyjde i pro druhou osu, kde místo x bude y a místo úhlu α bude β .

Nyní provedeme linearizaci odvozené rovnice kolem pracovního bodu $\alpha = 0$,

tímto omezením dojde ke zmenšení plochy, na které si kulička zachovává své lineární vlastnosti, tato linearizace dovoluje měnit úhel o $\pm 2^\circ$, následně dostaneme rovnici:

$$\left(m + \frac{I}{r^2}\right)\ddot{x} = -mg\alpha \quad (3.10)$$

Tuto linearizovanou rovnici (3.10) převedeme pomocí Laplacovy transformace na přenosovou funkci, kdy vstup $u = \alpha$:

$$\left(m + \frac{I}{r^2}\right)Y(s)s^2 = -mgU(s) \quad (3.11)$$

Výsledné přenosy systému jsou:

$$P_x(s) = \frac{U(s)}{Y(s)} = -\frac{mg}{L_1\left(\frac{I}{r^2} + m\right)} \frac{1}{s^2} \quad (3.12)$$

$$P_y(s) = \frac{U(s)}{Y(s)} = -\frac{mg}{L_2\left(\frac{I}{r^2} + m\right)} \frac{1}{s^2} \quad (3.13)$$

kde L_1 je délka desky a L_2 je šířka desky. Tyto odvozené přenosy budou sloužit pro návrh regulátoru ve zvolené stabilizační úloze. Při odvozování byly zavedeny předpoklady, které umožňovaly elegantní odvození rovnic systémů. Dále byl zaveden předpoklad, že jsou pohyby v osách x, y na sobě nezávislé, proto také v návrhu regulátoru nebude uvažován pouze jeden regulátor, ale dva pro každý systém odděleně. Pokud by nebyla provedena linearizace okolo pracovního bodu, muselo by se hledat řešení v oblasti řešení nelineárních rovnic, které je velmi obtížně řešitelné.

3.2 Návrh regulátoru

Při návrhu regulátorů bude vycházeno z přenosů systému, které byly odvozeny v předchozí části. Výsledné přenosy byly získané tak, že za parametry systému byly dosazeny skutečné hodnoty:

Název parametru	Hodnota parametru
hmotnost kuličky	32/1000 [kg]
poloměr kuličky	10.85/1000 [m]
délka nakloněné roviny	304.2/1000 [m]
šířka nakloněné roviny	228.10/1000 [m]

Tabulka 3.1: Hodnoty parametrů modelu

Přenosy systému:

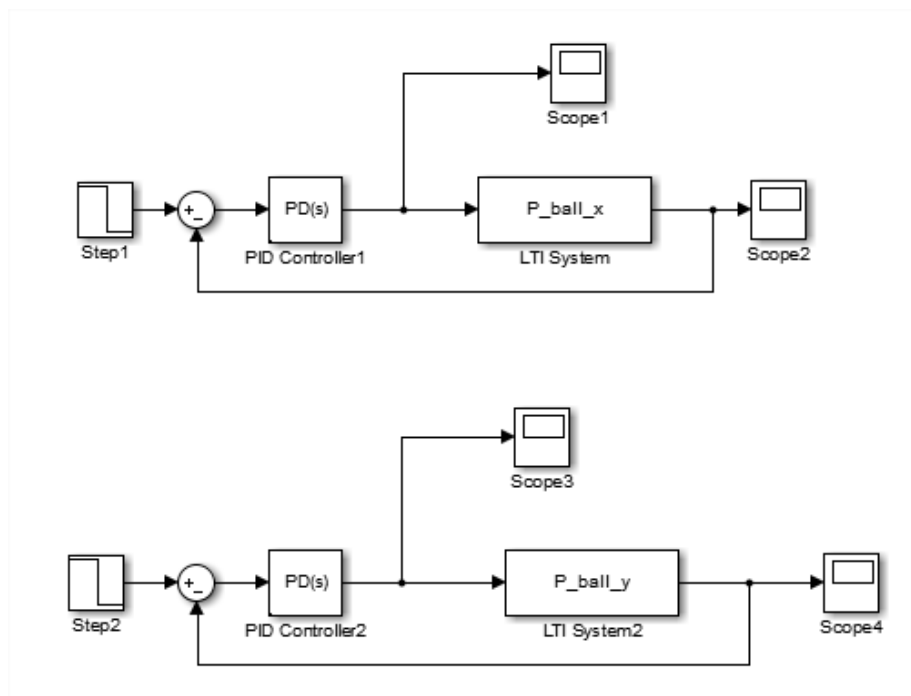
$$P_x(s) = \frac{9.2139}{s^2} \quad (3.14)$$

$$P_y(s) = \frac{12.288}{s^2} \quad (3.15)$$

Z uvedených rovnic je zřejmé, že se jedná o systém druhého řádu, který má dvojnásobnou nulu, která je na mezi stability, proto autor zvolil, pro řízení PD regulátor, který má obecný předpis.

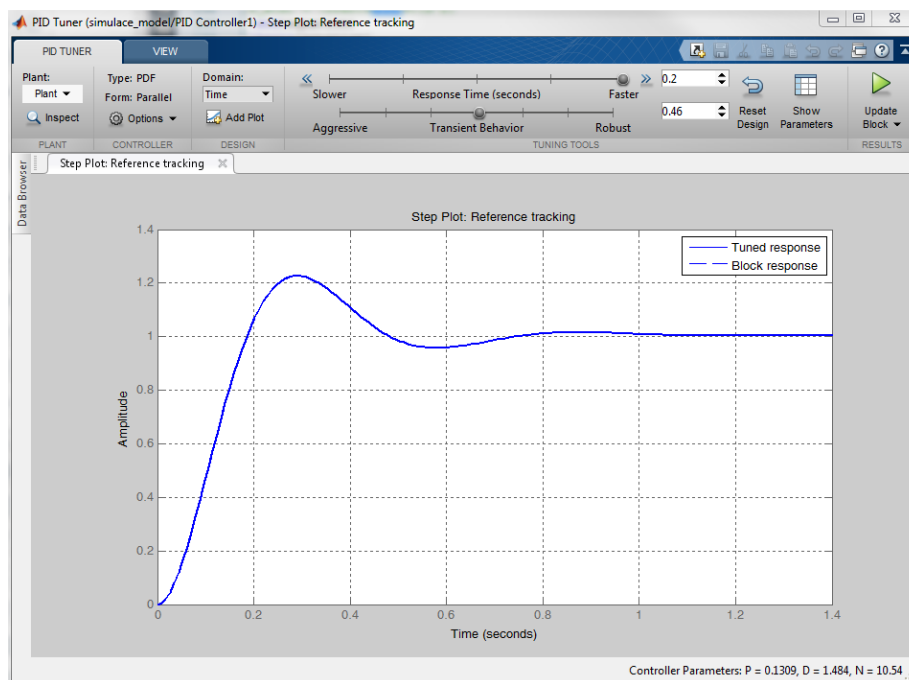
$$R(s) = k_P[1 + T_D s] \quad (3.16)$$

Tento regulátor byl zvolen, jelikož regulátory P, PI by byly nedostatečné, pro řízení této úlohy a nebylo by možné dosáhnout požadovaných vlastností tj. stabilizace kuličky v určité poloze. První návrh byl proveden na požadavek stabilizovat polohu kuličky na souřadnicích $[x, y] = [15, 12]$. Tyto souřadnice určují střed plochy, na které se může kulička pohybovat. Je nutné aby kulička byla umístěna v dostatečně blízkého okolí, ve které ještě platí linearita odvozená v předchozí kapitole. Pro nalezení parametrů regulátoru bylo využito simulačního nástroje Simulink.



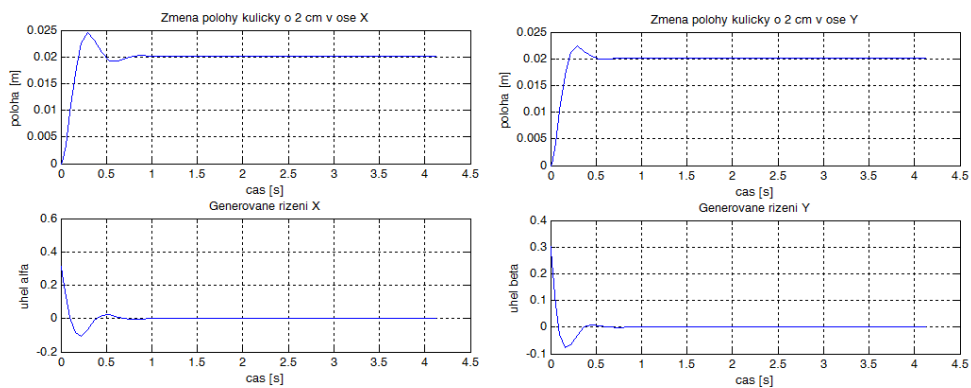
Obrázek 3.2: Regulační smyčka Simulink

Na obrázku 3.2 je vidět regulační obvod pro daný systém. Regulační obvod se skládá z bloků LTI System, který umožňuje v Simulinku pracovat s přenosem typu zero/pole/gain model, ve kterém jsou přenosové funkce systému kuličky na nakloněné rovině. Dále obsahuje blok PID Controller. Tento blok umožňuje vybrat, jaký typ regulátoru uživatel chce používat tj. různé kombinace P, I a D složky regulátoru. Dále obsahuje tlačítko na ladění parametrů. Této možnosti bylo využito při návrhu parametrů PD regulátoru.



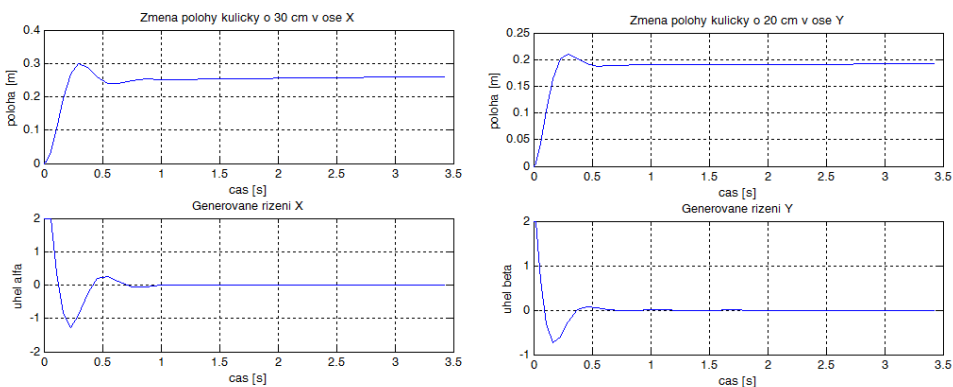
Obrázek 3.3: Návrh PD regulátoru pomocí PID Tuner

Na tomto obrázku 3.2 je vidět, jak probíhá ladění parametrů regulátoru. Na grafu se zobrazuje odezva regulační smyčky na skok. Uživatel si může posuvníky měnit dobu reakce na změnu vstupního signálu a přechodové chování regulátoru, jestli má být hodně agresivní tj. hodně kmitavý, nebo spíše robustní, kdy nedochází k překmitu, ale může se stát, že regulátor ani nebude schopný doregulovat na požadovanou hodnotu. Při tomto návrhu bylo zvoleno, aby regulátor velmi rychle reagoval na změnu vstupního signálu, zároveň, aby překmit byl maximálně 20% tj. někde v polovině možné volby mezi agresivním a robustním regulátorem. Získané parametry pro regulátor X: $k= 0.13$, $t_d= 1.48$, $n_d= 10.54$, regulátor Y: $k= 0.08$, $t_d= 0.96$, $n_d= 15.7$.



Obrázek 3.4: Grafy regulace v simulaci

Na tomto obrázku 3.2 jsou vidět průběhy změny polohy a generované akční zásahy, při změně polohy o 2 cm, z grafů je zřejmé, že regulátory jsou správně navrženy a není problém dosáhnout požadovaných hodnot. Na dalším obrázku 3.2 je vidět, že pokud je požadavek na změnu polohy o 30, respektive 20 cm, což jsou krajní polohy plochy, na které se kulička pohybuje, tak je již mimo linearizační bod a kulička se nikdy nedostane na požadované místo.



Obrázek 3.5: Grafy regulace v simulaci

Kapitola 4

Implementace stabilizační úlohy

V této kapitole bude popsáno, jak probíhala implementace systému kuličky na nakloněné rovině na modelu Stewartovy platformy. Nejprve bude popsán potřebný hardware, o který bylo nutné doplnit model Stewartovy platformy. V další části této kapitoly bude popsána implementace v řídicím systému Rex.

4.1 Hardwarové potřeby k implementaci stabilizační úlohy

Tato část diplomové práce se bude věnovat popisu hardwaru a softwaru, které byly použity pro realizaci vybrané stabilizační úlohy. Zde byla potřeba vyřešit několik problémů. První problém bylo vybrat vhodný hardware pro realizaci řídicího systému, na kterém bude implementován algoritmus regulátoru. Mohlo se zkusit řídicí zákon aplikovat přímo v Arduino Due, nicméně to by znamenalo naprogramovat veškeré sekvence regulátoru přímo v Arduino Due. Tímto by se hodně zkomplikovalo ladění regulátorů. Proto bylo rozhodnuto, že Arduino Due bude obstarávat pouze obsluhu modelu Stewartovy plošiny, jako řídicí hardware bylo vybráno Raspberry Pi, které umožňuje používat řídicí systém REX [1]. Tato volba tedy umožňuje pohodlné navržení regulační smyčky pomocí jednotlivých bloků v REXu a následné nahrání regulátoru přímo do řídicí desky. Další problém, který bylo nutné vyřešit, bylo snímání polohy kuličky na koncovém efektoru. Způsobů, jak měřit polohu kuličky na nakloněné rovině, je několik např. umístění kamer nad model Stewartovy platformy a zpracovávat obrazový materiál a z něho určit polohu na koncovém efektoru. Dále je možné využít kontaktních snímačů polohy, v této práci bylo využito právě tohoto způsobu, kdy se poloha na koncovém efektoru určuje tak, že kulička se pohybuje po dotykovém panelu. Než dojde k popsání jednotlivých hardwarových komponent bude uve-

den popis jednotlivých bloků v řídicím systému Rex, které jsou použity v této práci.

4.1.1 Řídicí hardware

Tato část bude popisovat vybraný řídicí hardware. Řídicí hardware byl vybrán na základě toho, že řídicí algoritmus byl implementován v řídicím systému REX. Tento řídicí systém lze spouštět přímo v počítači, nebo jej nahrát do nějakého zařízení jako např. průmyslové počítače, řídicí desky Raspberry, BeagleBone a jiné. V této práci byl zvolen jako řídicí hardware Raspberry Pi model B+ [2].



Obrázek 4.1: Řídicí hardware Raspberry Pi

Řídicí hardware Raspberry Pi běží na operačním systému Debian Jessie, který je linuxovou distribucí pro takovýto typ miniaturního počítače. V tomto systému byl nainstalován řídicí program REX. Dále je nutné spojit řídicí desku Raspberry s řídicí deskou Arduino Due, která řídí koncový efektor Stewartovy platformy. Toto spojení je zajištěno pomocí USB kabelu a vytvořením virtuální sériové linky mezi oběma zařízeními, což bylo už naznačeno v předchozí kapitole, že Arduino Due toto spojení umožňuje. Jen bylo potřeba upravit jeho firmware, tak aby neposílal žádné zbytečné informace do Raspberry Pi, jelikož není možnost, jak je zobrazit.

4.1.2 Hardware pro měření polohy

Tato část se bude věnovat popisu dotykového panelu, který je využíván pro určení polohy kuličky na koncovém efektoru modelu Stewartovy plošiny. Díky zjištění polohy kuličky, můžeme do systému s regulátorem zavést zpětnou vazbu,

kteřou právě obstarává dotykový panel. Použitý dotykový panel se často využívá ve výpočetní technice, pokud chce uživatel ze svého nedotykového monitoru udělat dotykový. Aby dotykový panel zaznamenával kuličky musí se jednat o resistivní panel. V dnešní době se rozlišují dva typy dotykových panelů resistivní a kapacitní. První zmíněný je možné ovládat čímkoliv, jelikož dotyk se určuje tak, že vrchní pružná část panelu se lehce promáčkne při dotyku a tím se určí místo dotyku. Nevýhodou těchto panelů je, že mají omezenou životnost uvádí se, že vydrží 35- 50 miliónů dotyků. Druhým typem jsou tzv. kapacitní panely, v dnešní době velmi rozšířené v mobilních telefonech. Tyto panely mají větší životnost, jelikož nejsou náchylné k mechanickému poškození z běžného používání, ovšem fungují na principu přirozené vodivosti lidského těla. Jelikož v této práci se pracuje s neživou kuličkou, která nemá povrchovou vodivost jako prsty lidského těla, je nutné použít resistivní dotykový panel. V této práci je využíván resistivní dotykový panel Fujitsu N010-0554-T902 viz obrázek 4.2.



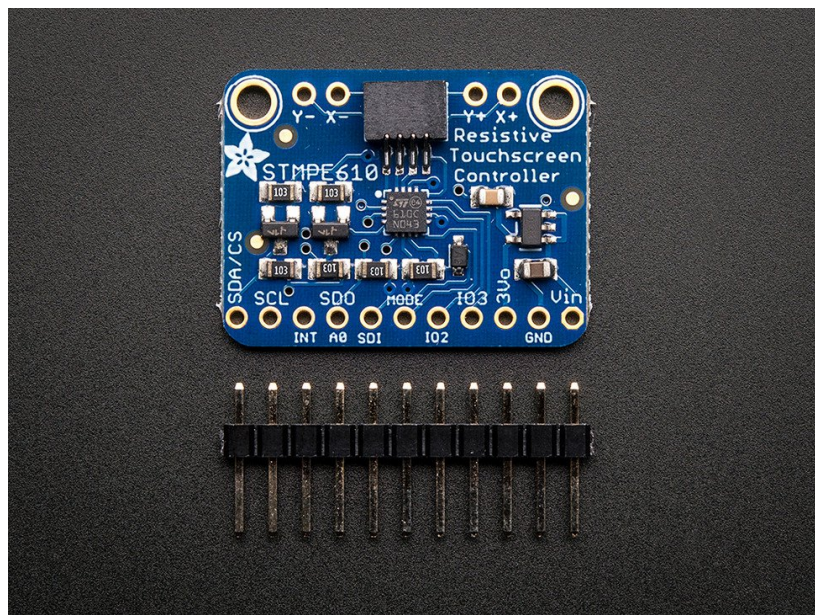
Obrázek 4.2: Používaný dotykový panel

Tento panel sám o sobě nedává informaci o tom, kde došlo k dotyku panelu s

objektem na něm. Abychom tuto informaci získal, je nutné použít nějaký řadič, který zpracovává informace z dotykového panelu. V této práci byl použit řadič STMPE610, který bude popsán v další části.

4.1.3 Řadič dotykového panelu

V této části bude představen mikročip STMPE610, jeho vlastnosti, použití a jak s ním pracovat. Řadič vyrábí firma STMicroelectronics, je určen pro řízení čtyř pinových dotykových panelů, pracuje na logickém napájení 1.8- 3.3 V. Umožňuje komunikovat jak po SPI tak po I2C sběrnici. Tento řadič byl pořízen od firmy Adafruit, která nabízí tento řadič doplněný o potřebný podpůrný obvod ve formě snadno připojitelného modulu viz. obrázek 4.3.

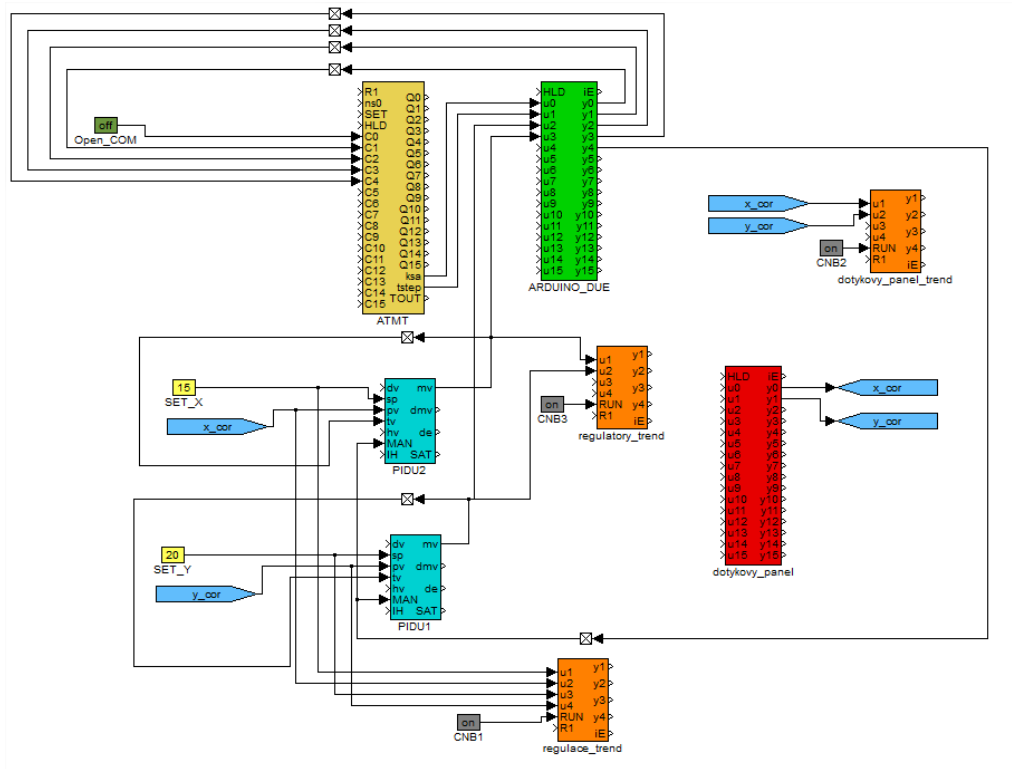


Obrázek 4.3: Používaný řadič dotykového panelu

4.2 Implementace řídicího algoritmu pomocí systému Rex

V této části bude popsána implementace řídicího algoritmu pomocí řídicího systému REX. Na obrázku je vidět celý řídicí algoritmus vytvořený pomocí bloků

v REXu. Řídicí algoritmus vychází ze získaného systému z předchozí kapitoly. Kompletní popis jednotlivých bloků lze dohledat v příručce [12]. Aby bylo možné algoritmus systému realizovat v řídicím systému Rex, bylo nutné správně řídicí algoritmus poskládat ze standardních bloků v programu Rex. V této úloze nešlo sestavit pouze bloky pro regulátory, jako tomu bylo v Simulinku, ale bylo nutné přidat další bloky, které nahrazují přenos idealizovaného systému, za připojení na reálný model Stewartovy platformy. Na obrázku 4.4 je toto schéma vidět. Dále bude uveden význam jednotlivých bloků, které toto schéma zobrazuje.

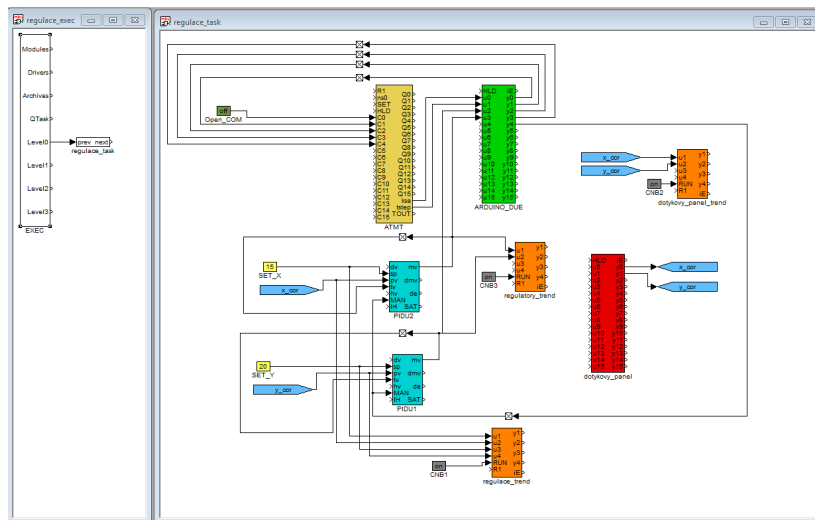


Obrázek 4.4: Regulační schéma v REXu.

Na tomto obrázku je vidět celkové schéma řídicí úlohy, které se skládá z několika částí jenž budou postupně popsány. Jedna část jsou samotné regulátory, které jsou ekvivalentní s regulátory v Simulinku. Dále v horní části schématu jsou vidět dva bloky ATMT a ARDUINO_DUE, tyto bloky jsou tvořeny standardními bloky ATMT a REXLANG a nahrazují ideální přenos systému, komunikací s reálným modelem Stewartovy platformy. Dále je na obrázku červený blok dotykovy_panel, který je opět blok REXLANG a zajišťuje zpětnou vazbu tj. infor-

maci o poloze kuličky na koncovém efektoru Stewartovy platformy. Dále jsou na obrázku vidět oranžové bloky, které slouží k vykreslování průběhu regulace v programu RexView.

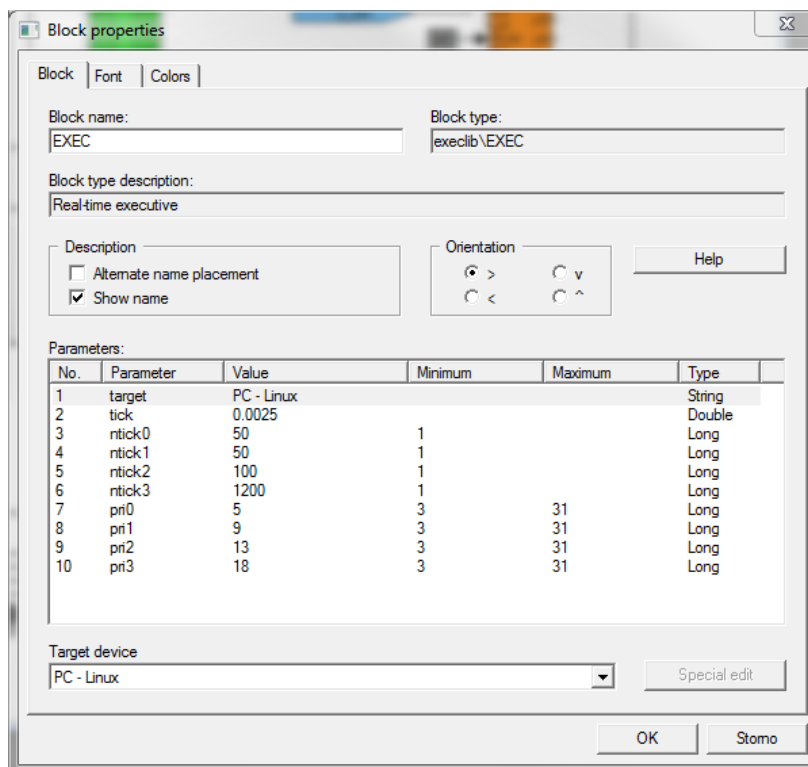
Aby bylo možné úlohu spustit je nutné k tomuto algoritmu ještě přidat řídicí exekutivu. Kompletní schéma v programu RexDraw je na následujícím obrázku.



Obrázek 4.5: Regulační program v RexDraw

Na tomto obrázku 4.5 je vidět, že kompletní regulační smyčka je tvořena dvěma soubory, které se nazývají regulace_exec a regulace_task. V první souboru je uložena informace o úloze tj. v jaké periodě se má úloha vykonávat a jaká úloha (task) se má vykonávat. K tomuto slouží bloky EXEC a TASK, které budou popsány. Druhý soubor tvoří již uvedené schéma regulační smyčky. Tyto části musí být spolu, jelikož tvoří celek, který se následně nahrává do cílového zařízení, na kterém systém Rex běží.

Blok EXEC tvoří základ tzv. hlavního souborového projektu ve formátu .mdl, kterým se konfigurují jednotlivé subsystémy řídicího systému REX. Tento blok má parametry, které se musí nastavit podle cílového zařízení, na kterém běží řídicí program Rex. Dále má blok výstupy, na které se připojují moduly, drivery a jiné dle úlohy. V této úloze je k bloku připojena pouze jedna úloha a blok je podle toho nakonfigurován viz obrázek 4.6.

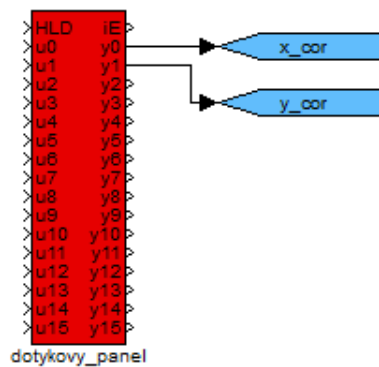


Obrázek 4.6: Konfigurace EXEC

Další blok se nazývá TASK a je připojen na LEVEL0. Tento blok jen odkazuje na model, kde je uložena regulační úloha, podrobnosti lze najít v referenční příručce. Nyní budou popsány jednotlivé bloky z modelu regulace_task.

4.2.1 Implementace snímání polohy kuličky

Firma Adafruit [9] na svých stránkách uvádí odkaz na knihovnu pro řídicí desky Arduino, ve kterých lze tento řadič používat. Nicméně, jelikož tento řadič posílá informace o poloze do řídicí desky Raspberry Pi, bylo nutné napsat vlastní ovládací program, který je kompatibilní s řídicím systémem REX. Tento řídicí systém umožňuje implementovat funkce, které nejsou běžně obsažené v REXu pomocí bloku REXLANG viz obrázek



Obrázek 4.7: Zapojení v programu RexDraw

Aby bylo možné tento řadič v REXu obsluhovat, byla napsaná funkce do bloku REXLANG. Tato funkce vychází z předlohy, kterou firma uvádí ve svém repozitáři na stránkách github [11]. Proto bylo nutné si celou funkci napsat, tak aby bylo možné přijímat informaci o poloze předmětu na dotykovém panelu. Výpis celé funkce lze najít v příloze. Při psaní funkce bylo vycházeno z technické dokumentace k mikročipu a z vytvořené funkční knihovny pro Arduino. Nyní budou uvedeny některé části kódu, které vhodné okomentovat.

Tento řádek, je důležitý, jelikož specifikuje, že v bloku REXLANG v parametru 5, bude uvedena adresa, na které je v tomto případě řadič STMPE připojen.

```
1 #define I2CDEV_FNAME 67
```

V této části kódu jsou zdefinovány jednotlivé adresy registrů mikročipu, které je nutné používat pro správné odečítání hodnot z dotykového panelu. Tyto adresy se nastavují přesně podle technické dokumentace k tomuto mikročipu.

```
1 long STMPE_SYS_CTRL1 [ 2 ];  
#define STMPE_SYS_CTRL1_RESET 0x02 ;  
3 long STMPE_SYS_CTRL2 [ 2 ];
```

```

5 long STMPE_TSC_CTRL[2];
  #define STMPE_TSC_CTRL_EN 0x01;
7 #define STMPE_TSC_CTRL_XYZ 0x00;

9 long STMPE_INT_CTRL[2];
  #define STMPE_INT_CTRL_POL_HIGH 0x04;
11 #define STMPE_INT_CTRL_POL_LOW 0x00;
  #define STMPE_INT_CTRL_EDGE 0x02;
13 #define STMPE_INT_CTRL_LEVEL 0x00;
  #define STMPE_INT_CTRL_ENABLE 0x01;
15 #define STMPE_INT_CTRL_DISABLE 0x00;

17 long STMPE_INT_EN[2];
  #define STMPE_INT_EN_TOUCHDET 0x01;
19 #define STMPE_INT_EN_FIFOTH 0x02;
  #define STMPE_INT_EN_FIFOOF 0x04;
21 #define STMPE_INT_EN_FIFOFULL 0x08;
  #define STMPE_INT_EN_FIFOEMPTY 0x10;
23 #define STMPE_INT_EN_ADC 0x40;
  #define STMPE_INT_EN_GPIO 0x80;
25
  long STMPE_INT_STA[2];
27 #define STMPE_INT_STA_TOUCHDET 0x01;

29 long STMPE_ADC_CTRL1[2];
  #define STMPE_ADC_CTRL1_10BIT 0x00;
31
  long STMPE_ADC_CTRL2[2];
33 #define STMPE_ADC_CTRL2_6_5MHZ 0x02;

35 long STMPE_TSC_CFG[2];
  #define STMPE_TSC_CFG_4SAMPLE 0x80;
37 #define STMPE_TSC_CFG_DELAY_1MS 0x20;
  #define STMPE_TSC_CFG_SETTLE_5MS 0x04;
39

  long STMPE_FIFO_TH[2];
41
  #define STMPE_FIFO_SIZE 0x4C;
43

  long STMPE_FIFO_STA[2];
45 #define STMPE_FIFO_STA_RESET 0x01;
  #define STMPE_FIFO_STA_OFLOW 0x80;
47 #define STMPE_FIFO_STA_FULL 0x40;
  #define STMPE_FIFO_STA_EMPTY 0x20;
49 #define STMPE_FIFO_STA_THTRIG 0x10;

51 long STMPE_TSC_I_DRIVE[2];
  #define STMPE_TSC_I_DRIVE_50MA 0x01;

```

Tato část kódu se vykoná po spuštění pouze jednou a jsou v ní nastavené hodnoty pro jednotlivé registry. Po této části následuje hlavní smyčka, která se volá s periodou dle nastavení REXu.

```
int init(void)
2 {
   i2c_bus_handle = Open(I2CDEV_FNAME); // open I2C bus
4   /* Address: 1 1 0 1 A2 A1 A0 */
   i2c_chip_address = 0x41; // 7-bit address of the I2C device
6
   STMPE_SYS_CTRL2[0] = 0x04;
8   STMPE_SYS_CTRL2[1] = 0x0;
   I2C(i2c_bus_handle, i2c_chip_address, STMPE_SYS_CTRL2, 2,
       i2c_hide, 0); // turn on clocks!
10
   STMPE_TSC_CTRL[0] = 0x40;
12   STMPE_TSC_CTRL[1] = 0x01;
   I2C(i2c_bus_handle, i2c_chip_address, STMPE_TSC_CTRL, 2,
       i2c_hide, 0); // XYZ and enable!
14
   STMPE_INT_EN[0] = 0x0A;
16   STMPE_INT_EN[1] = 0x01;
   I2C(i2c_bus_handle, i2c_chip_address, STMPE_INT_EN, 2, i2c_hide
       , 0);
18
   STMPE_ADC_CTRL1[0] = 0x20;
20   STMPE_ADC_CTRL1[1] = 0x96;
   I2C(i2c_bus_handle, i2c_chip_address, STMPE_ADC_CTRL1, 2,
       i2c_hide, 0); // 96 clocks per conversion
22
   STMPE_ADC_CTRL2[0] = 0x21;
24   STMPE_ADC_CTRL2[1] = 0x02;
   I2C(i2c_bus_handle, i2c_chip_address, STMPE_ADC_CTRL2, 2,
       i2c_hide, 0);
26
   STMPE_TSC_CFG[0] = 0x41;
28   STMPE_TSC_CFG[1] = 0xBF;
   I2C(i2c_bus_handle, i2c_chip_address, STMPE_TSC_CFG, 2, i2c_hide
       , 0);
30
   STMPE_FIFO_TH[0] = 0x4A;
32   STMPE_FIFO_TH[1] = 0x01;
   I2C(i2c_bus_handle, i2c_chip_address, STMPE_FIFO_TH, 2, i2c_hide
       , 0);
34
   STMPE_FIFO_STA[0] = 0x4B;
36   STMPE_FIFO_STA[1] = 0x00;
   I2C(i2c_bus_handle, i2c_chip_address, STMPE_FIFO_STA, 2,
       i2c_hide, 0); // unreset
```

```

38  STMPE_TSC_I_DRIVE[0] = 0x58;
40  STMPE_TSC_I_DRIVE[1] = 0x01;
    I2C(i2c_bus_handle, i2c_chip_address, STMPE_TSC_I_DRIVE, 2,
        i2c_hide, 0);
42
44  STMPE_INT_STA[0] = 0x0B;
    STMPE_INT_STA[1] = 0xFF;
    I2C(i2c_bus_handle, i2c_chip_address, STMPE_INT_STA, 2, i2c_hide
        , 0);
46
48  STMPE_INT_CTRL[0] = 0x09;
    STMPE_INT_CTRL[1] = 0x04;
    I2C(i2c_bus_handle, i2c_chip_address, STMPE_INT_CTRL, 2,
        i2c_hide, 0);
50
52  return 0;
    }

```

V této části se získávají data o poloze kuličky, tak že se čte hodnota z datového registru. Následně jsou tyto získané hodnoty, které udávají souřadnice bodu na dotykovém panelu, převedeny na hodnoty, které odpovídají metrickému systému. Systém je postavený tak, že v pravém horním rohu je počátek s hodnotami [0, 0]. Přepočty jsou udělané tak, aby hodnoty byly určeny na dvě desetinná místa.

```

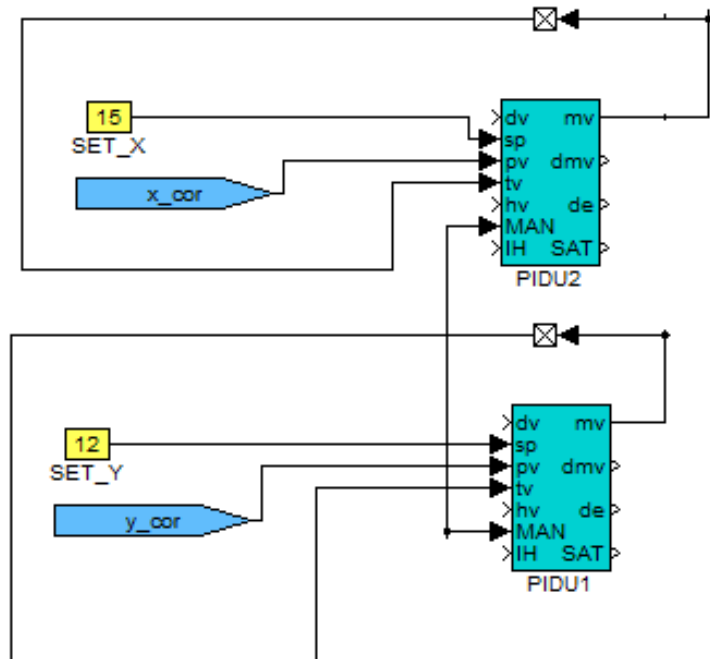
1  long main(void)
    {
3   i2c_bufTx[0] = 0xD7;
    i2c_write_count = 1;
5   i2c_read_count = 4;
    // Sending data via I2C
7   i2c_ret_fun = I2C(i2c_bus_handle, i2c_chip_address, i2c_bufTx
        , i2c_write_count, i2c_bufRx, i2c_read_count);
    x1 = (i2c_bufRx[0] << 4) | (i2c_bufRx[1] >> 4);
9   y1 = ((i2c_bufRx[1] & 0xf) << 8) | i2c_bufRx[2];

11  x1 = (x1-58)/125;
    y1 = (y1-98)/160;
13
15  x_value = x1;
    y_value = y1;
    return 0;
17  }

```

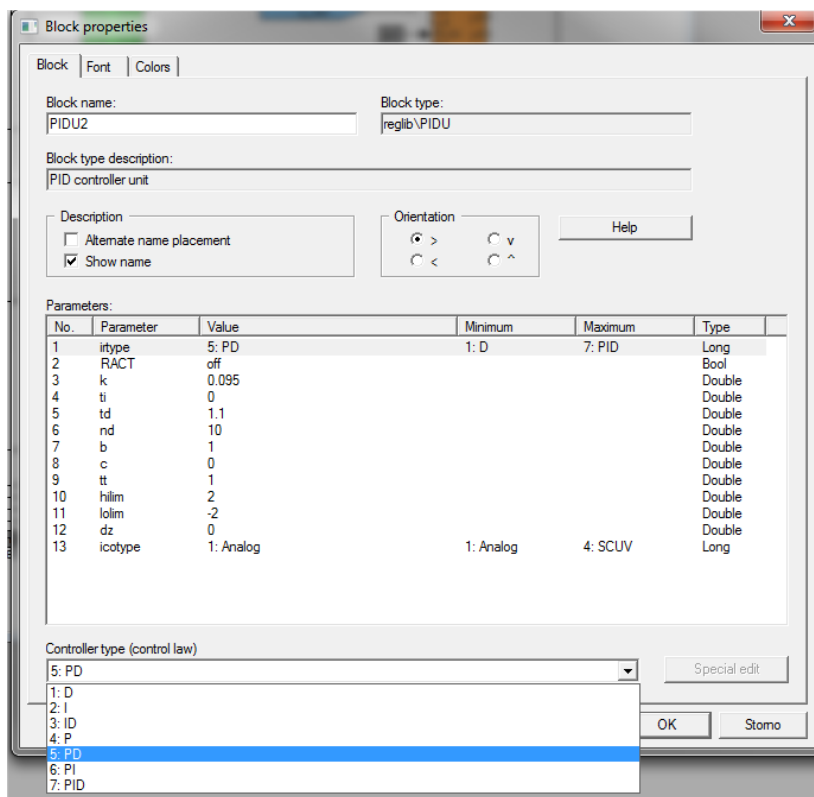
4.2.2 Implementace PD regulátorů

Tato část se bude věnovat implementaci regulátorů systému v řídicím systému Rex. Regulátory jsou implementovány pomocí bloků PIDU1 a PIDU2, tyto bloky jsou podobné bloků, které lze najít v programu Simulink. Zapojení regulátorů je vidět na následujícím obrázku.



Obrázek 4.8: Zapojení PIDU regulátorů

Na tomto obrázku je vidět zapojení PIDU regulátorů, které je použito v této implementaci. Výstup těchto bloků je označován jako *mv* a je to akční veličina, která se posílá do modelu Stewartovy platformy, dále je tato veličina přivedena na vstup regulátoru na parametr *tt*. Dalším vstupním parametrem je *sp*, což je požadovaná veličina. Parametr *pv* je řízená veličina, tedy informace z dotykového panelu o poloze kuličky. Poslední parametr, který se používá je *MAN* tento parametr říká, jak se má regulátor chovat. Pokud je v manuálním režimu, tak se nepočítá akční zásah, ale je na výstup přivedena definovaná hodnota, naopak pokud je tento parametr vypnutý tak dochází k výpočtu akčního zásahu. Do těchto bloků se nastavují parametry regulátoru. Parametry, které je potřeba zadat jsou ukázány v následujícím obrázku.

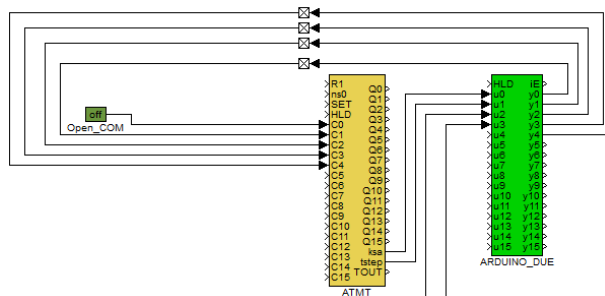


Obrázek 4.9: Konfigurace PIDU

Nejprve se volí typ regulátoru z nabízeného seznamu, který je v dolní části okna. Následně se volí, zda akční zásah je takový, jaký vygeneruje regulátor, nebo zda má obrácené znaménko parametr RACT. Dále jsou nastavovány parametry P,I a D složky, pokud není zvolen celý PID regulátor, tak není nutné nastavovat všechny parametry. Parametry b a c jsou váhové faktory pro proporcionální a derivační složku. Další parametr tt je časová konstanta výsledování, pokud ale není obsažena integrační složka, tak nemá žádný význam. Další dva parametry omezují maximální a minimální mez akčního zásahu. Poslední parametr je dz, což je pásmo necitlivosti regulátoru.

4.2.3 Komunikace s řídicí jednotkou modelu

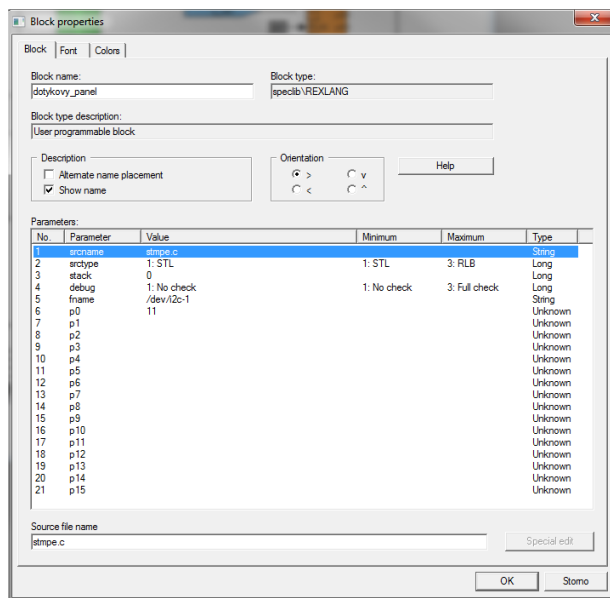
Pro spojení bylo potřeba použít dalšího bloku REXLANG, který navazuje spojení s Arduino Due a posílá mu příkazy, které se mají vykonat s koncovým efektořem Stewartovy platformy. Zdrojový kód je uveden v příloze B, zde bude uvedeny jen některé části.



Obrázek 4.10: Konfigurace PIDU

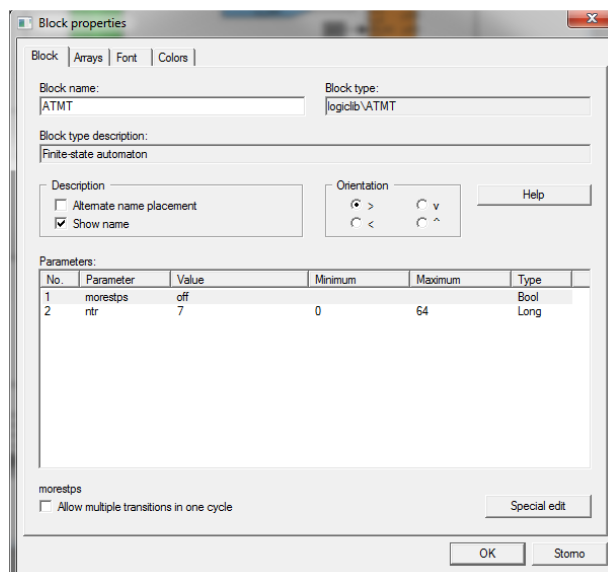
Na tomto obrázku je vidět, jak je realizována komunikace mezi Raspberry Pi a Arduino Due. Tato komunikace se skládá ze dvou bloků: ATMT a REXLANG. Nyní budou tyto dva bloky popsány.

Blok REXLANG slouží k implementaci funkcí, které nejsou implementovány v řídicím systému Rex a nelze je efektivně vytvořit z dostupných bloků. Tento blok se programuje pomocí skriptovacího jazyka, který vychází z jazyka C. Vlastnosti bloku jsou na následujícím obrázku.

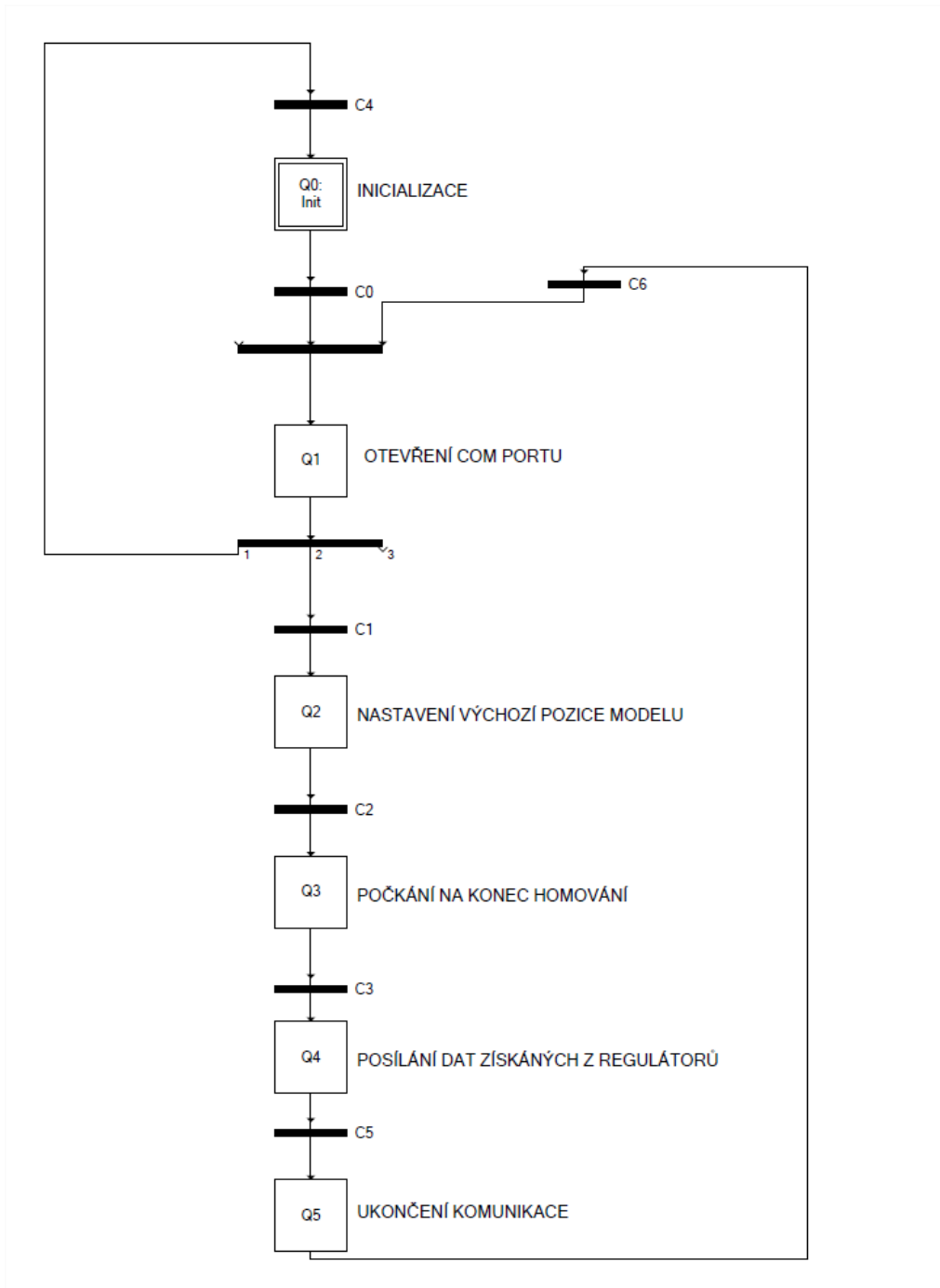


Obrázek 4.11: Konfigurace REXLANG

Na obrázku 4.11 je vidět, jaké parametry se nastavují, nastavují se zde název souboru, kde je skript k bloku REXLANG, dále typ skriptu, další důležitým polem je parametr fname, který specifikuje jméno datového souboru pro čtení a ukládání hodnot. V tomto regulačním schématu je tento parametr využíván pro zadání adresy, na které je k řídicí platformě připojen řadič dotykového panelu a Arduino Due. Tento parametr se dá zjistit v operačním systému Raspberry Pi. Dalším blokem je ATMT. Tento blok slouží pro konfiguraci sekvenčního řízení. Aby bylo možné realizovat úlohu, tak je nutné nejprve provést inicializaci, navázání komunikace, uvedení modelu Stewartovy platformy do domovské pozice a pozice pro regulační úlohu a až poté spustit samotnou regulační úlohu, hodí se tento blok zařadit do regulačního schématu a automatizovat tak jednotlivé kroky regulace. Konfigurace sekvenčního řízení lze zakreslit v programu SFC editor.



Obrázek 4.12: Konfigurace ATMT



Obrázek 4.13: Schéma sekvenčního automatu v SFC editoru

V této části kódu probíhá transformace hodnot akčních veličin regulátoru na pole znaků, toto je nutné, aby bylo možné poslat hodnoty do Arduino Due pomocí sériové linky.

```
1 void DoubleToBuffer(double val , long values [])
2 {
3     long data;
4     val=val*100;
5     data = (long) val;
6     if(data >0)
7     {
8         values [0]='+';
9         values [1]= (data/1000)+'0';
10        data=data%1000;
11        values [2]= (data/100)+'0';
12        data=data%100;
13        values [3]='.';
14        values [4]=(data/10)+'0';
15        data=data%10;
16        values [5]= data+'0';
17    }
18    else
19    { data=data*-1;
20      values [0]='-';
21      values [1]= (data/1000)+'0';
22      data=data%1000;
23      values [2]= (data/100)+'0';
24      data=data%100;
25      values [3]='.';
26      values [4]=(data/10)+'0';
27      data=data%10;
28      values [5]= data+'0';
29    }
30 }
31 }
```

V další části bude uveden hlavní část tohoto skriptu, která obstarává komunikaci mezi Raspberry Pi a Arduino Due. Tato část je rozdělena do několika kroků pomocí přepínače. Tyto kroky jsou určovány stavem sekvenčního automatu.

```
long main(void)
2 {
3     switch(stav_automatu)
4     {
5         case 0: break;
```

```

6
8     case 1:
10         if (hCom<0)
12             {
14                 hCom = Open(COM_DEVICE_FNAME,COM_BAUDRATE,
16                 COM_PARITY_NONE); //opening serial device
18             }
20         if(hCom>0)
22             {
24                 c1=1;
26             }
28         break;
30     case 2:
32         buffer [0] = 'G';
34         buffer [1] = '2';
36         buffer [2] = '8';
38         buffer [3] = ';';
40         dataCnt = Send(hCom,buffer ,4); //send data , number
42         of bytes = 4
44         buffer [0] = 'G';
46         buffer [1] = '0';
48         buffer [2] = '0';
50         buffer [3] = ' ';
52         buffer [4] = 'Z';
54         buffer [5] = '2';
56         buffer [6] = '.';
58         buffer [7] = '5';
60         buffer [8] = ';';
62         dataCnt = Send(hCom,buffer ,9);
64         c2=1;
66         break;
68     case 3:
70         if (time_atmt >1.5)
72             {
74                 c3=1;
76             }
78         break;
80     case 4:
82         povoleni_regulace = 0;
84         DoubleToBuffer(pozadovane_u , convData1);
86         DoubleToBuffer(pozadovane_v , convData2);
88         buffer [0] = 'G';
90         buffer [1] = '0';
92         buffer [2] = '0';
94         buffer [3] = ' ';
96         buffer [4] = 'U';
98         buffer [5] = convData1 [0];
100        buffer [6] = convData1 [1];
102        buffer [7] = convData1 [2];

```

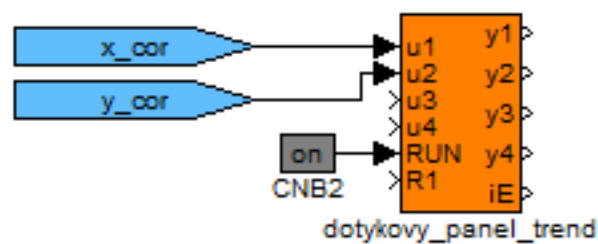
```

54     buffer[8] = convData1[3];
55     buffer[9] = convData1[4];
56     buffer[10] = convData1[5];
57     buffer[11] = ' ';
58     buffer[12] = 'V';
59     buffer[13] = convData2[0];
60     buffer[14] = convData2[1];
61     buffer[15] = convData2[2];
62     buffer[16] = convData2[3];
63     buffer[17] = convData2[4];
64     buffer[18] = convData2[5];
65     buffer[19] = ';';
66
67     //now all the data can be sent
68     dataCnt = Send(hCom, buffer, 20); //send data,
69     number of bytes = 20
70     break;
71     default: break;
72 } //end switch
return 0;
}

```

4.2.4 Vizualizace dat

Posledními použitými bloky jsou bloky TREND. Tyto bloky slouží k tomu, aby bylo možné zobrazit v grafu hodnoty z bloků, které jsou používány v regulaci. Tyto bloky umožňují záznam trendů v reálném čase až čtyř vstupních signálů. Pro zobrazení výsledků se používá program RexView.



Obrázek 4.14: Konfigurace REXLANG

Kapitola 5

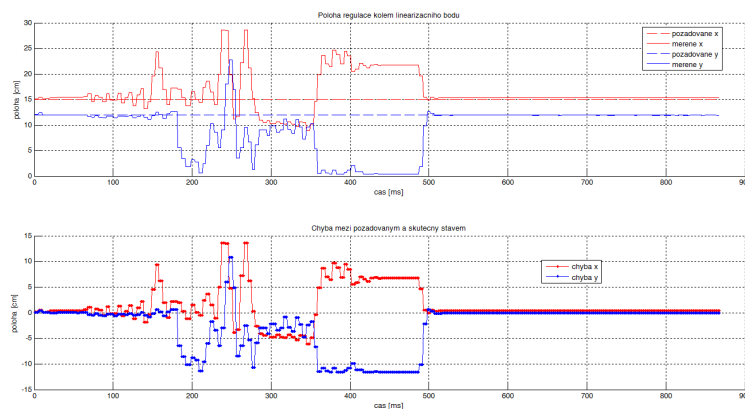
Ověření implementace

Tato kapitola bude věnována ověření implementace algoritmu stabilizační úlohy na modelu Stewartovy platformy. Jak bylo uvedeno v předchozích kapitolách 3. a 4., byla vybrána úloha stabilizace kuličky na nakloněné rovině. Tato úloha byla vybrána pro ověření algoritmu řízení orientace koncového efektoru. Nejprve bylo nutné tuto úlohu matematicky popsat, což bylo provedeno v kapitole tři. Během matematického popisu se uvažoval pouze model kuličky a koncový efektor byl brán, jako nakloněná rovina, nebylo vůbec uvažováno chování modelu Stewartovy platformy. Dále se uvažovalo, že koncový efektor se může otáčet pouze v ose X a Y tj. bylo zanedbáno, že model má šest stupňů volnosti a využívají se pouze dva stupně volnosti. V matematickém popisu systému též není uvažováno tření kuličky s podložkou a předpokládá se, že kulička má stále kontakt s podložkou, ačkoliv byly tyto předpoklady zavedeny, výsledné vztahy vedly na nelineární rovnice. Tyto rovnice byly linearizovány kolem pracovního bodu pro úhly $[0,0]$. Pro takto linearizovaný model byly navrženy regulátory. Tyto regulátory byly implementovány do řídicího systému Rex, což je popsáno v kapitole 4. Během ověřování implementace se ukázalo, že reálný systém se chová odlišně oproti lineárnímu modelu. Proto bylo nutné udělat korekce na nastavených parametrech regulátoru. Tyto korekce probíhaly během testování algoritmu metodou pokus omyl, kdy byly parametry upravovány tak, aby se kulička stabilizovala v určité poloze. Během implementace se ukázalo, že velmi záleží na volbě kuličky, byly zkoušeny dvě, gumová z počítačové myši a ocelová kulička, každá kulička měla jiné vlastnosti, které se projeví tak, že bylo nutné jinak nastavit regulátory. Dále se ukázalo, že regulátory fungují, jen pokud je kulička v dostatečně blízkém okolí požadovaného bodu, pokud je kulička dále od požadované hodnoty, tak již nepletí linearizační model a kulička již nejde stabilizovat. Možné řešení toho problému by bylo, kdyby se navrhlo dostatečné množství regulátorů pro různé pracovní body, poté by se v regulátoru přepínaly parametry podle toho v jakém pracovním bodě by byla kulička. Další možností

by bylo, kdyby se na úlohu dívalo, jako na MIMO systém (Multi Input, Multi Output - mnoho vstupů, mnoho výstupů) v tomto případě by se jednalo o systém 2×2 . Problém v tomto přístupu spočívá v tom, že ačkoliv by bylo možné určit přenosy z X na X a Y na Y, které by vycházely z matematického modelu, tak by bylo komplikované hledat přenosy z X na Y a Y na X. Dalším možným řešením této úlohy je použití neuronové sítě na nastavení parametrů, nebo návrh prediktivního řízení. Další možností, jak zlepšit chování stabilizační úlohy, by bylo navrhovat regulátory na nelineární model. Hledání regulátorů v nelineární oblasti je velmi složité, jelikož je potřeba najít oblast, ve které lze takové řešení najít, tyto metody nejsou tak jasné a na pro každý systém jsou tak specifické, že hledání řešení v nelineární oblasti by se nemuselo vůbec povést.

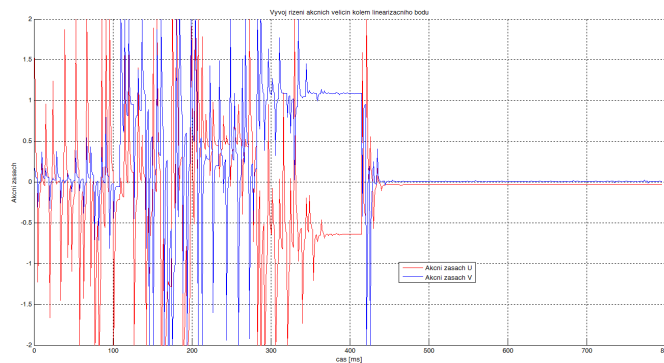
5.1 Realizace úlohy

Jak už bylo napsáno výše, ukázalo se, že bylo nutné na reálném modelu upravit parametry regulátorů tyto parametry jsou: regulátor X: $k=0.095$, $t_d=1.1$, $n_d=10$, regulátor Y: $k=0.094$, $t_d=0.75$, $n_d=10$. Je vidět, že parametry jsou si velmi podobné, jelikož se během testování ukázalo, že rozdíl v šířce a délce není tak velký, aby parametry musely být velmi odlišné na následujícím grafu bude ukázáno výstup z TREND bloků, které byly exportovány z RexView do Matlabu a tam vykresleny.



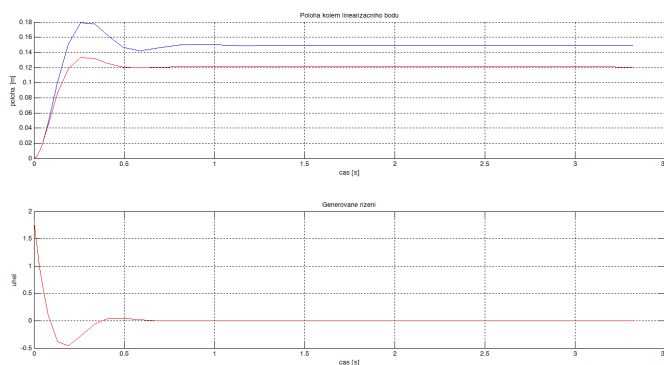
Obrázek 5.1: Vývoj polohy kolem linearizačního bodu na modelu Stewartovy platformy

Obrázek 5.1 ukazuje chování pokud byl nastaven, jako požadovaný bod střed koncového efektoru a kulička byla umístěna blízko tohoto místa, je vidět, že nějakou dobu trvalo, než se kulička dostala do požadované polohy a poté už byla v této poloze stabilizována. Na dalším obrázku bude vidět řízení, které bylo generováno regulátory.

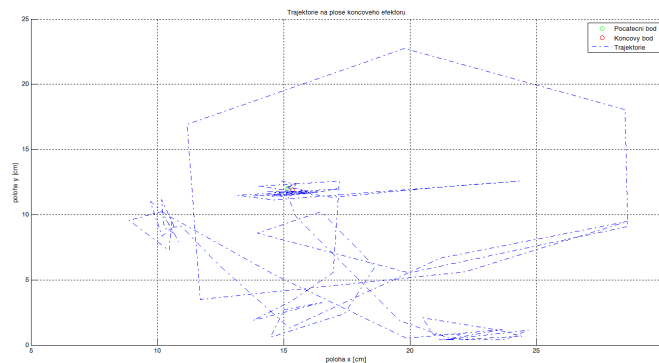


Obrázek 5.2: Vývoj řízení kolem linearizačního bodu na modelu Stewartovy platformy

Obrázek 5.2 ukazuje, jaké bylo generované řízení potřebné pro stabilizaci kuličky v požadované poloze, Toto řízení je z počátku velmi kmitavé, což zapříčiňuje volba PD regulátoru a také to, že reálná kulička se chová jinak, než ta která je modelována v Matlab/ Simulink.



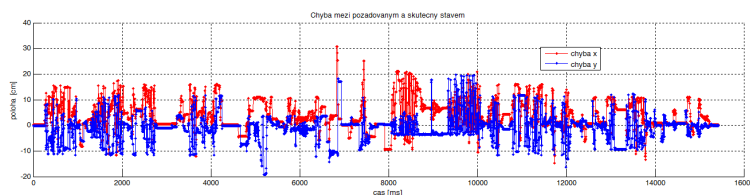
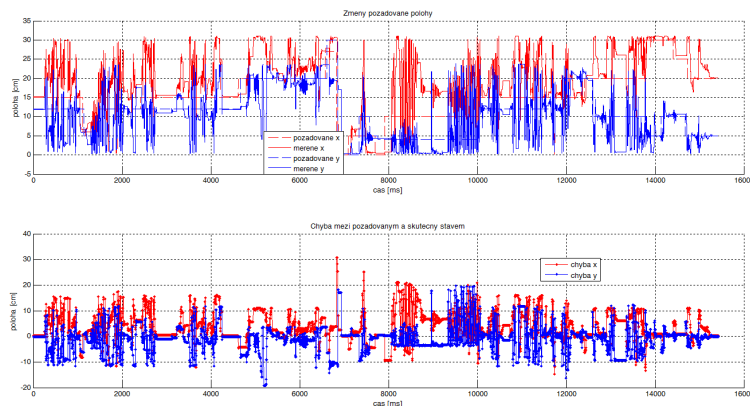
Obrázek 5.3: Vývoj řízení kolem linearizačního bodu v Matlab/Simulink



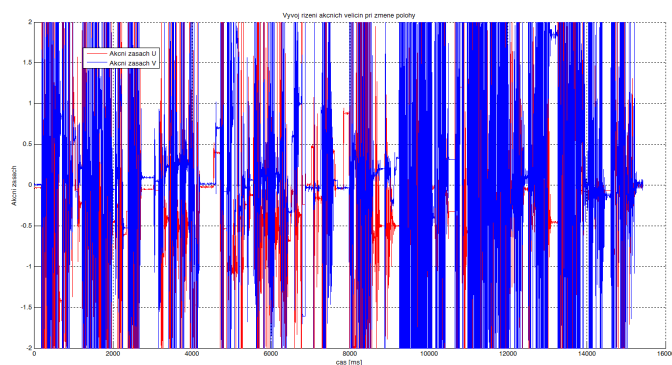
Obrázek 5.4: Trajektorie pohybu kuličky na koncovém efektoru

Na obrázku 5.4 je vidět zobrazení pohybu kuličky na koncovém efektoru modelu Stewartovy platformy kolem linearizačního bodu.

Následující obrázky budou zobrazovat průběh polohy a generovaného řízení, pokud byla prováděna velká změna polohy.



Obrázek 5.5: Vývoj polohy při změnách požadovaných bodů na modelu Stewartovy platformy



Obrázek 5.6: Vývoj řízení na modelu Stewartovy platformy

Na obrázcích 5.5 a 5.6 je vidět chování modelu, pokud byly prováděny skokové změny polohy a více než 5 cm. Na obrázcích je vidět, že tyto požadavky bylo velmi složité splnit, během testování se ukázalo, že pokud byly požadavky na změnu polohy o více centimetrů, tak regulátory vygenerovaly velký akční zásah, což znamenalo, že kulička na nakloněné rovině se rychle rozjela a přejela požadovaný bod a regulátor okamžitě vygeneroval velký akční zásah s opačným znaménkem a trvalo poměrně dlouho, než se kulička dostala do požadované polohy. Toto chování může být způsobeno nevhodným nastavením některé složky regulátoru, nebo porušením linearizovaného modelu, které se projevuje tímto chováním.

Kapitola 6

Závěr

V této části budou shrnuty dosažené výsledky této diplomové práce. Cílem diplomové práce bylo navrhnout řídicí elektroniku pro model Stewartovy platformy, realizovat systém pro určení orientace koncového efektoru a navrhnout a ověřit algoritmus řízení orientace koncového efektoru.

Nejprve byl použit prototyp modelu Stewartovy platformy, který byl přepracován, tak aby vyhovoval požadavkům, která byla očekávána od tohoto modelu. Následně byla navržena řídicí elektronika, která umožňuje realizovat řízení koncového efektoru. Celá navržená elektronika byla připojena k modelu Stewartovy platformy a byl navržen algoritmus řízení orientace koncového efektoru. Toto řízení je přímovazební a zakládá se na inverzní kinematice, díky které jsou matematicky popsány vztahy, které umožňují dle požadované polohy spočítat, jak se mají otočit krokové motory.

Dále byla na modelu Stewartovy platformy realizována stabilizační úloha kuličky na nakloněné rovině. Při této realizaci se plně nevyužilo všech vlastností Stewartovy platformy, ale bylo využito pouze rotací v osách x a y . Matematický model byl zjednodušen a linearizován tak, aby bylo snadnější nalézt přenosové funkce, které popisují daný systém. Během realizace úlohy se ukázalo, že některé omezení úlohy nejsou vhodně zvolená, proto realizace není taková, jaká byla očekávána v simulacích. Aby tyto problémy mohly být odstraněny, bylo by potřeba přepracovat matematický model, do kterého by se musela zahrnout i dynamika modelu Stewartovy platformy. Dále by bylo potřeba určit vazby mezi jednotlivými osami a dívat se na systém, ne jako lineární s jedním vstupem a jedním výstupem, ale jako na nelineární systém, který má dva vstupy a dva výstupy.

Příloha A

Zde budou uvedeny jednotlivé zdrojové kódy, které byly použité v této práci. Tyto zdrojové kódy jsou určeny pro Arduino Due a je to implementace přímova-
zebního řízení koncového efektoru

Main task

V této části je uveden zdrojový kód, který je hlavní část programu, která je puš-
těná během celé doby běhu programu.

```
1 //-----  
2 // INCLUDES  
3 //-----  
4 #include "configuration.h"  
5 #include "vector3.h"  
6 #include "segment.h"  
7 #include "hexapod.h"  
8 #include <LiquidCrystal.h>  
9 #include <DueTimer.h>  
10  
11  
12  
13 //-----  
14 // GLOBALS  
15 //-----  
16 // speeds  
17 float feed_rate=0; // human version  
18 float feed_rate_home=100;  
19  
20 // settings  
21 char mode_abs=0; // absolute mode?  
22  
23 #ifdef VERBOSE  
24 char *letter="UVWXYZ";  
25 #endif  
26  
27 DueTimer timer = DueTimer(0); // vybrany casovac
```

```

29 //-----
// METHODS
31 //-----
/**
33 * Delay for the appropriate time. delayMicroseconds() doesn't
    work for ms values > ~16k.
    * @input us how many microseconds to wait
35 */
void pause(long us) {
37     delay(us/1000);
    delayMicroseconds(us%1000);
39 }

41
/**
43 * Set the feedrate (speed motors will move)
    * @input nfr the new speed in steps/second
45 */
float feedrate(float nfr) {
47     if(feed_rate==nfr) return nfr; // same as last time? quit
        now.

49     if(nfr>MAX_FEEDRATE) {
        Serial.print(F("Feedrate set to maximum ("));
51         Serial.print(MAX_FEEDRATE);
        Serial.println(F(" steps/s"));
53         nfr=MAX_FEEDRATE;
    }

55     if(nfr<MIN_FEEDRATE) { // don't allow crazy feed rates
        Serial.print(F("Feedrate set to minimum ("));
57         Serial.print(MIN_FEEDRATE);
        Serial.println(F(" steps/s"));
59         nfr=MIN_FEEDRATE;
    }

61     feed_rate=nfr;

63     return feed_rate;
}

65
67 /**
    * Set the motor position in number of steps
69 */
void motor_position(int n0,int n1,int n2,int n3,int n4,int n5)
    {
71     // here is a good place to add sanity tests
    h.arms[0].last_step=n0;
73     h.arms[1].last_step=n1;

```

```

75     h.arms[2].last_step=n2;
76     h.arms[3].last_step=n3;
77     h.arms[4].last_step=n4;
78     h.arms[5].last_step=n5;
79 }
80
81 /**
82  * Grips the power on the motors
83  */
84 void motor_enable() {
85     int i;
86     for(i=0;i<NUM_AXIES;++i) {
87         digitalWrite(h.arms[i].motor_enable_pin,LOW);
88     }
89 }
90
91 /**
92  * Releases the power on the motors
93  */
94 void motor_disable() {
95     int i;
96     for(i=0;i<NUM_AXIES;++i) {
97         digitalWrite(h.arms[i].motor_enable_pin,HIGH);
98     }
99 }
100
101
102 /**
103  * print the current position , feedrate , and absolute mode.
104  */
105 void hexapod_where() {
106     output("X",h.ee.pos.x);
107     output("Y",h.ee.pos.y);
108     output("Z",h.ee.pos.z);
109     output("U",h.ee.r);
110     output("V",h.ee.p);
111     output("W",h.ee.y);
112     output("F",feed_rate);
113     Serial.println(mode_abs?"ABS":"REL");
114 }
115
116
117 /**
118  * print the current motor positions in steps
119  */
120 void motor_where() {
121     output("0",h.arms[0].last_step);

```

```

123   output("1",h.arms[1].last_step);
      output("2",h.arms[2].last_step);
125   output("3",h.arms[3].last_step);
      output("4",h.arms[4].last_step);
127   output("5",h.arms[5].last_step);
      }
129
131  /**
      * display helpful information
133  */
      void help() {
135     Serial.print(F("StewartPlatform"));
          Serial.println(VERSION);
137     Serial.println(F("Commands:"));
          Serial.println(F("M17/M18; – enable/disable motors"));
139     Serial.println(F("M100; – this help message"));
          Serial.println(F("M114; – report position and feedrate"));
141     Serial.println(F("F, G00, G01, G04, G28, G90, G91 as
          described by http://en.wikipedia.org/wiki/G-code"));
      }
143
144  /**
145   * First thing this machine does on startup. Runs only once.
      */
147  // initialize the library with the numbers of the interface
          pins RS EN DB4 DB5 DB6 DB7
      LiquidCrystal lcd(42, 43, 44, 45, 46, 47);
149
      void setup() {
151     Serial.begin(BAUD); // open coms
          lcd.begin(20, 4);
153     lcd.setCursor(0, 0);
          lcd.print("Stewart platform");
155     lcd.setCursor(0,1);
          lcd.print("Receive command:");
157     help(); // say hello
159
          // pinMode(13,OUTPUT);
161
          hexapod_setup();
          motor_enable();
163     feedrate(800); // set default speed
          hexapod_position(0,0,0,0,0,0);
165     motor_position(0,0,0,0,0,0);
          // hexapod_find_home();
167     timer.attachInterrupt(motory).start();
          parser_ready();
169 }

```

```

171 void loop() {
173     parser_listen();
}

```

configuration

V této části je ukázán konfigurační část programu, kde se nastavují parametry modelu Stewartovy platformy.

```

2 #ifndef CONFIGURATION_H
3 #define CONFIGURATION_H
4
5 //-----
6 // CONSTANTS
7 //-----
8 // #define VERBOSE           (1) // add to get a lot more
9 //     serial output.
10 // #define DEBUG_SWITCHES   (1)
11
12 #define VERSION             (1) // firmware version
13
14 #define BAUD                (115200) // How fast is the
15 //     Arduino talking?
16 #define MAX_BUF            (64) // What is the longest
17 //     message we can store?
18
19 #define STEPS_PER_TURN     (200) // depends on your stepper
20 //     motor. most are 200.
21 #define MICROSTEPS         (16.0) // depends on set_up
22 //     microstep
23
24 #define MAX_FEEDRATE       (4000.0) // depends on timer
25 //     interrupt & hardware
26 #define MIN_FEEDRATE      (50.0)
27
28 // measurements based on computer model of robot
29 #define BICEP_LENGTH       ( 5.00)
30 #define FOREARM_LENGTH    (15.000)
31 #define SWITCH_ANGLE      (19.690)
32 // top center to wrist hole: X7.635 Y+/-0.553 Z0.87
33 #define T2W_X              ( 7.635)
34 #define T2W_Y              ( 0.553)
35 #define T2W_Z              (-0.870)
36 // base center to shoulder hole: X8.093 Y+/-2.15 Z7.831
37 #define B2S_X              ( 8.093)

```



```

32 #define B2S_Y          ( 2.150)
   #define B2S_Z          ( 6.618)
34
   #define STEPS_PER_CM    (25)
36 #define STEPS_PER_DEG  (8.888)
38
   // ** Nothing below this line needs to be configured **
40
42 // convenience macros
   #define NUM_AXIES       (6)
44
   #define TWOPI           (PI*2.0)
46 #define DEG2RAD         (PI/180.0)
   #define RAD2DEG        (180.0/PI)
48
   #define MICROSTEPS_PER_TURN (STEPS_PER_TURN*MICROSTEPS)
50 #define CIRCUMFERENCE    (BICEP_LENGTH*TWOPI)
   #define MICROSTEP_DISTANCE (CIRCUMFERENCE/MICROSTEPS_PER_TURN
   ) // distance elbow moves in a single microstep
52 #define MICROSTEP_PER_DEGREE (MICROSTEPS_PER_TURN/360.0)
54
   #endif

```

hexapod.h

V této části, je ukázána část programu, kde se vytváří struktura virtuálního modelu Stewartovy platformy pro výpočty inverzní kinematické úlohy.

```

2 #ifndef HEXAPOD_H
   #define HEXAPOD_H
4
   // _____
6 // INCLUDES
   // _____
8 #include "configuration.h"
10
   // _____
12 // STRUCTS
   // _____
14 struct EndEffector {
   Vector3 up;
16 Vector3 left;

```

```

18   Vector3 forward;
19   Vector3 pos;
20   Vector3 relative;
21   float r,p,y; // roll, pitch, yaw
22 };
23
24 struct Arm {
25     Vector3 shoulder;
26     Vector3 elbow;
27     Vector3 shoulder_to_elbow;
28     Vector3 wrist;
29
30     float angle;
31     int last_step;
32     int new_step;
33
34     int motor_step_pin;
35     int motor_dir_pin;
36     int motor_enable_pin;
37     int motor_scale; // 1 or -1
38
39     char limit_switch_pin;
40     int limit_switch_state;
41 };
42
43
44 struct Hexapod {
45     Vector3 base;
46     Arm arms[NUM_AXIES];
47     EndEffector ee;
48
49     Vector3 v; // max XYZ vel
50     Vector3 destination; // target XYZ position
51 };
52
53
54 //-----
55 // GLOBALS
56 //-----
57 extern Hexapod h;
58
59
60 //-----
61 // METHODS
62 //-----
63 void hexapod_setup();
64 void hexapod_update_ik(Vector3 &mov, Vector3 &rpy);
65 void hexapod_line(float newx, float newy, float newz, float newu,

```

```
        float newv, float neww, float new_feed_rate);
66
68 #endif
```

hexapod

V této části kódu se definují piny, na kterých jsou připojeny výstupy na řadiče krokových motorů, vstupní piny koncových efektorů. Tato část slouží pro výpočty kroků dle požadavků na polohu koncového efektoru.

```
1
3 //-----
4 // INCLUDES
5 //-----
6 #include "configuration.h"
7 #include "hexapod.h"
8 #include "segment.h"
9
11 //-----
12 // GLOBALS
13 //-----
14 Hexapod h;
15
17 //-----
18 // METHODS
19 //-----
20 /**
21  * set up the digital pins and call hexapod_build_model()
22  */
23 void hexapod_setup() {
24     int i;
25
26     // set up the pins
27     h.arms[0].motor_step_pin=24;
28     h.arms[0].motor_dir_pin=23;
29     h.arms[0].motor_enable_pin=26;
30     h.arms[0].limit_switch_pin=28;
31
32     h.arms[1].motor_step_pin=17;
33     h.arms[1].motor_dir_pin=16;
34     h.arms[1].motor_enable_pin=22;
35     h.arms[1].limit_switch_pin=30;
```

```

37  h.arms[2].motor_step_pin=2;
    h.arms[2].motor_dir_pin=3;
39  h.arms[2].motor_enable_pin=15;
    h.arms[2].limit_switch_pin=32;
41
42  h.arms[3].motor_step_pin=61;
43  h.arms[3].motor_dir_pin=60;
    h.arms[3].motor_enable_pin=62;
45  h.arms[3].limit_switch_pin=34;
46
47  h.arms[4].motor_step_pin=64;
    h.arms[4].motor_dir_pin=63;
49  h.arms[4].motor_enable_pin=65;
    h.arms[4].limit_switch_pin=36;
51
52  h.arms[5].motor_step_pin=51;
53  h.arms[5].motor_dir_pin=53;
    h.arms[5].motor_enable_pin=49;
55  h.arms[5].limit_switch_pin=38;
56
57  for(i=0;i<NUM_AXIES;++i) {
    // set the motor pin & scale
59  h.arms[i].motor_scale=((i%2)? -1:1);
    pinMode(h.arms[i].motor_step_pin,OUTPUT);
61  pinMode(h.arms[i].motor_dir_pin,OUTPUT);
    pinMode(h.arms[i].motor_enable_pin,OUTPUT);
63  // set the switch pin
    h.arms[i].limit_switch_state=HIGH;
65  pinMode(h.arms[i].limit_switch_pin,INPUT_PULLUP);
    digitalWrite(h.arms[i].limit_switch_pin,HIGH);
67  }
    int relay = 58;
69  pinMode(relay, OUTPUT);
    digitalWrite(relay, HIGH);
71  hexapod_build_model();
    }
73
74  /**
75  * Build a virtual model of the hexapod at home position.
76  */
77  void hexapod_build_model() {
79  h.ee.relative.Set(0,0,0);
    Vector3 zero(0,0,0);
81  hexapod_update_endeffector(zero,zero);
    hexapod_build_shoulders();
83  hexapod_update_wrists();

```

```

85 // find the starting height of the end effector at home
    position
86 // @TODO: project wrist-on-bicep to get more accurate
    distance
87 Vector3 el=h.arms[0].elbow;
    Vector3 wr=h.arms[0].wrist;
89 float aa=(el.y-wr.y);
    float cc=FOREARM_LENGTH;
91 float bb=sqrt((cc*cc)-(aa*aa));
    aa=el.x-wr.x;
93 cc=bb;
    bb=sqrt((cc*cc)-(aa*aa));
95 h.ee.relative.z=bb+B2S_Z-T2W_Z;

97 hexapod_update_ik(h.ee.relative,zero);
    }
99

101 /**
    * Build a virtual model of the hexapod shoulders for
    calculating angles later.
103 */
void hexapod_build_shoulders() {
105     Vector3 n,o,n1,o1;
    float c,s;
107     int i;
    for(i=0;i<3;++i) {
109         Arm &arma=h.arms[i*2+0];
        Arm &armb=h.arms[i*2+1];
111
        c=cos(i*TWOPI/3.0f);
113         s=sin(i*TWOPI/3.0f);

115         n=h.ee.forward;
        o=h.ee.up ^ h.ee.forward;
117         o.Normalize();

119         n1 = n* c + o*s;
        o1 = n*-s + o*c;
121
        arma.shoulder = n1*B2S_X - o1*B2S_Y + h.ee.up*B2S_Z;
123         armb.shoulder = n1*B2S_X + o1*B2S_Y + h.ee.up*B2S_Z;
        arma.elbow = n1*B2S_X - o1*(B2S_Y+BICEP_LENGTH) + h.ee.up*
        B2S_Z;
125         armb.elbow = n1*B2S_X + o1*(B2S_Y+BICEP_LENGTH) + h.ee.up*
        B2S_Z;
        arma.shoulder_to_elbow=-o1;
127         armb.shoulder_to_elbow=o1;
    }
}

```

```

129 #ifdef VERBOSE
131     for(i=0;i<6;++i) {
133         Arm &arm=h.arms[i];
135         Serial.print(i);
137         Serial.print("\ts =");
139         Serial.print(arm.shoulder.x);
141         Serial.print(F(", "));
143         Serial.print(arm.shoulder.y);
145         Serial.print(F(", "));
147         Serial.print(arm.shoulder.z);
149
151         Serial.print("\te =");
153         Serial.print(arm.elbow.x);
155         Serial.print(F(", "));
157         Serial.print(arm.elbow.y);
159         Serial.print(F(", "));
161         Serial.println(arm.elbow.z);
163     }
165 #endif
167
169 /**
171  * Update the end effector, the wrist positions, the elbows,
173  * and then calculate the new shoulder angles
175  * @input mov final end effector position
177  * @input rpy final end effector roll pitch yaw (relative to
179  * base)
181  */
183 void hexapod_update_ik(Vector3 &mov, Vector3 &rpy) {
185     hexapod_update_endeffector(mov, rpy);
187     hexapod_update_wrists();
189     hexapod_update_shoulder_angles();
191 }
193
195 /**
197  * Update the end effector according to the desired motion
199  * @input mov final end effector position
201  * @input rpy final end effector roll pitch yaw (relative to
203  * base)
205  */
207 void hexapod_update_endeffector(Vector3 &mov, Vector3 &rpy) {
209     // translation
211     h.ee.pos = mov;
213
215     // roll pitch & yaw
217     h.ee.r=rpy.x*DEG2RAD;

```

```

175 h.ee.p=rpy.y*DEG2RAD;
176 h.ee.y=rpy.z*DEG2RAD;
177 h.ee.up.Set(0,0,1);
178 h.ee.forward.Set(1,0,0);
179 h.ee.left.Set(0,1,0);

181 // roll
182 Vector3 axis;
183 axis.Set(1,0,0);
184 h.ee.up.Rotate(axis,h.ee.r);
185 h.ee.forward.Rotate(axis,h.ee.r);
186 h.ee.left.Rotate(axis,h.ee.r);
187
188 // pitch
189 axis.Set(0,1,0);
190 h.ee.up.Rotate(axis,h.ee.p);
191 h.ee.forward.Rotate(axis,h.ee.p);
192 h.ee.left.Rotate(axis,h.ee.p);
193
194 // yaw
195 axis.Set(0,0,1);
196 h.ee.up.Rotate(axis,h.ee.y);
197 h.ee.forward.Rotate(axis,h.ee.y);
198 h.ee.left.Rotate(axis,h.ee.y);
199 }

201
202 /**
203  * Update the wrist positions according to the end effector
204  */
205 void hexapod_update_wrists() {
206     Vector3 n,o,n1,o1;
207     float c,s;
208     int i;
209     for(i=0;i<3;++i) {
210         Arm &arma=h.arms[i*2+0];
211         Arm &armb=h.arms[i*2+1];

212         c=cos(i*TWOPI/3.0f);
213         s=sin(i*TWOPI/3.0f);

214         n=h.ee.forward;
215         o=h.ee.up ^ h.ee.forward;
216         o.Normalize();

217         n1 = n* c + o*s;
218         o1 = n*-s + o*c;

219         arma.wrist = h.ee.pos + n1*T2W_X - o1*T2W_Y + h.ee.up*T2W_Z

```

```

;
    armb.wrist = h.ee.pos + n1*T2W_X + o1*T2W_Y + h.ee.up*T2W_Z
;
225 }

227 #ifdef VERBOSE
    for(i=0;i<6;++i) {
229     Arm &arm=h.arms[i];
        Serial.print(i);
231     Serial.print("\twrist =");
        Serial.print(arm.wrist.x);
233     Serial.print(F(", "));
        Serial.print(arm.wrist.y);
235     Serial.print(F(", "));
        Serial.println(arm.wrist.z);
237     }
    #endif
239 }

241
/**
243  * update the elbow positions according to the wrists and
        shoulders , then find shoulder angle.
    */
245 void hexapod_update_shoulder_angles() {
    Vector3 ortho,w,wop,temp,r;
247     float a,b,d,r1,r0,hh,y,x;

249     int i;
        for(i=0;i<6;++i) {
251         Arm &arm=h.arms[i];

253         #ifdef VERBOSE
            Serial.print(i);
255         #endif

257         // project wrist position onto plane of bicep (wop)
            ortho.x=cos((i/2)*TWOPI/3.0f);
259         ortho.y=sin((i/2)*TWOPI/3.0f);
            ortho.z=0;

261
            w = arm.wrist - arm.shoulder;
263
            a=w | ortho; //endeffector' distance
265         wop = w - (ortho * a);
            //arm.wop.pos=wop + arm.shoulder; // e' location
267         // vector_add(arm.wop,wop,arm.shoulder);

269         // because wop is projected onto the bicep plane , wop-elbow

```



```

    is not the same as wrist-elbow.
    // we need to find wop-elbow to calculate the angle at the
    shoulder.
271    b=sqrt(FOREARM_LENGTH*FOREARM_LENGTH-a*a); // e'j distance

273    // use intersection of circles to find elbow point.
    //a = (r0r0 - r1r1 + d*d) / (2 d)
275    r1=b; // circle 1 centers on e'
    r0=BICEP_LENGTH; // circle 0 centers on shoulder
277    d=wop.Length();
    // distance from shoulder to the midpoint between the two
    possible intersections
279    a = ( r0 * r0 - r1 * r1 + d*d ) / ( 2*d );

281 #ifdef VERBOSE
    Serial.print("\tb =");
283    Serial.println(b);
    Serial.print("\td =");
285    Serial.println(d);
    Serial.print("\ta =");
287    Serial.println(a);
#endif

289    // normalize wop
291    wop /= d;
    // find the midpoint
293    temp=arm.shoulder+(wop*a);
    // with a and r0 we can find h, the distance from midpoint
    to intersections.
295    hh=sqrt(r0*r0-a*a);
    // get a normal to the line wop in the plane orthogonal to
    ortho
297    r = ortho ^ wop;
    if(i%2==0) arm.elbow=temp + r * hh;
299    else      arm.elbow=temp - r * hh;

301    // use atan2 to find theta
    temp=arm.elbow-arm.shoulder;
303    y=-temp.z;
    temp.z=0;
305    x=temp.Length();
    if( ( arm.shoulder_to_elbow | temp ) < 0 ) x=-x;
307    arm.angle= atan2(-y,x) * RAD2DEG;
#endif
309    Serial.print(i);
    Serial.print("\tangle =");
311    Serial.println(arm.angle);
#endif
313 }

```

```

315 }
317 /**
319  * Uses bresenham's line algorithm to move both motors
321  * @input newx the destination x position
322  * @input newy the destination y position
323  */
324 void hexapod_line(float newx, float newy, float newz, float newu,
325                  float newv, float neww, float new_feed_rate) {
326     Vector3 endpos(newx, newy, newz);
327     Vector3 endrpy(newu, newv, neww);
328
329     endpos+=h.ee.relative;
330
331     Vector3 startpos=h.ee.pos+h.ee.relative;
332     Vector3 startpy(h.ee.r, h.ee.p, h.ee.y);
333     Vector3 dpos=endpos-startpos;
334     Vector3 drpy=endrpy-startrpy;
335     Vector3 ipos, irpy;
336
337     //@TODO: find which of the wrist positions moves the furthest
338     , base everything on that.
339     // ** BEGIN TOTAL JUNK
340     long steps_pos=ceil(dpos.Length()/STEPS_PER_CM);
341     long steps_rpy=ceil(drpy.Length()/STEPS_PER_DEG);
342     long steps = ( steps_pos > steps_rpy ? steps_pos : steps_rpy)
343     ;
344     if(steps > MAX_SEGMENTS) steps=MAX_SEGMENTS;
345     // ** END TOTAL JUNK
346
347     if( steps >=MAX_SEGMENTS-1) steps=MAX_SEGMENTS-2;
348
349 #ifdef VERBOSE
350     Serial.print(steps);
351     Serial.println(F(" steps."));
352 #endif
353
354     if(steps==0) return;
355
356     float istep = 1.0/(float)steps;
357
358     long i;
359     int j;
360
361     for(i=0;i<=steps;++i) {
362         ipos=startpos+dpos*(i*istep);
363         irpy=startrpy+drpy*(i*istep);

```

```

hexapod_update_ik(ipos , irpy);
361
#ifdef VERBOSE
363   Serial.print(i);
   Serial.print(" = ");   Serial.print(h.arms[0].angle);
365   Serial.print(F(", "));   Serial.print(h.arms[1].angle);
   Serial.print(F(", "));   Serial.print(h.arms[2].angle);
367   Serial.print(F(", "));   Serial.print(h.arms[3].angle);
   Serial.print(F(", "));   Serial.print(h.arms[4].angle);
369   Serial.print(F(", "));   Serial.println(h.arms[5].angle);
#endif
371
   // convert angle to motor steps
373   for(j=0;j<6;++j) {
   h.arms[j].new_step = h.arms[j].angle *
MICROSTEP_PER_DEGREE;
375   }

377 #ifdef VERBOSE
   Serial.print(i);
379   Serial.print(" = ");   Serial.print(h.arms[0].new_step);
   Serial.print(F(", "));   Serial.print(h.arms[1].new_step);
381   Serial.print(F(", "));   Serial.print(h.arms[2].new_step);
   Serial.print(F(", "));   Serial.print(h.arms[3].new_step);
383   Serial.print(F(", "));   Serial.print(h.arms[4].new_step);
   Serial.print(F(", "));   Serial.println(h.arms[5].new_step);
385 #endif

   motor_prepare_segment(h.arms[0].new_step ,
                           h.arms[1].new_step ,
389                           h.arms[2].new_step ,
                           h.arms[3].new_step ,
391                           h.arms[4].new_step ,
                           h.arms[5].new_step , new_feed_rate);
393   }

395   // @TODO: This does not take into account movements of a
   fraction of a step. They will be misreported and lead to
error.
hexapod_position(newx , newy , newz , newu , newv , neww);
397   }

399
void hexapod_position(float npx , float npy , float npz , float npu ,
float npv , float npw) {
401   h.ee.pos.x=npx;
   h.ee.pos.y=npv;
403   h.ee.pos.z=npz;
   h.ee.r=npu;

```

```

405 h.ee.p=npv;
406 h.ee.y=npw;
407 // @TODO: Update the motor positions to match the new virtual
         position or they will go crazy on the next hexapod_line()
         // @TODO: Until motor positions can match hexapod position
         G92 will have unintended effects.
409 }

411
412 /**
413  * read the limit switch states
414  * @return 1 if a switch is being hit
415  */
416 char hexapod_read_switches() {
417     char i, hit=0;
418     int state;
419
420     for(i=0;i<6;++i) {
421         state=digitalRead(h.arms[i].limit_switch_pin);
422 #ifdef DEBUG_SWITCHES
423         Serial.print(state);
424         Serial.print('\t');
425 #endif
426         if(h.arms[i].limit_switch_state != state) {
427             h.arms[i].limit_switch_state = state;
428 #ifdef DEBUG_SWITCHES
429             Serial.print(F("Switch "));
430             Serial.println(i,DEC);
431 #endif
432         }
433         if(state == LOW) ++hit;
434     }
435 #ifdef DEBUG_SWITCHES
436     Serial.print('\n');
437 #endif
438     return hit;
439 }

441
442 /**
443  * Move one motor in a given direction
444  * @input the motor number [0..6]
445  * @input the direction to move 1 for forward, -1 for backward
446  */
447 void hexapod_onestep(int motor, int dir) {
448 #ifdef VERBOSE
449     Serial.print(letter[motor]);
450 #endif
451     Arm &a = h.arms[motor];

```

```

dir *= a.motor_scale;
453 digitalWrite(a.motor_dir_pin, dir>0?LOW:HIGH);
digitalWrite(a.motor_step_pin, HIGH);
455 digitalWrite(a.motor_step_pin, LOW);
}
457
459 /**
* Move the motors until they connect with the limit switches,
then return to "zero" position.
461 */
void hexapod_find_home() {
463 // Serial.println(F("Finding min..."));

motor_enable();

467 char i;
// until all switches are hit
469 while(hexapod_read_switches()<6) {
#ifdef VERBOSE
471 Serial.println(hexapod_read_switches(), DEC);
#endif
473 // for each stepper,
for(i=0; i<6; ++i) {
475 // if this switch hasn't been hit yet
if(h.arms[i].limit_switch_state == HIGH) {
477 // move "down"
hexapod_onestep(i, -1);
479 }
}
481 pause(100000L/feed_rate_home);
}
483

485 // The arms are 19.69 degrees from straight down when they
hit the switch.
// @TODO: This could be better customized in firmware.
487 float horizontal = 90.00 - SWITCH_ANGLE;
long steps_to_zero = MICROSTEP_PER_DEGREE * horizontal;
489 // Serial.println(F("Homing..."));
#ifdef VERBOSE
491 Serial.print("steps=");
Serial.println(step_size);
493 #endif

495 for(; steps_to_zero>0; --steps_to_zero) {
for(i=0; i<6; ++i) {
497 hexapod_onestep(i, 1);
}
}

```

```

499     pause(100000L/feed_rate_home);
    }
501     hexapod_position(0,0,0,0,0,0);
    motor_position(0,0,0,0,0,0);
503     Serial.println("1");
    }

```

parser

Tato část kódu slouží ke zpracování příkazů, které jsou kladeny na řídicí hardware Arduino Due. V této části se zpracovávají přijaté zprávy a jsou zpracovány podle řídicího algoritmu.

```

2 //-----
// GLOBALS
4 //-----
static char buffer[MAX_BUF]; // where we store the message
    until we get a ';'
6 static int sofar; // how much is in the buffer

8
//-----
10 // METHODS
//-----
12 /**
 * Look for character /code/ in the buffer and read the float
    that immediately follows it.
14 * @return the value found. If nothing is found, /val/ is
    returned.
 * @input code the character to look for.
16 * @input val the return value if /code/ is not found.
 */
18 float parsenumber(char code, float val) {
    char *ptr=buffer;
20     LCDClearLine(2);
    lcd.setCursor(0,2);
22     lcd.print(buffer);
    while(ptr && *ptr && ptr<buffer+sofar) {
24         if(*ptr==code) {
            return atof(ptr+1);
26         }
        ptr=strchr(ptr, ';')+1;
28         if(ptr<buffer) break;
    }
30     return val;

```

```

32 }
34 /**
   * write a string followed by a float to the serial line.
   * Convenient for debugging.
36 * @input code the string.
   * @input val the float.
38 */
void output(char *code, float val) {
40     Serial.print(code);
   Serial.print(F("="));
42     Serial.println(val);
   }
44
46 /**
   * Read the input buffer and find any recognized commands. One
   * G or M command per line.
48 */
void parser_processCommand() {
50     int cmd = parsenumber('G', -1);
   switch(cmd) {
52     case 0: // move linear
   case 1: // move linear
54         hexapod_line( parsenumber('X', (mode_abs?h.ee.pos.x:0)) +
   (mode_abs?0:h.ee.pos.x),
   parsenumber('Y', (mode_abs?h.ee.pos.y:0)) +
   (mode_abs?0:h.ee.pos.y),
56         parsenumber('Z', (mode_abs?h.ee.pos.z:0)) +
   (mode_abs?0:h.ee.pos.z),
   parsenumber('U', (mode_abs?h.ee.r:0)) + (
   mode_abs?0:h.ee.r),
58         parsenumber('V', (mode_abs?h.ee.p:0)) + (
   mode_abs?0:h.ee.p),
   parsenumber('W', (mode_abs?h.ee.y:0)) + (
   mode_abs?0:h.ee.y),
60         feedrate( parsenumber('F', feed_rate) ) );
   break;
62     case 4: pause(parsenumber('P', 0)*1000); break; // dwell
   case 28: hexapod_find_home(); break;
64     case 90: mode_abs=1; break; // absolute mode
   case 91: mode_abs=0; break; // relative mode
66
   default: break;
68     }
70     cmd = parsenumber('M', -1);
   switch(cmd) {

```

```

72 case 17: motor_enable(); break;
73 case 18: motor_disable(); break;
74 case 100: help(); break;
75 case 114: hexapod_where(); break;
76 default: break;
77 }
78 }
79
80
81
82 /* Prepares the input buffer to receive a new message and
83    tells the serial connected device it is ready for more.
84    */
85 void parser_ready() {
86     sofar=0; // clear input buffer
87     Serial.print(F(">")); // signal ready to receive input
88 }
89
90 void LCDClearLine(int l) {
91
92     lcd.setCursor(0, l);
93     for(int ii=0; ii<20; ii = ii+1){
94         lcd.print(" ");
95     }
96 }
97
98 /**
99  * Listen to the serial port for incoming commands and deal
100  * with them
101  */
102 void parser_listen() {
103     // listen for serial commands
104     while(Serial.available() > 0) {
105         char c=Serial.read(); // get it
106         Serial.print(c); // repeat it back so I know you got the
107         message
108         if(sofar<MAX_BUF) buffer[sofar++]=c; // store it
109         if(buffer[sofar-1]==';') break; // entire message received
110     }
111
112     if(sofar>0 && buffer[sofar-1]==';') {
113         // we got a message and it ends with a semicolon
114         buffer[sofar]=0; // end the buffer so string functions
115         work right
116         Serial.print(F("\r\n")); // echo a return character for
117         humans
118         parser_processCommand(); // do something with the command
119         parser_ready();
120     }

```


segment.h

Tato část kódu připravuje strukturu pro segmenty, jelikož posílání kroků na řadiče krokových motorů je řešeno pomocí segmentů. Vytváření segmentů je v následující části kódu.

```
2 #ifndef SEGMENT_H
3 #define SEGMENT_H
4
5 //-----
6 // CONSTANTS
7 //-----
8
9
10 #define MAX_SEGMENTS          (64)
11
12 //-----
13 // STRUCTS
14 //-----
15
16
17 struct Axis {
18     int step_count;
19     int delta;
20     int absdelta;
21     int dir;
22     int over;
23 };
24
25
26 struct Segment {
27     Axis a[NUM_AXIES];
28     int steps;
29     int steps_left;
30     long feed_rate;
31 };
32
33
34 //-----
35 // GLOBALS
36 //-----
37
38 extern Segment line_segments [MAX_SEGMENTS];
```

```

extern volatile int current_segment;
40 extern volatile int last_segment;

42 //-----
44 // METHODS
46 //-----
46 int get_next_segment(int i);
47 int get_prev_segment(int i);
48 void motor_prepare_segment(int n0,int n1,int n2,int n3,int n4,
    int n5,float new_feed_rate);
49 void motor_move_segment(Segment &seg);
50 void motor_move_all_segments();

52 #endif

```

segment

Tato část kódu slouží ke zpracování kroků do segmentů, pro řadiče krokových motorů. Dále je zde funkce *motor()*, která již byla zmíněna a slouží pro realizaci pohybu koncového efektoru.

```

1 //-----
3 // INCLUDES
4 //-----
5 #include "configuration.h"
6 #include "segment.h"
7
9 //-----
11 // GLOBALS
12 //-----
13 // A ring buffer of line segments.
13 Segment line_segments[MAX_SEGMENTS];
14 volatile int current_segment=0;
15 volatile int last_segment=0;
17
19 //-----
21 // METHODS
22 //-----
21 int get_next_segment(int i) {
22     return ( i + 1 ) % MAX_SEGMENTS;
23 }

```

```

25 int get_prev_segment(int i) {
27     return ( i + MAX_SEGMENTS - 1 ) % MAX_SEGMENTS;
29 }

31 /**
32  * Add a segment to the line buffer if there is room.
33  */
34 void motor_prepare_segment(int n0,int n1,int n2,int n3,int n4,
35     int n5,float new_feed_rate) {
36     int next_segment = get_next_segment(last_segment);
37     while( next_segment == current_segment ) {
38         // the buffer is full , we are way ahead of the motion
39         // system
40         delay(1);
41     }

42     Segment &new_seg = line_segments[ last_segment ];
43     new_seg.a[0].step_count = n0;
44     new_seg.a[1].step_count = n1;
45     new_seg.a[2].step_count = n2;
46     new_seg.a[3].step_count = n3;
47     new_seg.a[4].step_count = n4;
48     new_seg.a[5].step_count = n5;

49     int i;

50     Segment &old_seg = line_segments[ get_prev_segment(
51         last_segment) ];
52     new_seg.a[0].delta = n0 - old_seg.a[0].step_count;
53     new_seg.a[1].delta = n1 - old_seg.a[1].step_count;
54     new_seg.a[2].delta = n2 - old_seg.a[2].step_count;
55     new_seg.a[3].delta = n3 - old_seg.a[3].step_count;
56     new_seg.a[4].delta = n4 - old_seg.a[4].step_count;
57     new_seg.a[5].delta = n5 - old_seg.a[5].step_count;

58     new_seg.steps=0;
59     new_seg.feed_rate=new_feed_rate;

60

61     for(i=0;i<NUM_AXIES;++i) {
62         new_seg.a[i].over = 0;
63         new_seg.a[i].dir = (new_seg.a[i].delta * h.arms[i].
64             motor_scale > 0 ? LOW:HIGH);
65         new_seg.a[i].absdelta = abs(new_seg.a[i].delta);
66         if( new_seg.steps < new_seg.a[i].absdelta ) {
67             new_seg.steps = new_seg.a[i].absdelta;
68         }
69     }

```

```

71  if( new_seg.steps==0 ) return;
73  new_seg.steps_left = new_seg.steps;
75  #ifdef VERBOSE
    Serial.print(F("At ")); Serial.println(current_segment);
77  Serial.print(F("Adding ")); Serial.println(last_segment);
    Serial.print(F("Steps= ")); Serial.println(new_seg.steps_left);
79  #endif
81  if( current_segment==last_segment ) {
    timer.setFrequency(new_feed_rate).start();
83  }
85  last_segment = next_segment;
}
87
89  /**
    * Process all line segments in the ring buffer. Uses
    * bresenham's line algorithm to move all motors.
91  */
void motory() {
93  //digitalWrite(13,digitalRead(13)^1);
    // segment buffer empty? do nothing
95  if( current_segment == last_segment ) return;
97  // Is this segment done?
    if( line_segments[current_segment].steps_left <= 0 ) {
99  // Move on to next segment without wasting an interrupt
        tick.
        current_segment = get_next_segment(current_segment);
101  if( current_segment == last_segment ) return;
    }
103
    int j;
105  Segment &seg = line_segments[current_segment];
    // is this a fresh new segment?
107  if( seg.steps == seg.steps_left ) {
        // set the direction pins
109  for(j=0;j<NUM_AXIES;++j) {
            digitalWrite( h.arms[j].motor_dir_pin , line_segments[
                current_segment].a[j].dir );
111  }
        // set frequency to segment feed rate
113  timer.setFrequency(seg.feed_rate).start();
    }
}

```

```

115 // make a step
117 --seg.steps_left;

119 // move each axis
120 for(j=0;j<NUM_AXIES;++j) {
121     Axis &a = seg.a[j];

123     a.over += a.absdelta;
124     if(a.over >= seg.steps) {
125         digitalWrite(h.arms[j].motor_step_pin,LOW);
126         a.over -= seg.steps;
127         digitalWrite(h.arms[j].motor_step_pin,HIGH);
128     }
129 }
}

```

vector

Tato část kódu slouží pro zpracování přepočtů inverzní kinematické úlohy, jelikož standardní knihovna math.h nepodporuje všechny vektorové operace, bylo nutné tyto operace manuálně dopsat.

```

2 #ifndef VECTOR3_H
3 #define VECTOR3_H
4
5 #include <math.h>
6
7
8 class Vector3 {
9 public:
10 // these usions allow the Vector3 to be used as a color
11 // component
12 float x;
13 float y;
14 float z;
15
16 public:
17 inline Vector3() {}
18
19 inline Vector3( float xx, float yy, float zz ) {
20     x = xx;
21     y = yy;
22     z = zz;

```

```

24     }
26     inline Vector3( float v[ 3 ] ) {
27         x = v[ 0 ];
28         y = v[ 1 ];
29         z = v[ 2 ];
30     }
32
33     ~Vector3() {};
34
35     inline Vector3 &MakeZero() {
36         x=0;
37         y=0;
38         z=0;
39
40         return *this;
41     }
42
43
44     inline Vector3 &Set( float xx, float yy, float zz ) {
45         x = xx;
46         y = yy;
47         z = zz;
48
49         return *this;
50     }
51
52
53     inline Vector3 operator + () const { // Unary negation
54         return Vector3(*this);
55     }
56
57
58     inline Vector3 operator - () const { // Unary negation
59         return Vector3( -x, -y, -z );
60     }
61
62
63     inline Vector3 operator *= ( float v ) { // assigned
64         multiply by a float
65         x *= v;
66         y *= v;
67         z *= v;
68
69         return *this;
70     }

```

```

72 inline Vector3 operator /= ( float t ) { // assigned
73     division by a float
74     float v;
75
76     if( t == 0.0f )
77         v = 0;
78     else
79         v = 1.0f / t;
80
81     x *= v;
82     y *= v;
83     z *= v;
84
85     return *this;
86 }
87
88 inline Vector3 operator -= ( const Vector3 &v ) { //
89     assigned subtraction
90     x -= v.x;
91     y -= v.y;
92     z -= v.z;
93
94     return *this;
95 }
96
97
98 inline Vector3 operator += ( const Vector3 &v ) { //
99     assigned addition
100    x += v.x;
101    y += v.y;
102    z += v.z;
103
104    return *this;
105 }
106
107
108 inline Vector3 operator *= ( const Vector3 &v ) { //
109    assigned mult.
110    x *= v.x;
111    y *= v.y;
112    z *= v.z;
113
114    return *this;
115 }

```

```

116 inline Vector3 operator ^= ( const Vector3 &v ) { //
    assigned cross product
    float nx, ny, nz;
118
    nx = ( y * v.z - z * v.y );
120    ny = -( x * v.z - z * v.x );
    nz = ( x * v.y - y * v.x );
122    x = nx;
    y = ny;
124    z = nz;

126    return *this;
    }
128

130 inline bool operator == ( const Vector3 &v ) const {
    return ( fabs( x - v.x ) < 0.01f &&
132            fabs( y - v.y ) < 0.01f &&
            fabs( z - v.z ) < 0.01f );
134 }

136 inline bool operator != ( const Vector3 &v ) const {
138     return ( fabs( x - v.x ) > 0.01f ||
            fabs( y - v.y ) > 0.01f ||
140            fabs( z - v.z ) > 0.01f );
    }
142

144 // METHODS
    inline float Length() const {
146         return (float)sqrt( *this | *this );
    }
148

150 inline float LengthSquared() const {
    return *this | *this;
152 }

154 inline void Normalize() {
156     float len, iLen;

158     len = Length();
    if( !len ) iLen = 0;
160     else iLen = 1.0f / len;

162     x *= iLen;
    y *= iLen;

```



```

164     z *= iLen;
165 }
166
167 inline float NormalizeLength() {
168     float len, iLen;
169
170     len = Length();
171     if( !len ) iLen = 0;
172     else iLen = 1.0f / len;
173
174     x *= iLen;
175     y *= iLen;
176     z *= iLen;
177
178     return len;
179 }
180
181
182 inline void ClampMin( float min ) { // Clamp to minimum
183     if( x < min ) x = min;
184     if( y < min ) y = min;
185     if( z < min ) z = min;
186 }
187
188
189 inline void ClampMax( float max ) { // Clamp to maximum
190     if( x > max ) x = max;
191     if( y > max ) y = max;
192     if( z > max ) z = max;
193 }
194
195
196 inline void Clamp( float min, float max ) { // Clamp to
197     range ]min,max[
198     ClampMin( min );
199     ClampMax( max );
200 }
201
202 // Interpolate between *this and v
203 inline void Interpolate( const Vector3 &v, float a ) {
204     float b( 1.0f - a );
205
206     x = b * x + a * v.x;
207     y = b * y + a * v.y;
208     z = b * z + a * v.z;
209 }
210

```

```

212 inline float operator | ( const Vector3 &v ) const { // Dot
      product
214     return x * v.x + y * v.y + z * v.z;
      }
216
218 inline Vector3 operator / ( float t ) const { // vector /
      float
220     if( t == 0.0f )
        return Vector3( 0, 0, 0 );

222     float s( 1.0f / t );

224     return Vector3( x * s, y * s, z * s );
      }
226
228 inline Vector3 operator + ( const Vector3 &b ) const { //
      vector + vector
230     return Vector3( x + b.x, y + b.y, z + b.z );
      }
232
234 inline Vector3 operator - ( const Vector3 &b ) const { //
      vector - vector
236     return Vector3( x - b.x, y - b.y, z - b.z );
      }
238
240 inline Vector3 operator * ( const Vector3 &b ) const { //
      vector * vector
242     return Vector3( x * b.x, y * b.y, z * b.z );
      }
244
246 inline Vector3 operator ^ ( const Vector3 &b ) const { //
      cross(a,b)
248     float nx, ny, nz;

250     nx = y * b.z - z * b.y;
252     ny = z * b.x - x * b.z;
254     nz = x * b.y - y * b.x;

      return Vector3( nx, ny, nz );
      }

```

```

256     return Vector3( x * s, y * s, z * s );
257 }
258
259 inline void Rotate( Vector3 &axis, float angle ) {
260     float sa = (float)sin( angle );
261     float ca = (float)cos( angle );
262     Vector3 axis2( axis );
263     float m[9];
264
265     axis2.Normalize();
266
267     m[ 0 ] = ca + (1 - ca) * axis2.x * axis2.x;
268     m[ 1 ] = (1 - ca) * axis2.x * axis2.y - sa * axis2.z;
269     m[ 2 ] = (1 - ca) * axis2.z * axis2.x + sa * axis2.y;
270     m[ 3 ] = (1 - ca) * axis2.x * axis2.y + sa * axis2.z;
271     m[ 4 ] = ca + (1 - ca) * axis2.y * axis2.y;
272     m[ 5 ] = (1 - ca) * axis2.y * axis2.z - sa * axis2.x;
273     m[ 6 ] = (1 - ca) * axis2.z * axis2.x - sa * axis2.y;
274     m[ 7 ] = (1 - ca) * axis2.y * axis2.z + sa * axis2.x;
275     m[ 8 ] = ca + (1 - ca) * axis2.z * axis2.z;
276
277     Vector3 src( *this );
278
279     x = m[0] * src.x + m[1] * src.y + m[2] * src.z;
280     y = m[3] * src.x + m[4] * src.y + m[5] * src.z;
281     z = m[6] * src.x + m[7] * src.y + m[8] * src.z;
282 }
283
284 inline operator float *() {
285     return &this->x;
286 }
287 };
288
289 #endif

```

Příloha B

Zde budou uvedeny zdrojové kódy použité v blokách REXlang pro řídicí hardware Raspberry Pi.

sender_rexlang

Tento kód slouží pro komunikaci mezi Raspberry Pi a Arduino Due.

```
*****
2 *
*   REXLANG example – data exchange via serial line   *
4 *
*   Sender station                                     *
6 *
*   (c) REX Controls , www.rexcontrols.com           *
8 *
*****
10 #define COM_DEVICE_FNAME 63 // serial device is defined by
    the fname parameter of the REXLANG block (e.g. /dev/ttyS0 in
    Linux or COM1: in Windows)
12 #define COM_BAUDRATE 115200 //baudrate , e.g. 9600, 19200,
    57600, 115200
14
16 #define COM_PARITY_NONE 0 //no parity
    #define COM_PARITY_ODD 1 //odd parity
18 #define COM_PARITY_EVEN 2 //even parity
20 #define BUFFER_SIZE 50 //maximum number of bytes to send
22 //assigning inputs to variables , these variables are READ-ONLY
    long input(0) stav_automatu; //integer number to send to
    the receiver
24 double input(1) time_atmt;
    double input(2) pozadovane_u;
26 double input(3) pozadovane_v;
```

```

long output(0) c1;
28 long output(1) c2;
long output(2) c3;
30 long output(3) c4;
long output(4) povoleni_regulace;
32

34 long output(15) handle; //handle of the serial device

36 //declaration of variables
long hCom; //communication handle
38 long buffer[BUFFER_SIZE]; //buffer for data
long buffer2[BUFFER_SIZE]; //buffer for data
40 long dataCnt; //number of bytes sent
long dataCnt2; // number of bytes receive
42 long convData1[20]; //array for data conversions
long convData2[20];
44 long signal; //signal from DUE

46 /* Function for conversion of decimal number (the val parameter
) to 2 numbers
of type long (i.e. 8 bytes) representing the number in the
double format
48 according to IEEE 754. Little-endian format is used. */

50 /* Initialization of the REXLANG algorithm */
// the init procedure is executed once when the REXLANG
function block initializes
52 long init(void)
{
54 hCom = -1;
povoleni_regulace = 1;
56 c1=0;
c2=0;
58 c3=0;
c4=0;

60 return 0;
62 }

64 /* The body of the REXLANG algorithm */
// the main procedure is executed once in each sampling period
66
void DoubleToBuffer(double val, long values[])
68 {
long data;
70 val=val*100;
data = (long) val;
72 if(data >0)

```

```

74     {
75         values[0]='+';
76         values[1]= (data/1000)+'0';
77         data=data%1000;
78         values[2]= (data/100)+'0';
79         data=data%100;
80         values[3]='.';
81         values[4]=(data/10)+'0';
82         data=data%10;
83         values[5]=data+'0';
84     }
85     else
86     { data=data*-1;
87       values[0]='-';
88       values[1]= (data/1000)+'0';
89       data=data%1000;
90       values[2]= (data/100)+'0';
91       data=data%100;
92       values[3]='.';
93       values[4]=(data/10)+'0';
94       data=data%10;
95       values[5]=data+'0';
96     }
97
98 }
99
100 long main(void)
101 {
102     switch(stav_automatu)
103     {
104         case 0: break;
105
106         case 1:
107             if (hCom<0)
108             {
109                 hCom = Open(COM_DEVICE_FNAME,COM_BAUDRATE,
110 COM_PARITY_NONE); //opening serial device
111             }
112             if(hCom>0)
113             {
114                 c1=1;
115             }
116             break;
117         case 2:
118             buffer[0] = 'G';
119             buffer[1] = '2';
120             buffer[2] = '8';

```

```

122     buffer[3] = ' ';
    dataCnt = Send(hCom,buffer ,4); //send data , number
of bytes = 4
124     buffer[0] = 'G';
    buffer[1] = '0';
126     buffer[2] = '0';
    buffer[3] = ' ';
128     buffer[4] = 'Z';
    buffer[5] = '2';
130     buffer[6] = '.';
    buffer[7] = '5';
132     buffer[8] = ' ';
    dataCnt = Send(hCom,buffer ,9); //send data , number
of bytes = 9
134     c2=1;
    break;
    case 3:
136         if (time_atmt > 1)
            {
138             c3=1;
            }
140         break;
    case 4:
142         povoleni_regulace = 0;
    DoubleToBuffer(pozadovane_u , convData1);
144     DoubleToBuffer(pozadovane_v , convData2);
    buffer[0] = 'G';
146     buffer[1] = '0';
    buffer[2] = '0';
148     buffer[3] = ' ';
    buffer[4] = 'U';
150     buffer[5] = convData1 [0];
    buffer[6] = convData1 [1];
152     buffer[7] = convData1 [2];
    buffer[8] = convData1 [3];
154     buffer[9] = convData1 [4];
    buffer[10] = convData1 [5];
156     buffer[11] = ' ';
    buffer[12] = 'V';
158     buffer[13] = convData2 [0];
    buffer[14] = convData2 [1];
160     buffer[15] = convData2 [2];
    buffer[16] = convData2 [3];
162     buffer[17] = convData2 [4];
    buffer[18] = convData2 [5];
164     buffer[19] = ' ';

166
//now all the data can be sent

```

```

168         dataCnt = Send(hCom, buffer ,20); //send data ,
           number of bytes = 20
           break;
170         default: break;
           } //end switch
172
173 /* handle=hCom;
174 Trace(1,handle);*/
           return 0;
176 } //end main

178 /* Closing the REXLANG algorithm */
           //the exit procedure is executed once when the task is
           correctly terminated
180 // (system shutdown, downloading new control algorithm, etc.)
           long exit(void)
182 {
           if(hCom>=0) Close(hCom);
184           return 0;
           }

```

stmpe

```

1  /* *****
   *
3  * REXLANG – Reading data from STMPE610 converter via I2C
   *
5  ***** */

7  #define I2CDEV_FNAME 67 // I2C device is defined by the fname
           parameter of the REXLANG block (e.g. set it to /dev/i2c-1 on
           the Raspberry Pi minicomputer)

9  //assigning variables to outputs, these variables are WRITE-
           ONLY
           double output(0) x_value;
11          double output(1) y_value;
           double x1;
13          double y1;
           //declaration of variables
15          long i2c_bufTx [3]; //buffer for transmitting data
           long i2c_bufRx [4]; //buffer for receiving data
17          long i2c_hide [3];
           long i2c_bus_handle;
19          long i2c_chip_address;
           long i2c_write_count;

```



```

21 long i2c_read_count;
   long i2c_ret_fun;
23
   long STMPE_SYS_CTRL1[2];
25 #define STMPE_SYS_CTRL1_RESET 0x02;
   long STMPE_SYS_CTRL2[2];
27
   long STMPE_TSC_CTRL[2];
29 #define STMPE_TSC_CTRL_EN 0x01;
   #define STMPE_TSC_CTRL_XYZ 0x00;
31
   long STMPE_INT_CTRL[2];
33 #define STMPE_INT_CTRL_POL_HIGH 0x04;
   #define STMPE_INT_CTRL_POL_LOW 0x00;
35 #define STMPE_INT_CTRL_EDGE 0x02;
   #define STMPE_INT_CTRL_LEVEL 0x00;
37 #define STMPE_INT_CTRL_ENABLE 0x01;
   #define STMPE_INT_CTRL_DISABLE 0x00;
39
41 long STMPE_INT_EN[2];
   #define STMPE_INT_EN_TOUCHDET 0x01;
43 #define STMPE_INT_EN_FIFOTH 0x02;
   #define STMPE_INT_EN_FIFOOF 0x04;
45 #define STMPE_INT_EN_FIFOFULL 0x08;
   #define STMPE_INT_EN_FIFOEMPTY 0x10;
47 #define STMPE_INT_EN_ADC 0x40;
   #define STMPE_INT_EN_GPIO 0x80;
49
   long STMPE_INT_STA[2];
51 #define STMPE_INT_STA_TOUCHDET 0x01;

53 long STMPE_ADC_CTRL1[2];
   #define STMPE_ADC_CTRL1_10BIT 0x00;
55
   long STMPE_ADC_CTRL2[2];
57 #define STMPE_ADC_CTRL2_6_5MHZ 0x02;

59 long STMPE_TSC_CFG[2];
   #define STMPE_TSC_CFG_4SAMPLE 0x80;
61 #define STMPE_TSC_CFG_DELAY_1MS 0x20;
   #define STMPE_TSC_CFG_SETTLE_5MS 0x04;
63
   long STMPE_FIFO_TH[2];
65
   #define STMPE_FIFO_SIZE 0x4C;
67
   long STMPE_FIFO_STA[2];
69 #define STMPE_FIFO_STA_RESET 0x01;

```

```

71 #define STMPE_FIFO_STA_OFLOW 0x80;
72 #define STMPE_FIFO_STA_FULL 0x40;
73 #define STMPE_FIFO_STA_EMPTY 0x20;
74 #define STMPE_FIFO_STA_THTRIG 0x10;

75 long STMPE_TSC_I_DRIVE[2];
76 #define STMPE_TSC_I_DRIVE_50MA 0x01;
77
78
79 //the init procedure is executed once when the REXLANG function
80 //block initializes
81 int init(void)
82 {
83     i2c_bus_handle = Open(I2CDEV_FNAME); // open I2C bus
84     /* Address: 1 1 0 1 A2 A1 A0 */
85     i2c_chip_address = 0x41; // 7-bit address of the I2C device

86
87     STMPE_SYS_CTRL2[0] = 0x04;
88     STMPE_SYS_CTRL2[1] = 0x0;
89     I2C(i2c_bus_handle, i2c_chip_address, STMPE_SYS_CTRL2, 2,
90         i2c_hide, 0); // turn on clocks!

91     STMPE_TSC_CTRL[0] = 0x40;
92     STMPE_TSC_CTRL[1] = 0x01;
93     I2C(i2c_bus_handle, i2c_chip_address, STMPE_TSC_CTRL, 2,
94         i2c_hide, 0); // XYZ and enable!

95     STMPE_INT_EN[0] = 0x0A;
96     STMPE_INT_EN[1] = 0x01;
97     I2C(i2c_bus_handle, i2c_chip_address, STMPE_INT_EN, 2, i2c_hide
98         , 0);

99     STMPE_ADC_CTRL1[0] = 0x20;
100    STMPE_ADC_CTRL1[1] = 0x96;
101    I2C(i2c_bus_handle, i2c_chip_address, STMPE_ADC_CTRL1, 2,
102        i2c_hide, 0); // 96 clocks per conversion

103    STMPE_ADC_CTRL2[0] = 0x21;
104    STMPE_ADC_CTRL2[1] = 0x02;
105    I2C(i2c_bus_handle, i2c_chip_address, STMPE_ADC_CTRL2, 2,
106        i2c_hide, 0);

107    STMPE_TSC_CFG[0] = 0x41;
108    //STMPE_TSC_CFG[1] = 0xA4;
109    //STMPE_TSC_CFG[1] = 0xB4;
110    STMPE_TSC_CFG[1] = 0xBF;
111    I2C(i2c_bus_handle, i2c_chip_address, STMPE_TSC_CFG, 2, i2c_hide
112        , 0);

```

```

113 STMPE_FIFO_TH[0] = 0x4A;
114 STMPE_FIFO_TH[1] = 0x01;
115 I2C(i2c_bus_handle ,i2c_chip_address ,STMPE_FIFO_TH, 2,i2c_hide
    ,0);

117 STMPE_FIFO_STA[0] = 0x4B;
118 STMPE_FIFO_STA[1] = 0x00;
119 I2C(i2c_bus_handle ,i2c_chip_address ,STMPE_FIFO_STA, 2,
    i2c_hide ,0); // unreset

121 STMPE_TSC_I_DRIVE[0] = 0x58;
122 STMPE_TSC_I_DRIVE[1] = 0x01;
123 I2C(i2c_bus_handle ,i2c_chip_address ,STMPE_TSC_I_DRIVE, 2,
    i2c_hide ,0);

125 STMPE_INT_STA[0] = 0x0B;
126 STMPE_INT_STA[1] = 0xFF;
127 I2C(i2c_bus_handle ,i2c_chip_address ,STMPE_INT_STA, 2,i2c_hide
    ,0);

129 STMPE_INT_CTRL[0] = 0x09;
130 STMPE_INT_CTRL[1] = 0x04;
131 //STMPE_INT_CTRL[1] = 0x81;
132 I2C(i2c_bus_handle ,i2c_chip_address ,STMPE_INT_CTRL, 2,
    i2c_hide ,0);
133
135 return 0;
136 }
137
138 //the main procedure is executed once in each sampling period
139 long main(void)
140 {
141     i2c_bufTx[0] = 0xD7;
142     i2c_write_count = 1;
143     i2c_read_count = 4;
144     //Sending data via I2C
145     i2c_ret_fun = I2C(i2c_bus_handle , i2c_chip_address , i2c_bufTx
        , i2c_write_count , i2c_bufRx , i2c_read_count);
146     //Trace(123,i2c_bufRx[0]);
147     x1 = (i2c_bufRx[0] << 4) | (i2c_bufRx[1] >> 4);
148     y1 = ((i2c_bufRx[1] & 0xf) << 8) | i2c_bufRx[2];
149
150     x1 = (x1-58)/125;
151     y1 = (y1-98)/160;

153     x_value = x1;
154     y_value = y1;

```

```
155     return 0;
156 }
157
158 //the exit procedure is executed once when the task is
159 //correctly terminated
160 // (system shutdown, downloading new control algorithm, etc.)
161 long exit(void)
162 {
163     if(i2c_bus_handle >=0) Close(i2c_bus_handle); // close I2C bus
164     return 0;
165 }
```

Literatura

- [1] REX pro Raspberry Pi, 2000-2016. <https://www.rexcontrols.cz/ridici-system-rex-raspberry-pi> [Online; navštíveno 20. 01. 2016]].
- [2] Raspberry Pi 2 Model B, 2015. <https://www.raspberrypi.org/products/raspberry-pi-2-model-b/> [Online; navštíveno 12. 01. 2016]].
- [3] Arduino - ArduinoBoardDue, 2016. <https://www.arduino.cc/en/Main/ArduinoBoardDue> [Online; navštíveno 05. 01. 2016]].
- [4] Marginally Clever Robots 2016. Rotary stewart platform v2 - marginally clever robots, 2013. <https://www.marginallyclever.com/product/rotary-stewart-platform-v2/> [Online; navštíveno 10. 10. 2015]].
- [5] Arduino. Arduino - compare, 2016. <https://www.arduino.cc/en/Products/Compare> [Online; navštíveno 20. 03. 2016]].
- [6] Cosmo Electronics Corp. K1010 series, 2015. <http://www.cosmo-ic.com/object/products/k1010.pdf> [Online; navštíveno 10. 12. 2015]].
- [7] Pololu Corporation. Stepper motor: Unipolarbipolar, 2001 - 2016. <https://www.pololu.com/product/1200> [Online; navštíveno 20. 03. 2015]].
- [8] Euer Angelo. Radds - radds electronics for 3d printer, 2015. <http://doku.radds.org/dokumentation/radds/> [Online; navštíveno 12. 12. 2015]].
- [9] Adafruit Industries. Resistive touch screen controller - stmpe610, 2015. <https://www.adafruit.com/product/1571> [Online; navštíveno 7. 05. 2016]].

- [10] Pololu Corporation. Pololu - a4988 stepper motor driver carrier, 2001-2016. <https://www.pololu.com/product/1182> [Online; navštíveno 12. 02. 2014]].
- [11] Rexcontrols. Github - rexcontrols/rexexamples: Example projects for the rex control system, 2015. <https://github.com/rexcontrols/REXexamples> [Online; navštíveno 10. 02. 2016].
- [12] Rexcontrols. Funkční bloky systému REX, 2016. <https://www.rexcontrols.cz/media/HTML/DOC/CZECH/index.html> [Online; navštíveno 10. 02. 2016].
- [13] Jaroslav Růžička. Návrh, modelování a řízení manipulátoru se šesti stupni volnosti. Bakalářská práce, Západočeská univerzita v Plzni, Fakulta aplikovaných věd, Plzeň, 2014.
- [14] Dan Royer. Rotarystewartplatform2, 2013. <https://github.com/MarginallyClever/RotaryStewartPlatform2> [Online; navštíveno 10. 10. 2014]].
- [15] Wokingham U3A. Maths, 2011. <http://www.wokinghamu3a.org.uk/Maths.html> [Online; navštíveno 20. 12. 2013]].
- [16] Wikipedia. Stewart platform — wikipedia, the free encyclopedia, 2015. https://en.wikipedia.org/w/index.php?title=Stewart_platform&oldid=696678744 [Online; navštíveno 20. 01. 2016]].
- [17] Wikipedia. G-code — wikipedia, the free encyclopedia, 2016. <https://en.wikipedia.org/w/index.php?title=G-code&oldid=718587457> [Online; navštíveno 6. 05. 2016]].
- [18] Wikipedie. Krokový motor — wikipedie: Otevřená encyklopedie, 2013. https://cs.wikipedia.org/w/index.php?title=Krokov%C3%BD_motor&oldid=11043557 [Online; navštíveno 7. 05. 2016].