

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
STROJNÍCKA FAKULTA**

**AEROSHIELD: MINIATÚRNY EXPERIMENTÁLNY MODUL
AEROKYVADLA**

Bakalárska práca

SjF-číslo b. práce

2022

Peter Tibenský

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
STROJNÍCKA FAKULTA**

**AEROSHIELD: MINIATÚRNY EXPERIMENTÁLNY MODUL
AEROKYVADLA**

Bakalárska práca

SjF-12345-67890

Študijný odbor:	Automatizácia a informatizácia strojov a procesov
Študijný program:	5.2.14 automatizácia
Školiace pracovisko:	Ústav automatizácie, merania a aplikovanej informatiky
Vedúci záverečnej práce:	Ing. Mgr. Anna Vargová.
Konzultant:	Ing. Erik Mikuláš

Bratislava, 2022

Peter Tibenský

Úlohou študenta je navrhnúť, realizovať a sériovo vyrobiť rozširovací modul pre prototypizačnú platformu Arduino v rámci open-source projektu „AutomationShield“. Jedná sa o návrh miniaturizovaného laboratórneho experimentu so spätnoväzobným riadením tzv. aerokyvadla, spolu s ovládacím softvérom a inštruktážnymi príkladmi. Študent navrhne plošný spoj v CAD prostredí DipTrace, vytvorí programátorské rozhranie (API) v jazyku C/C++ pre Arduino IDE, ďalej pre MATLAB a Simulink. Študent manažuje verzie projektu v Git pre GitHub a píše úplnú dokumentáciu v MarkDown.

Čestné prehlásenie

Vyhlasujem, že predloženú záverečnú prácu som vypracoval samostatne pod vedením vedúceho záverečnej práce, s použitím odbornej literatúry a ďalších informačných zdrojov, ktoré sú citované v práci a uvedené v zozname použitej literatúry. Ako autor záverečnej práce ďalej prehlasujem, že som v súvislosti s jej vytvorením neporušil autorského práva tretích osôb.

Bratislava, 23. máj 2022

.....
Vlastnoručný podpis

V prvom rade by som sa chcel poďakovať vedúcej mojej bakalárskej práce, Ing. Mgr. Anne Vargovej, za výborne kávičky v kabinete, najlepšie meme obrázky v skupine ako aj za cenné rady a nekonečné opravy tejto práce. Ďalej chcem poďakovať aj Ing. Erikovi Mikulášovi, za pomoc pri tvorbe a kontrole (skoro)všetkých elektronických komponentov AeroShieldu, ako aj za pevné ruky pri spájkovaní. Ďakujem aj mojej partnerke Slávke, za to že každý deň trpezlivо počúvala moje nekonečné nadávky, stаžnosti ale aj radosti, pri tvorbe tejto práce. Zároveň sa chcem ospravedlniť spolužiakom, za stres ktorý mali z toho, keď som sa chválil 40. stranou práce, zatiaľ čo podaktorý ešte nepoznali tému ich BP.

Bratislava, 20. mája 2018

Peter Tibenský

Názov práce: AeroShield: Miniatúrny experimentálny modul aerokyvadla

Kľúčové slová: Arduino, AutomationShield, PID, AeroShield, AeroPendulum

Abstrakt: Cieľom bakalárskej práce je návrh experimentálneho modulu pre platformu Arduino. Tento modul má podobu externého shieldu, ktorý sa dá jednoducho pripojiť ku doskám Arduino a slúži na výučbu základov riadenia. Ich súčasťou je hardwareova a softwaerova časť. V rámci bakalárskej práce bol navrhnutý jeden modul s názvom AeroShield.

Title: AeroShield: Miniature experimental module of aeropendulum

Keywords: Arduino, AutomationShield, PID, AeroShield, AeroPendulum

Abstract: The aim of the bachelor's thesis is to design an experimental module for the Arduino platform. This module takes the form of an external shield that can be easily connected to Arduino boards and is used to teach the basics of control. Each module consists of hardware and a software part. As a part of this bachelor thesis, one module was designed, the AeroShield.

Obsah

Zoznam obrázkov	i
Zoznam zdrojových kódov	iii
Úvod	1
1 AeroShield	5
1.1 Hardvér	7
1.1.1 Popis súčiastok	7
1.1.2 Schéma zapojenia	11
1.1.3 Doska plošných spojov	12
1.1.4 Model držiaku kyvadla	13
1.1.5 Cenová kalkulácia AeroShieldu	15
1.1.6 Tretia verzia AeroShieldu	15
1.2 Softvér	17
1.2.1 Arduino API	17
1.2.2 MATLAB	26
1.2.3 Simulink	29
2 Didaktické príklady	39
2.1 Programy v otvorenej slučke, bez spätnej väzby	39
2.1.1 Arduino IDE	39
2.1.2 MATLAB	41
3 PID regulácia	45
3.1 Programy v uzavorennej slučke, so spätnou väzbou	47
3.1.1 Arduino IDE	47
3.1.2 MATLAB	51
3.1.3 Simulink	55
4 Záver	57
Literatúra	59
Arduino IDE	v
Zdrojový kód AeroShield.h	v
Zdrojový kód AeroShield.cpp	vi

Zdrojový kód AeroShieldOpenLoop.ino	viii
Zdrojový kód AeroShieldPID.ino	ix
MATLAB	
Zdrojový kód AeroShield.m	xi
Zdrojový kód AeroShieldOpenLoop.m	xii
Zdrojový kód AeroShielPID.m	xiii

Zoznam obrázkov

1	Experimentálne moduly vzdušného kyvadla.	2
2	Arduino UNO R3.[1]	3
3	Arduino Mega 2560 R3.[2]	4
1.1	Prvá verzia AeroShieldu.	5
1.2	Meranie uhla kyvadla	6
1.3	buck converter	7
1.4	Zapojenie akčného člena a typ motorčeka	8
1.5	meranie prúdu	9
1.6	Schematická reprezentácia hallovho javu.	9
1.7	meranie uhla kyvadla	10
1.8	Schéma zapojenia AeroShieldu.	11
1.9	Vedľajšia doska AeroShieldu- breakout board.	12
1.10	(a) Vrchná strana AeroShieldu. (b) Spodná strana AeroShieldu.	13
1.11	Dosky plošných spojov AeroShieldu.	14
1.12	Model kyvadla, tvorený v programe CATIA.	14
1.13	Tretia verzia AeroShieldu.	16
1.14	Schéma zapojenia tretej verzie AeroShieldu.	16
1.15	Knižnica AeroLibrary.	29
1.16	Reference read- Simulink.	30
1.17	Automatická trajektória- Simulink.	31
1.18	Angle read- Simulink.	32
1.19	Mapovanie uhlu kyvadla- Simulink.	33
1.20	Kalibrácia- Simulink.	33
1.21	Podsystém na kontrolu uhlu kyvadla.	34
1.22	Nastavenia parametrov masky bloku Reference read.	36
1.23	Reference read- Simulink.	36
1.24	AeroShield_OpenLoop.	37
2.1	Výstup z programu AeroShieldOpenLoop.ino.	40
2.2	Výstup z programu AeroShieldOpenLoop.m.	43
3.1	Schéma riadenia PID regulátorom.	45
3.2	Reakcia systému na jednotkový skok.	51
3.3	Automatická trajektória.	51
3.4	Manuálna trajektória.	52
3.5	Manuálne zavedenie výchylky.	52

3.6	Reakcia systému na jednotkový skok.	53
3.7	Automatická trajektória.	54
3.8	Manuálna trajektória.	54
3.9	Ukážka riadenia systému pomocou PID regulátora v API Simulink. . .	55
3.10	Reakcia systému na skokovú zmenu.	56
3.11	Manuálna trajektória.	56

Zoznam zdrojových kódov

1.1	Ukážka zdrojového kódu headeru.	17
1.2	Zdrojový kód funkcie mapFloat.	19
1.3	Zdrojový kód funkcie serialPrint.	19
1.4	Zdrojový kód funkcie readOneByte.	20
1.5	Zdrojový kód funkcie readTwoBytes.	21
1.6	Zdrojový kód funkcie detectMagnet.	21
1.7	Zdrojový kód funkcie getMagnetStrength.	22
1.8	Zdrojový kód funkcie getRawAngle.	22
1.9	Zdrojový kód funkcie begin.	23
1.10	Zdrojový kód funkcie calibration.	24
1.11	Zdrojový kód funkcie convertRawAngleToDegrees.	24
1.12	Zdrojový kód funkcie referenceRead.	25
1.13	Zdrojový kód funkcie actuatorWrite.	25
1.14	Zdrojový kód funkcie currentMeasure.	26
1.15	Knižnica AeroShield.m properties.	27
1.16	Knižnica AeroShield.m properties.	27
1.17	MATLAB function blok autiomatická trajektoria.	30
1.18	Mapovacia funkcia vo fcn bloku.	32
1.19	Callback funkcia.	34
1.20	Initialization- maska Reference read.	35
2.1	AeroShield open loop dekleracia.	39
2.2	AeroShield open loop setup().	39
2.3	AeroShield open loop loop().	40
2.4	AeroShield open loop inicializacia.	41
2.5	AeroShield open loop grafy.	41
2.6	AeroShield open loop, while cyklus.	42
3.1	Načítanie knižníc a premenných do programu.	47
3.2	Organzačná funkcia setup.	48
3.3	Funkcia stepEnable().	48
3.4	Organzačná funkcia loop.	49
3.5	Funkcia step().	50
4.1	Zdrojový kód súboru AeroShield.h.	v
4.2	Zdrojový kód súboru AeroShield.cpp.	vi
4.3	Zdrojový kód súboru AeroShieldOpenLoop.ino.	viii
4.4	Zdrojový kód súboru AeroShieldPID.ino.	ix
4.5	Zdrojový kód súboru AeroShield.m.	xi

4.6	Zdrojový kód súboru AeroShieldOpenLoop.m.	xii
4.7	Zdrojový kód súboru AeroShieldPID.m.	xiii

Úvod

Cieľom tejto práce je návrh, výroba a naprogramovanie modernej učebnej pomôcky AeroShieldu (ďalej len „shield“), ktorá slúži na výučbu základov teórie riadenia a elektrotechniky. Učebné pomôcky sú nevyhnutnou, no často zanedbávanou súčasťou výučby. Študenti si vďaka nim môžu lepšie predstaviť a pochopiť problematiku daného učiva, keďže môžu pracovať nie len s počítačovými modelmi sústavy, ale aj s jej fyzickou reprezentáciou. Avšak, takéto pomôcky bývajú častokrát príliš zložité na používanie a priveľmi drahé [3]. Z týchto dôvodov je ich použitie pri výučbe častokrát nepraktické.

Experimentálny modul vzdušného kyvadla je pomerne jednoduché zariadenie, po zostávajúce z niekoľkých častí. Akčným členom kyvadla je motorček, ktorý má na rotor pripojené lopatky, ktoré vďaka otáčaniu produkujú ťah. Motorček je zvyčajne upevnený na koniec ťahkej tyčky, ktorá je v mieste otáčania pripevnená k zariadeniu na meranie uhlu pootočenia. Zariadenie na meranie pootočenia môže byť potenciometer, senzor hallovho javu(efektu), alebo iné [4]. Zariadenie na meranie uhlu je následne upevnené na podstavec, ktorý zariadenie stabilizuje a umožňuje volný pohyb kyvadla.

Tvorba AeroShieldu bola inšpirovaná experimentom známym pod názvom Aerpendulum, čo v doslovnom preklade znamená vzdušné kyvadlo. Na túto tému existuje niekoľko odborných článkov, ktoré sa zaoberajú zostavením, ovládaním, alebo simuláciami takéhoto kyvadla. Medzi najviac citované články patria práce autorov Mila Mary Job a P. Subha Hency Jose[5] a dvojice Eniko T. Enikov a Giampiero Campa[6]. Práca Mila Mary Job a P. Subha Hency Jose bola zameraná hlavne na simuláciu kyvadla a matematiku, ktorá je na takúto simuláciu potrebná. Kyvadlo od autorov Eniko T. Enikov a Giampiero Campa vznikalo na univerzite Arizona. Ovládané bolo pomocou špeciálne navrhnutej dosky plošných spojov a ktorá sa programovala v softvéri Simulink.

Na Arizonský projekt nadviazali aj dve záverečné práce vypracované na Strojníckej fakulte Slovenskej technickej univerzity v Bratislave. Boli to diplomové práce študentov Andreja Poláka[7] a Jakuba Onderu[8]. Tieto práce sa zaoberali vylepšením Arizonského kyvadla, lepším pohonom, presnejším ovládaním, rôznymi meraniami polohy a zrýchlenia a zmenou ovládacieho modulu za mikrokontrolér Arduino. Cena tohto kyvadla bola vzhľadom na použité materiály a množstvo súčiastok pomerne vysoká, rádovo stovky eur.

Existuje niekoľko spoločností, ktoré na predaj ponúkajú hotové experimentálne moduly vzdušného kyvadla. Konkrétnie sa jedná napríklad o kyvadlo značky Real Sim obr.1.a, ktorá ponúka hotový, zostavený modul. Cenu tohto kyvadla sa nám žiaľ nepodarilo dohľadať. Ďalším takýmto modulom je kyvadlo od univerzity Arizona[6] obr.1.b, ktoré je predávané ako nezostavený model. Cena tohto kyvadla je 95€.



(a) Aeropendulum značky Real Sim[9].

(b) Aeropendulum univerzity Arizona[6].

Obr. 1: Experimentálne moduly vzdušného kyvadla.

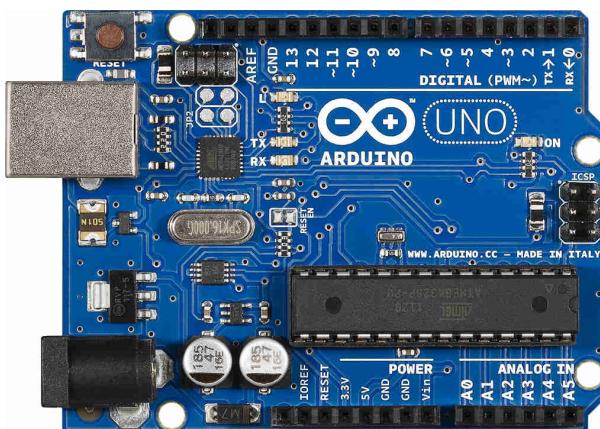
Open-source¹ projekt AutomationShield je zameraný na vývoj hardwarových a softwarových nástrojov určených na vzdelávanie a doplnenie vzdelávacieho procesu. Jadrom celého projektu je tvorba rozširujúcich dosiek (shieldov) vyvíjaných pre populárny typ prototypizačných dosiek s mikrokontrolérmi Arduino. Tieto pomerne lacné učebné pomôcky, majú za cieľ lepšiu výučbu strojného inžinierstva, mechatroniky a základov automatického riadenia[10].

Všetky informácie ohľadom projektu AutomationShield, sú dostupné na platforme GitHub[11], ktorá slúži ako knižnica kódov, návodov a postupov, ktoré sú voľne dostupné na čítanie a úpravu. Na samostatnej stránke AutomationShieldu nájdeme zoznam jednotlivých shieldov a to, v akom procese výroby a fungovania sa nachádzajú. Ku každému shieldu nájdeme jeho podrobnú dokumentáciu, knižnice, zdrojové kódy, ako aj predprogramované ukážky jeho fungovania. GitHub je open-source platforma. Dokumenty zdieľané na tejto stránke teda môže ktokoľvek upravovať, kontrolovať alebo vylepšovať, čo tvorí ideálny priestor pre rozvoj nových myšlienok a podporu tvorivého procesu.

¹Open-source je zo všeobecného pohľadu akákoľvek informácia, ktorá je dostupná verejnosti bez poplatku(s voľným prístupom), s ohľadom na fakt, že jej voľné šírenie zostane zachované.

Hlavnou motiváciou tohto projektu je nízka dostupnosť a vysoká cena podobných učebných pomôcok. Z môjho pohľadu je výučba častokrát až príliš zameraná na memorovanie faktov a teórie, namiesto praktických experimentov a skúseností typu pokus-omyl. Študenti pochopia vyučovanú teóriu jednoduchšie, pokiaľ majú možnosť experimenty sami tvoriť, skúmať a testovať[12].

S úmyslom priniesť širokej verejnosti lacnejšiu a výkonnejšiu alternatívu vtedajším mnohonásobne drahším a menej výkonným prototypizačným doskám[13] prišla na trh v roku 2005 prototypizačná doska Arduino. Projekt vznikol v Taliansku ako kolaborácia medzi viacerými nadšencami elektrotechniky a programovania, na ktorého čele bol Massimo Banzi. Veľkou výhodou dosiek Arduino a ich nadstavbových shieldov je fakt, že sú pomerne lacné a majú malé rozmer (Arduino UNO: 68.6*53.4mm[14]). Tieto skutočnosti umožňujú študentom pracovať na experimentoch nielen na pôde školy, ale experimenty si môžu zobrať domov a pracovať na nich aj mimo vyučovacieho procesu.



Obr. 2: Arduino UNO R3.[1]

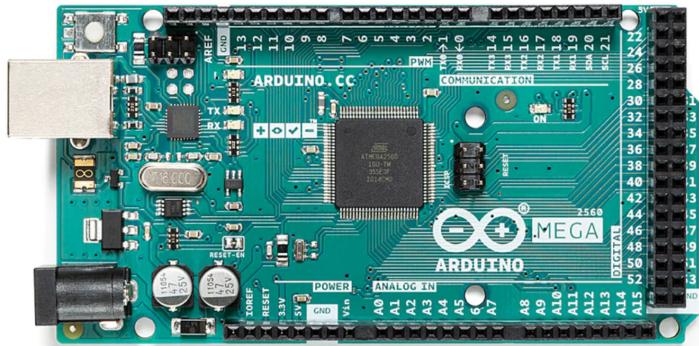
Na fungovanie a programovanie dosky postačuje len USB kábel, programovací softvér a samotná doska. Vzhľadom na nízky počet potrebných súčiastok a fakt, že mikročip Arduina je v prípade poruchy jednoducho vymeniteľný², je jeho používanie na školách príjemné a jednoduché. Mikrokontroléri Arduino využívame z dôvodu nízkej ceny, širokej dostupnosti rôznych modelov, postačujúcej výpočtovej sile a príjemnému používateľskému rozhraniu. Pre naše účely využívame dve verzie Arduina. Prvou z nich je doska Arduino UNO R3 obr.2, ktorú používame na programy Arduino API. Na doske sa nachádza 14 digitálnych a 6 analógových pinov.

Na prácu v MATLABE a Simulinku využívame Arduino Mega 2560 R3 obr.3. Na tejto doske sa nachádza 54 digitálnych a 16 analógových pinov. AeroShield je kompatibilný so všetkými doskami s označením R3, alebo s doskami ktoré majú rozloženie pinov rovnaké ako Arduino UNO R3.

Niektoré piny sú označené špeciálnym symbolom "˜". Tieto piny sú schopné generovať PWM³ signál, ktorý využívame na ovládanie motora kyvadla.

²Platí pri mikročipoch typu DIP(Dual in-line package), ktoré stačí jednoducho vytiahnuť z konektora bez použitia spájkovania.

³Šírková modulácia impulzov alebo PWM je technika na dosiahnutie analógových výsledkov pomocou digitálnych prostriedkov a to za pomoci striedania dĺžok medzi High a Low stavom, resp. zapnutý a vypnutý stav.



Obr. 3: Arduino Mega 2560 R3.[2]

Práca je rozdelená do štyroch logických celkov. Na začiatku v časti Hardvér je opísaný základný princíp fungovania shieldu a následne jeho jednotlivé súčiastky. Nasleduje časť tvorby schémy zapojenia shieldu a dosky plošných spojov v programe DipTrace. Na konci časti Hardvér je spomenutá tvorba modelu kyvadla a cenová kalkulácia výroby experimentálneho modulu.

V softvérovej časti sú bližšie predstavené spôsoby programovania shieldu. Opisuje sa tu tvorba knižníc jednotlivých programov, v ktorých sú tvorené didaktické príklady pre AeroShield.

Poslednú časť práce tvoria samotné didaktické príklady nasledované finálnym zhodenotením práce.

1 AeroShield

Práca je založená na už započatom projekte vzdušného kyvadla. Na jeho tvorbe sa najviac podieľali študenti: Dávid Vereš, Ján Boldocký, Tadeas Vojtko a Denis Skokan. Prvá verzia dosky a samotného kyvadla, vznikla ako záverečný projekt na predmet Mikropočítače a mikroprocesorová technika. Fotografiu zostavenej dosky, môžeme vidieť na obr.1.1.



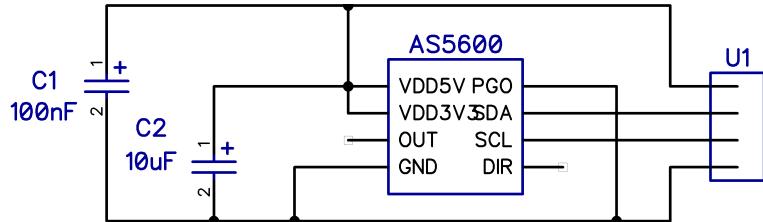
Obr. 1.1: Prvá verzia AeroShieldu.

V novej verzii dosky bolo odstránených niekoľko nedostatkov predchádzajúcej verzie. Jednalo sa o:

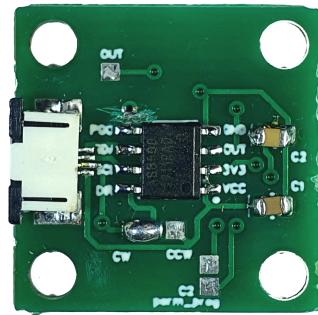
- neprepojenie pinov komunikácie I2C tj. piny SDA a SCL senzoru hall efektu, ktorý slúži na meranie uhlia natočenia kyvadla,
- nesprávne zapojenie mosfetu PMW45EN, ktorý ovláda PWM signál idúci do akčného člena,
- nesprávne umiestnená ochranná dióda na konektoroch akčného člena,
- nesprávne zapojený obvod s čipom INA169, ktorý slúži na meranie prúdu,
- neprepojenie nulového konektora shieldu s nulovým konektorm arduina.

Základom tejto bakalárskej práce teda bolo pochopiť jednotlivé časti zapojenia, analyzovať chyby a ich následná oprava. V rámci projektu bola vytvorená hlavná doska, na ktorej sa nachádza väčšina elektroniky a menšia doska tzv. breakout board obr.1.2.b,

ktorý je uchytený v hornej časti kyvadla a slúži na fungovanie senzoru hall efektu. Táto doska fungovala bezproblémovo a teda nebolo potrebné nijakým spôsobom meniť jej schému zapojenia obr.1.2.a. Breakout boardu sa budeme bližšie venovať v časti 1.1.3.



(a) Schéma zapojenia breakout boardu.



(b) Breakout board.

Obr. 1.2: Meranie uhla kyvadla

1.1 Hardvér

1.1.1 Popis súčiastok

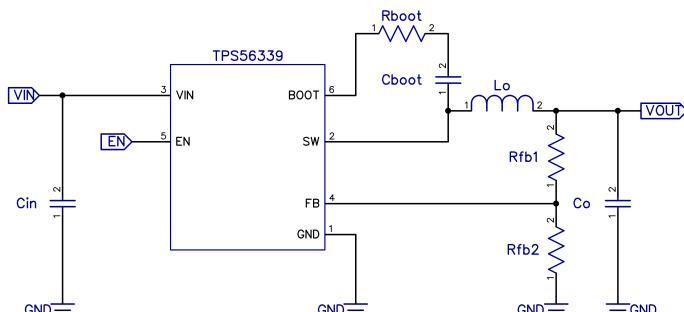
V tejto časti sa bližšie pozrieme na jednotlivé súčasti zapojenia AeroShieldu. Konkrétnie sa jedná o tieto prvky:

- napájanie
- ovládanie akčného člena
- meranie uhla natočenia kyvadla
- meranie prúdu

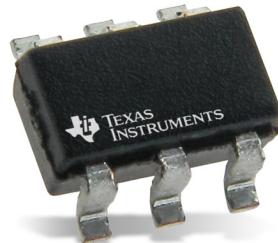
Znižovací menič

Na napájanie akčného člena, motorčeka, potrebujeme napätie v rozmedzí 0-3,7V. Na shield je však privádzané, pomocou koaxiálneho napájacieho konektora, napätie 12V, ktoré by mohlo motor pri dlhšom používaní zničiť. Na zníženie napäcia preto použijeme znižovací menič tzv. buck converter.

Hlavnou časťou konvertora je čip TPS56339 od výrobcu Texas Instruments obr.1.3.b. Znižovanie napäcia funguje za pomoci dvoch integrovaných N-kanálových $70\text{-m}\Omega$ a $35\text{-m}\Omega$ high-side mosfetov⁴, v spolupráci s ďalšími komponentami. Celkový prevádzkový prúd zariadenia je približne $98\mu\text{A}$, keď funguje bez spínania a bez záťaže. Keď je zariadenie vypnuté, napájací prúd je približne $3\mu\text{A}$ a zariadenie umožňuje nepretržitý výstupný prúd do 3 A[15].



(a) Schéma zapojenia znižovacieho meniča.



(b) čip TPS56339.[15]

Obr. 1.3: buck converter

Na čip je privádzané napätie 12V ktoré sa pomocou zapojenia viditeľného na schéme obr.1.3.a, znižuje na napätie 3,7V. Domnievali sme sa že napájanie motora musí byť realizované externe, pomocou koaxiálneho napájacieho konektora z dôvodu vysokého prúdu odoberaného motorom počas silného zataženia. Rovnaký konektor sa súčasne nachádza aj na doske Arduino UNO a pomocou VIN pinu sa z neho dajú napájať napäťom 6-12V aj iné zariadenia, avšak tento pin je napojený na diódu, obmedzujúcu prúd na 1A[16][17].

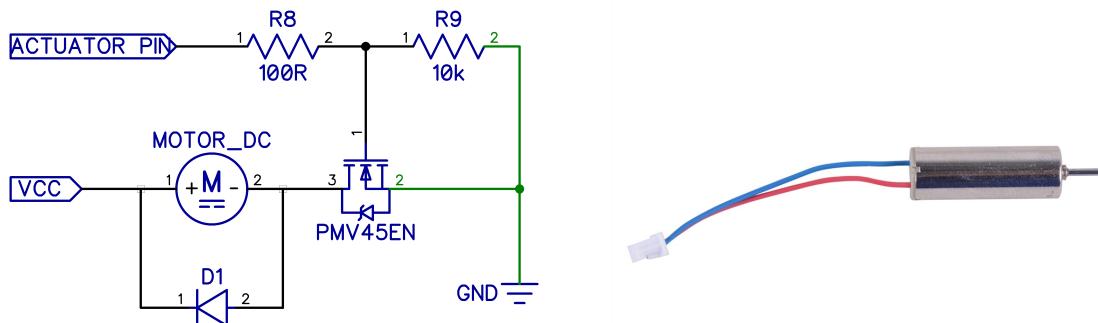
⁴N-kanálový mosfet je typ mosfetu, v ktorom tok prúdu nastáva kvôli pohybujúcim sa, záporne nabitém elektrónom. "High - side" znamená, že prúd prechádza z napájania, cez mosfet, do záťaže a potom do zeme.

Pri testovaní prototypov druhej verzie AeroShieldu, sa merané hodnoty prúdu pochybovali nad hodnotu 1A, čo by zničilo spomínanú internú diódu na doske Arduino UNO. Po zostavení druhej verzie shieldu a opäťovnom meraní odoberaného prúdu, bola maximálna dosahovaná hodnota menšia ako 0,5A. Z tohto dôvodu sme sa rozhodli pre zmenu v schéme kvyadla z ktorej sme odobrali externý napájací konektora následne bola vyrobená nová doska plošných spojov. Tretej verzii AeroShieldu sa bližšie venujeme v časti...

Akčný člen

Ako akčný člen AeroShieldu je použitý 7mm, 3,7V motorček na jednosmerný prúd bez jadra, používaný hlavne pre pohon dronov. "Coreless motor", alebo motor bez jadra, je motor s cievkou navinutou samou na sebe a nie na železe[18]. Stator je vyrobený z magnetov na báze vzácných zemín, ako je neodým alebo SmCo(samárium-kobalt).

Takýto motor ponúka mnoho výhod oproti motoru so železným jadrom. Tým že jadro v sebe nemá železo, výrazne sa znižuje hmotnosť a tým aj zotrvačnosť rotora, čo je dôležité pre naše použitie, kedy potrebujeme dosahovať vysokú akceleráciu a rýchle spomalenie rotora. Ďalšou výhodou je fakt, že nedochádza k stratám na železe a tým pádom sa účinnosť takýchto motorov blíži až ku 90%[19]. Motor, resp. otáčky motora sú riadené pomocou PWM signálu a ten do motoru prechádza cez N-kanálový mosfet PMV45EN2 od výrobcu Nexperia[20].



(a) Schéma zapojenia motorčeka.

(b) Akčný člen sústavy.[21]

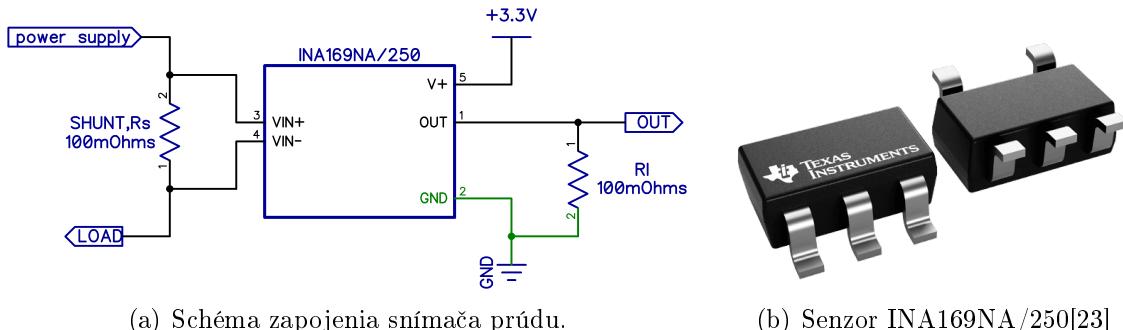
Obr. 1.4: Zapojenie akčného člena a typ motorčeka

Meranie prúdu

Z dôvodu merania prúdu odoberaného motorom, bol do schémy pridaný monitor prúdu, takzvaný "current shunt monitor". Nameraný prúd môžeme využiť na implementáciu riadenia motora na základe prúdu, ktorý odoberá. AeroShield používa snímač INA169NA/250 od výrobcu Texas Instruments obr.1.5.b.

INA169 funguje na základe zaznamenávania zmien napäťia na stranách shunt rezistora obr.1.5.a. Na základe nameraného úbytku napäťia, vysiela senzor podľa nami zvoleného stupňa zosilnenia, prúd ktorý je ďalej pomocou rezistoru R_l premenený na napätie s maximálnou hodnotou $V_{OUTMAX} = V_{IN-} - 0.5V$.

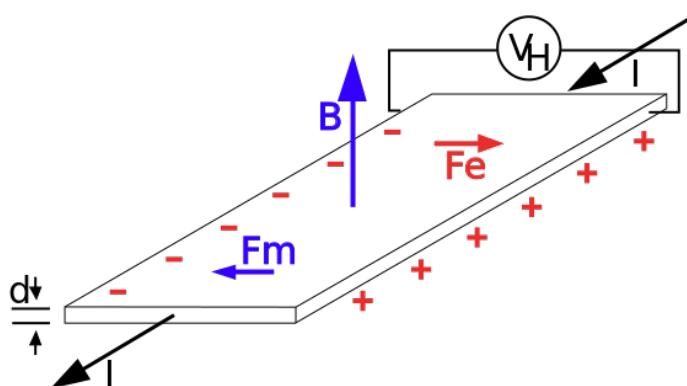
Prúd I_s odoberaný motorom, vypočítame pomocou vzorca $I_s = (V_{OUT} \times 1k\Omega) / (R_s \times R_l)$ kde V_{OUT} je napätie namerané na výstupe, $1k\Omega$ je konštantá vnútorných odporov senzoru, R_s je hodnota shunt rezistora v Ω a R_l je hodnota rezistora na výstupe, taktiež v Ω [22].



Obr. 1.5: meranie prúdu

Meranie uhlia kyvadla

Na fungovanie AeroShieldu je dôležité vedieť s vysokou presnosťou merat uhol náklonenia kyvadla. Na tento účel sme si zvolili meranie uhlia bezkontaktnou formou, pomocou snímača hall efektu. Hallov jav vieme opísť ako vznik priečneho elektrického poľa v pevnom materiáli, keď ním preteká elektrický prúd a tento materiál je umiestnený v magnetickom poli, ktoré je kolmé na prúd[24]. Toto elektrické pole resp. vznik elektrického potenciálu vieme detegovať ako Hallovo napätie a na základe jeho zmeny, vieme určiť rotáciu kyvadla. Fyzikálna podstata tohto javu je na obr.1.6, kde V_H je Hallovo napätie, B je magnetické pole, F_m magnetická sila pôsobiaca na negatívne prenášače náboja, F_e elektrická sila z nahromadeného náboja, I je dohodnutý smer elektrického prúdu.

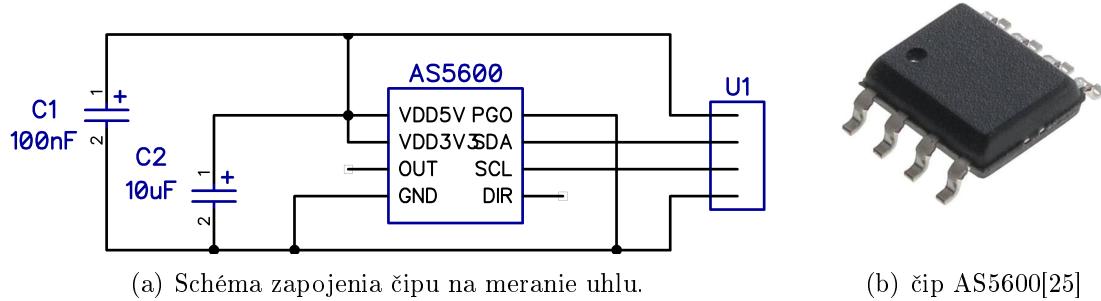


Obr. 1.6: Schematická reprezentácia hallovho javu.

V kyvadle je na konci horizontálneho ramena umiestnený špeciálny magnet kruhového tvaru, ktorý je polarizovaný naprieč prierezom magnetu. Ako senzor na meranie

hall efektu je použitý AS5600 od výrobcu OSRAM obr.1.7.b. Signály prichádzajúce zo snímača sa najprv zosilnia, následne sú filtrované a prechádzajú konverziou pomocou analógovo-digitálneho prevodníka(ADC). Snímaná je aj intenzita magnetického poľa, ktorou senzor pomocou automatického riadenia zosilnenia(AGC), kompenzuje zmeny teploty priestoru a taktiež zmeny sily magnetického poľa.

Na výber sú dva typy výstupu a to analógový výstup alebo digitálny výstup s kódovaním PWM. Senzor má taktiež možnosti interného programovania pomocou rozhrania I2C. V našom prípade používame 12-bitový analógový výstup s rozlíšením $0^{\circ}5'16''$. Toto rozlíšenie nám umožňuje s vysokou presnosťou kontrolovať naklonenie kyvadla a na základe získaných informácií ovplyvňovať fungovanie akčného členu sústavy. Schéma zapojenia čipu na meranie uhlu môžeme vidieť na obr.1.7.a.



Obr. 1.7: meranie uhla kyvadla

1.1.2 Schéma zapojenia

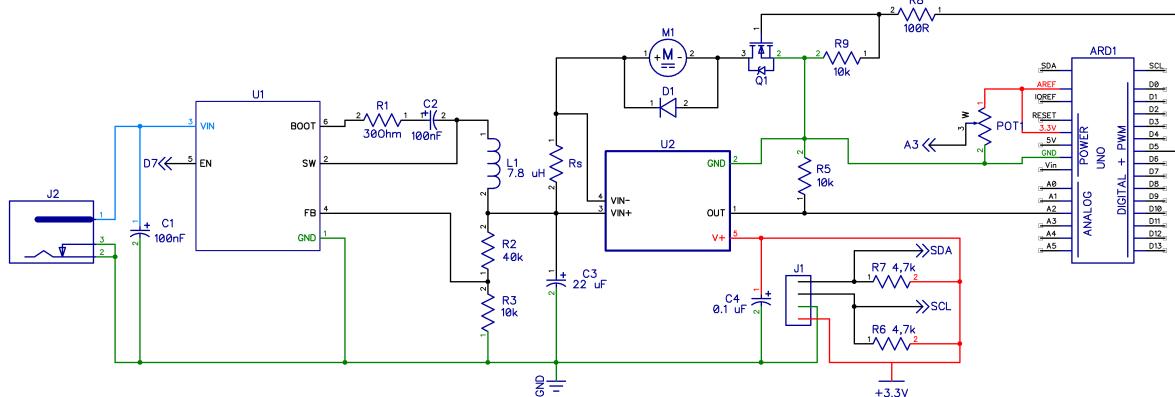
Všetky schémy zapojenia boli tvorené v bezplatnej verzii programu DipTrace. DipTrace slúži ako prostredie na tvorbu elektrotechnických schém a dosiek plošných spojov. Program v sebe zahŕňa aj časť pre tvorbu jednotlivých komponentov, pokiaľ sa tieto už nenachádzajú v niektornej z knižníc programu.

Nie všetky komponenty potrebné na tvorbu schémy zapojenia boli zahrnuté v knižniciach DipTracu, avšak tieto komponenty sú dostupné na stránkach výrobcov, odkiaľ sa dajú stiahnuť a následne použiť v schéme[26][27][23]. Do programu bola taktiež vložená knižnica AutomationShieldu ktorá má v sebe najčastejšie používané komponenty. Pri tvorbe schémy zapojenia sa najskôr všetky potrebné komponenty umiestnia na štvorčkovú plochu a približne sa určí ich poloha. Jednotlivé komponenty majú podobu elektrotechnických značiek a každý komponent má ku sebe priradené reálne vlastnosti daného dielu (veľkosť, zapojenie, dĺžka pinov a iné).

Polohu volíme takú, aby schéma bola čo najprehľadnejšia a komponenty ktoré sú medzi sebou prepojené, boli čo najblížšie pri sebe. Akonáhle máme všetky komponenty uložené začneme s ich postupným prepájaním. Pri zapájaní jednotlivých komponentov sa riadime katalógovými listami jednotlivých komponentov, v ktorých býva zväčša aj návrh ich zapojenia.

DipTrace umožňuje rozdielne zafarbovanie jednotlivých elektrických spojení, rozličnými farbami a názvami obr.1.8. Tento fakt nám veľmi uľahčuje na prvý pohľad rozoznať napríklad elektrické spojenia zeme- 0V zelená, fázové spojenia- 3,3V červená. Na schéme zapojenia sú použité nasledujúce komponenty:

- R- Rezistor
- U- Mikročip
- M- Motor
- C- Kapacitor
- L- Cievka
- D- Dióda
- J- Konektor



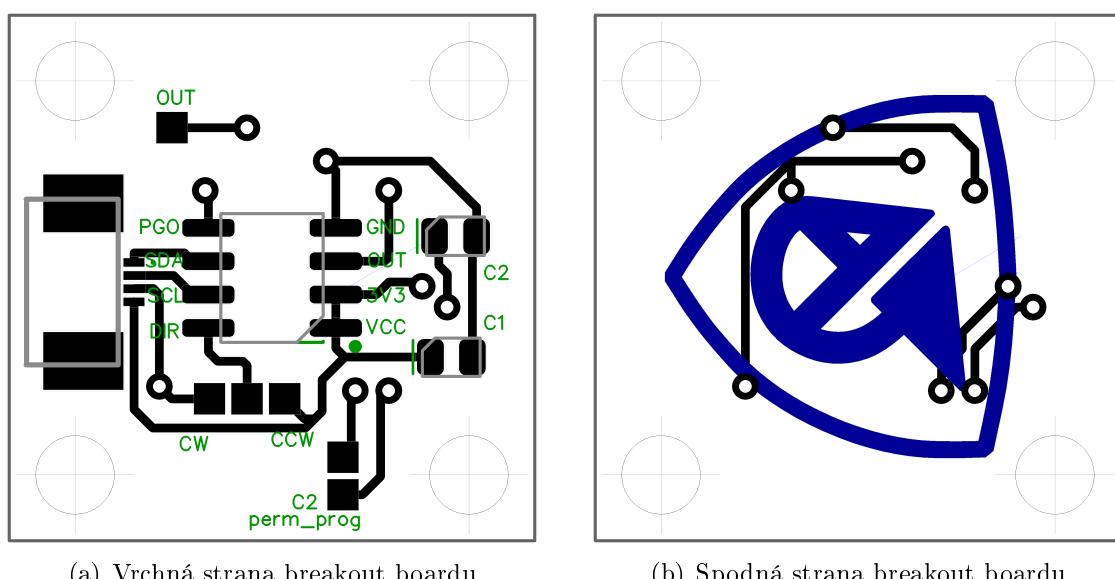
Obr. 1.8: Schéma zapojenia AeroShieldu.

1.1.3 Doska plošných spojov

Po návrhu a kontrole schém zapojenia sa schémy ďalej spracovávajú do podoby dosky plošných spojov. Schémy exportujeme do programu DipTrace PCB v ktorom máme následne niekoľko možností postupu. Jednotlivé komponenty sa nám už zobrazujú v reálnej podobe, takže vidíme ich veľkosť a rozmiestnenie konektorov na spájkovanie. Dosky plošných spojov majú niekoľko výhod, ale aj negatív oproti ponúkaným alternatívam[28].

Výhodou je fakt, že vodivé spojenia medzi jednotlivými súčasťami sú narozené od typických káblových spojení, realizované vrstvou medi, ktorá je ukrytá pod ochrannými vrstvami dosky. Ďalšou z výhod dosiek plošných spojov je skutočnosť, že sú odolné a kompaktné[29]. Tým že vodivé cesty môžu mať veľmi malé rozmer, ovplyvňujúcim faktorom veľkosti dosky plošných spojov je samotná veľkosť použitých komponentov.

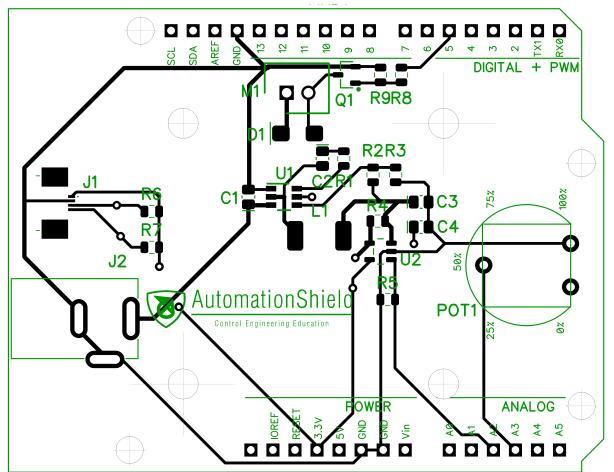
Hotová schéma zapojenia je prenesená do programu DipTrace PCB. Program ponúka možnosť automatického alebo manuálneho rozmiestnenia komponentov.



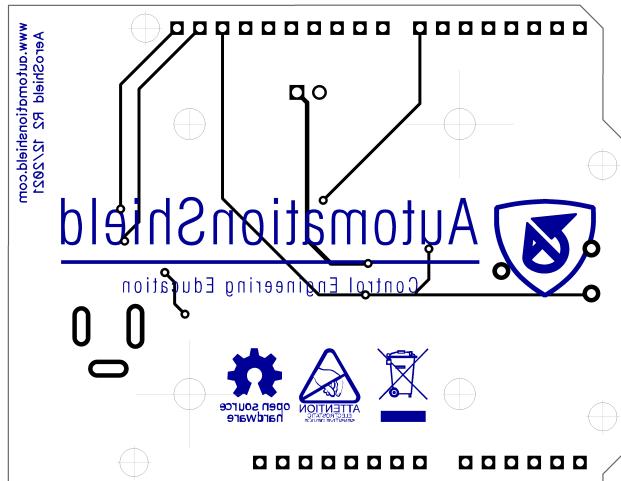
Obr. 1.9: Vedľajšia doska AeroShieldu- breakout board.

Po zvolení optimálneho rozmiestnenia komponentov treba jednotlivé piny poprepačať vodivými cestami, ktoré nám nahradzajú funkciu káblov. Máme možnosť zvoliť automatické rozmiestnenie ciest alebo ich manuálnu tvorbu. Ako je vidieť na obr. 1.10.a, nie všetky cesty majú rovnakú šírku. Dôvodom je fakt že niektorými cestami tečie vyšší prúd. V zásade sa používa pravidlo, čím vyšší prúd preteká vodičom, tým väčšiu plochu prierezu by mal mať. Prúdy pretekajúci vodičom tento vodič zahrieva. Pokial į je toto zahrievanie nadmerné, môže dôjsť k poškodeniu vodiča.

Tvorba elektrických ciest má niekoľko pravidiel. Najdôležitejšie z nich je, že cesty spájajúce rozdielne vodiče sa nemôžu križovať. Pokiaľ by k takému kríženiu došlo, jednotlivé cesty sa vzájomne vyskratujú. Z toho dôvodu treba niekedy cestu priviesť na druhú stranu dosky plošných spojov kde v jej pokračovaní neprekáža iná cesta. Na tento účel sa používajú vodivé diery "via", spájajúce obe strany dosky. Druhá verzia dosky AeroShieldu je na obr.1.10.



(a)



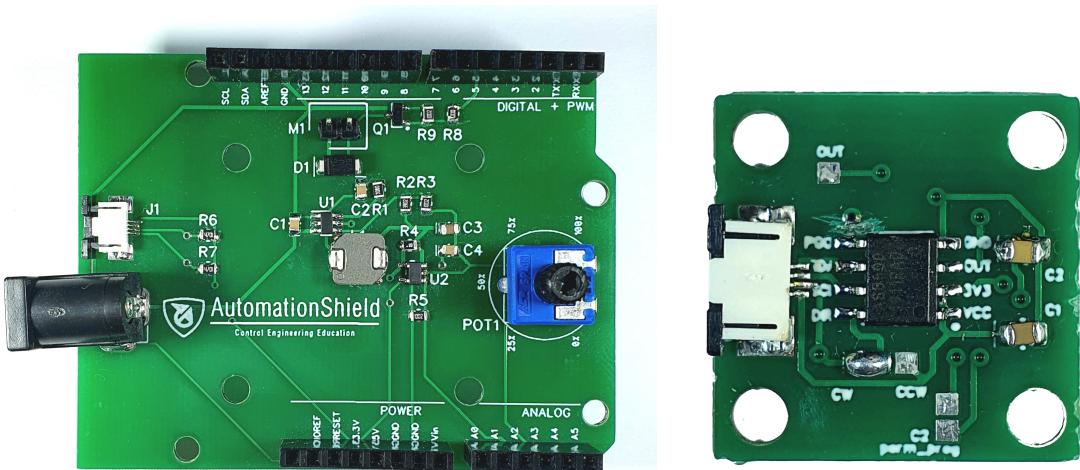
(b)

Obr. 1.10: (a) Vrchná strana AeroShieldu. (b) Spodná strana AeroShieldu.

Po finálnej kontrole zapojenia komponentov na doske plošných spojov, môžeme tieto dosky uložiť do formátu gerber. Súbory typu gerber v sebe ukladajú presné zloženie finálnej dosky plošných spojov a to po jej jednotlivých vrstvách. Zkonvertované súbory zasielame výrobcovi PCB dosiek kde si môžeme zvoliť parametre dosky ako jej farbu, typy spájkovacích doštičiek a iné. Podobu finálnej dosky AeroShieldu môžeme vidieť na obr.1.11.a a dosky breakout boardu na obr.1.11.b.

1.1.4 Model držiaku kyvadla

Telo kyvadla ako aj všetky konektory spájajúce jeho jednotlivé časti, boli vytvorené v 3D modelovacom softvéri CATIA obr.1.12. Zámerom bolo vytvoriť pevný a zároveň estetický držiak, ktorý sa dá následne vytlačiť na 3D tlačiarni. Telo kyvadla stojí na štyroch nožičkách, ktoré sú priskrutkované k hlavnej doske plošných spojov. Stred kyvadla je dutý, a sú nim vedené káble napájania motorčeka. V hornej časti držiaku sa nachádzajú diery na priskrutkovanie breakout boardu.



(a) Hlavná doska AeroShieldu

(b) Vedľajšia doska AeroShieldu

Obr. 1.11: Dosky plošných spojov AeroShieldu.



Obr. 1.12: Model kyvadla, tvorený v programe CATIA.

Názov	Popis	Ks.	Cena v €	Spolu
Kapacitor	SMD, sot23	6	0,08	0,48
Dióda	1N400IG	1	0,1	0,1
FFC konektor	FFC 4pin	2	0,2	0,4
Cievka	IND1210	1	0,2	0,2
Konektor DC motora	JST-XH 2,54	1	0,4	0,4
Motor	Howellp 7x20mm Motor	1	2,1	2,1
Potenciometer	CA14	1	0,45	0,45
Mosfet	pmv45en2	1	0,04	0,04
Rezistor	SMD, sot23	9	0,08	0,72
Buck converter	TPS56339	1	2,78	2,78
Shunt monitor	INA169/NA	1	0,98	0,98
Hall senzor	AS5600	1	1,48	1,48
3D komponenty	model kyvadla a spojovacie prvky	4	2,2	2,2
Gulôčkové ložiská	BB-694-B180-30-ES IGUS	2	2,75	5,5
Prepájacie káble FFC	akékoľvek 4 pin, dĺžka min 15cm	1	0,52	0,52
Prepajací kábel motor	akékoľvek, dĺžka min 35cm	1	0,3	0,3
Šróby	4x M3x40 4x M4x15	8	0,25	2
Karbónové trubičky	1x kruhový prierez 10cm, 1x štvorcový prierez 10cm	2	1,9	3,8
PCB shield	Výroba JLCPCB	1	0,35	0,35
PCB brakout	Výroba JLCPCB	1	0,35	0,35
Matice	M4	4	0,2	0,8
Spolu				25,95€

Tabuľka 1.1: Cenová kalkulácia AeroShieldu.

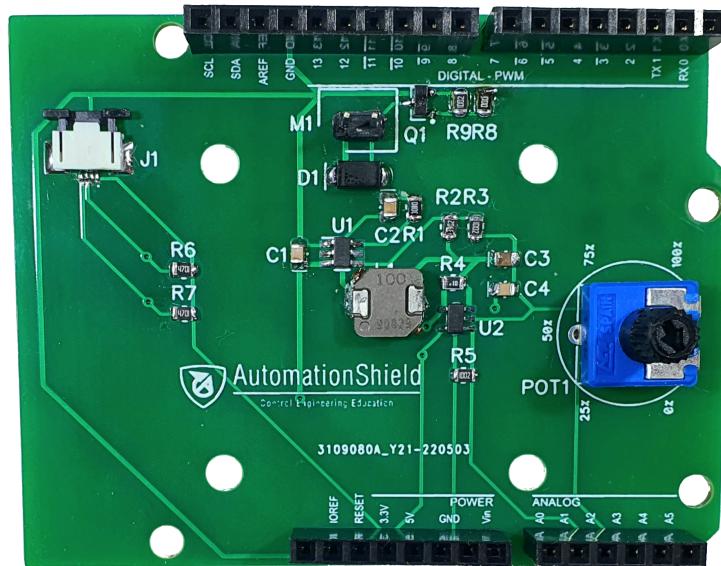
1.1.5 Cenová kalkulácia AeroShieldu

Hlavnou podmienku pri tvorbe AeroShieldu, bola jeho funkciunalita, no zároveň nízka cena. Za účelom predstavy cenovej relácie jedného kusu AeroShieldu, bola zo-stavená tabuľka1.1 s použitými komponentami, ich počtom a reálnou kúpnou cenou v eurách(spolu s DPH). Pri komponentoch ako sú rezistory a kapacitory, bola cena určená ako priemerná hodnota týchto komponentov pri kúpe viac ako 100kusov, keďže pri kúpe zopár kusov(1-20) je ich cena rádovo vyššia, ako cena pri nákupoch viacero kusov.

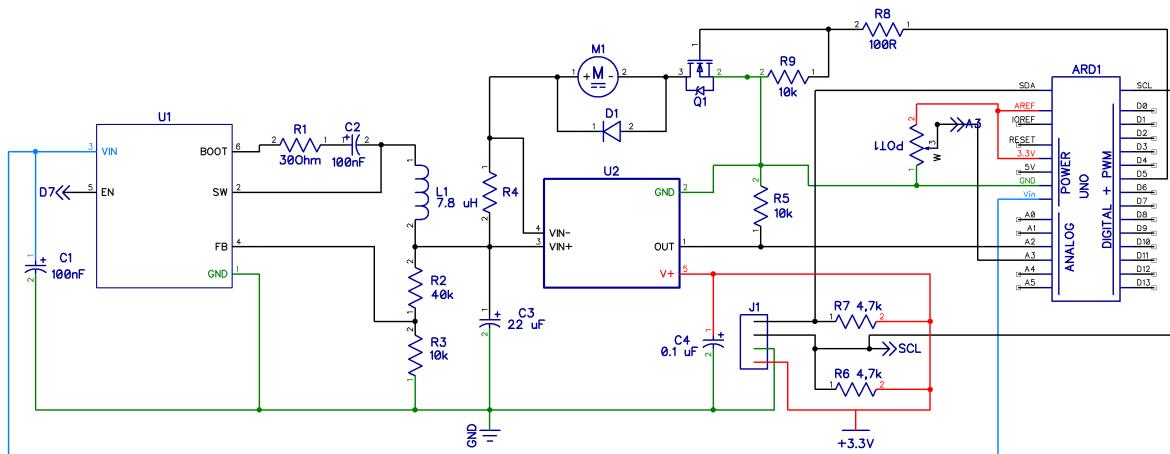
1.1.6 Tretia verzia AeroShieldu

V tretej verzii AeroShieldu obr.1.13, bol odstránený napájací konektor a napájanie je realizované z konektora, ktorý sa nachádza na doske Arduino. Na Shield je napätie 12V privádzané pomocou vin pinu obr.1.14. Prúd odoberaný motorom nedosahuje hodnoty väčšie ako 1A a teda takéto napájanie je bezpečné a nehrozí pri ňom poškodenie hardvéru Arduina. Ďalej bolo upravené rozloženie niektorých komponentov. Jedná sa hlavne o konektor J1, spájajúci hlavnú dosku s breakout doskou, rezistory R8 a R9 a mosfet Q1. Ako posledná úprava bola otočená polarity potenciometra, ktorý bol na

druhej verzie AeroShieldu pripojený opačne, ako značil popis na doske.



Obr. 1.13: Tretia verzia AeroShieldu.



Obr. 1.14: Schéma zapojenia tretej verzie AeroShieldu.

1.2 Softvér

1.2.1 Arduino API

Vývojové prostredie pre platformy arduino sa nazýva Arduino IDE⁵ a využíva programovací jazyk C++ resp. jeho nadstavbu, s pridanými špecializovanými príkazmi a funkciami. C++ patrí medzi jeden z najviac používaných programovacích jazykov na svete. V rámci projektu AutomationShield existuje niekoľko rôznych shildov, ktoré však často využívajú rovnaké funkcie a príkazy. Z tohto dôvodu bola vytvorená knižnica "AutomationShield", ktorá v sebe zahrňa niekoľko ďalších, často využívaných knižníc a súborov. Jedná sa napríklad o funkcie potrebné pri riadení pomocou PID regulátora, vzorkovanie programu alebo mapovanie premenných. O jednotlivých funkciách v rámci knižnice "AutomationShield" si povieme viacej pri opise didaktických príkladov v časti:2.

V rámci programovania knižníc využívame objektovo orientované programovanie (OOP)[30]. OOP je výhodné z hľadiska prehľadnosti, úpravy funkcií, redukcie nepotrebného alebo duplicitného kódu a mnoho ďalšieho. Pri OOP vytvárame dátové štruktúry nazývané objekty. Objekty majú svoje vlastnosti, metódy, udalosti a pomocou kombinácie týchto vlastností vykonávajú určité naprogramované činnosti.

Knižnice väčšinou rozdeľujeme do dvoch súborov. Prvým z nich je **header** teda hlavička s koncovkou .h a druhý **source** alebo zdrojový dokument s koncovkou .cpp. Header slúži ako akýsi navádzací a sklad pre premenné a funkcie, ktoré následne komunikuje so source dokumentom v ktorom sú uložené samotné funkcie. Takéto rozdelenie súborov má za cieľ zrýchlenie komplikácie programu.

Header

Header súbor má niekoľko náležitostí ktoré obsahuje. Na začiatok deklarujeme súbor samotný. Robíme to pomocou príkazu **#define**. Avšak ak by sa takáto deklarácia nachádzala vo viacerých súboroch, a teda header súbor by sa načítal niekoľko krát, spôsobovalo by to problém pri komplikácii kódu. Z toho dôvodu používame funkciu **Include guard**, ktorá zamedzuje niekoľkonásobnému načítaniu rovnakých súborov.

Hned za definovaním knižnice AeroShield.h môžeme vkladať ďalšie knižnice, ktoré sú potrebné pre funkcie danej knižnice, a to pomocou príkazu **#include**.

Za knižnicami následne určujeme premenné, ktoré môžu mať priradené fyzické čísla pinov na Arduine. Tieto premenné potom využívame buď na posielanie, alebo prijímanie signálov z daných pinov, ako aj na ukladanie rôznych konštánt. Príkaz **#endif** vkladáme do záveru header súboru.

```
#ifndef AEROSHIELD_H          // Pokial nie je definovana AEROSHIELD_H
#define AEROSHIELD_H           // Definuj kniznicu AEROSHIELD_H

#include "AutomationShield.h"   // Hlavná kniznica AutomationShieldu
#include <Wire.h>               // Kniznica potrebna pre komunikaciu I2C
#include <Arduino.h>            // Zakladna arduino kniznica

#define AERO_RPIN A3             // Vstup z potenciometra
```

⁵Arduino Integrated Development Environment.

```
#define VOLTAGE_SENSOR_PIN A2      // Vstup pre meranie prudu
#define AERO_UPIN 5                 // Aktuator
```

Zdrojovy kod

```
#endif                                // Koniec if podmienky
```

Zdrojový kód 1.1: Ukážka zdrojového kódu headeru.

V časti **Zdrojovy kod**, vytvárame **triedu** (class), ktorá v sebe zahŕňa funkcie a premenné, ktoré sa nazývajú **objekty** (objects). Class obsahuje podmnožinu objektov, ktoré vieme prepájať a spájať vo väčšie celky, vďaka čomu vieme dosiahnuť veľmi komplexné funkcie. Týmto funkciám a premenným vieme obmedziť prístup resp. ich dosah v rámci programu, pomocou modifikátorov prístupu (access modifiers). Tieto modifikátory delíme do štyroch skupín. Základný modifikátor je **default**, teda akýsi predvolený prístup, ktorý nadobúdajú všetky funkcie a premenné automaticky. Ďalšími modifikátorami sú **public**, teda funkcie a premenné verejne prístupné v triede aj mimo nej a modifikátor **privat**, ktorý obmedzuje prístup len pre danú triedu. Posledným modifikátorom prístupu je **protected**, čo je prístup chránený. V header súbore sa môže nachádzať jedna, alebo viacero tried, záleží to od logicky deliteľných úsekov kódu, alebo od preferencie programátora.

Rozdelenie na public a privat má zmysel hlavne v prípade ak chceme mať zadefinované premenné, pri ktorých nechceme aby sa dala externe zmeniť ich hodnota alebo typ. V prípade privat, takáto zmena nie je možná. Zmeniť takúto premennú môžeme len jej ručným prepísaním v zdrojovom súbore. V časti private deklarujeme funkcie, ktoré následne využívame v rámci triedy, alebo slúžia ako pomocné funkcie pri tvorbe komplexnejších častí kódu. V časti public sú funkcie viditeľné a schopné interagovať s inými triedami, ako aj s inými knižnicami.

Source

Kedže všetky potrebné knižnice sa už definovali a načítavali v súbore header, stačí nám už len časti source a header prepojiť. Urobíme tak pomocou príkazu **#include "AeroShield.h"**, ktorý vložíme na začiatok súboru. Ďalej v súbore deklarujeme jednotlivé funkcie, ktoré majú na začiatku zápisu, zvolený istý dátový typ. Dátové typy funkcií poznáme rôzne. Vyberáme si ich na základe potreby ako chceme aby funkcia reagovala resp. aké hodnoty by mala prenášať. Dátové typy⁶ poznáme nasledovné[31]:

Popis použitých funkcií z knižnice AutomationShield

Pri rôznych veľkostiacach a rozsahoch číselných stupní, je dobré vyjadrovať hodnoty v percentách, namiesto ich absolútnej hodnoty. Arduino ponúka funkciu **map()**, ktorá však pracuje len s dátovým typom integer. Aby sme docielili vyššiu presnosť, potrebujeme mapovať dátový typ float. Na tento účel nám slúži funkcia **mapFloat** do ktorej vstupuje veličina x, ktorej priradíme požadované hodnoty. Funkcie funguje na základe princípu lineárneho mapovania[32].

⁶všetky hodnoty sú platné pre arduino UNO, pre iné typy arduina sa hodnoty môžu lísiť

dátový typ	vlastnosti	dátový typ	vlastnosti
array	skupina premenných s priradeným indexom. Maximálna veľkosť je obmedzená veľkosťou pamäte RAM	short	16 bitové celé čísla
boolean	má buď hodnotu 0-nepravda, alebo 1-pravda	char array	spojenie viacerých dát typu char ukončené hodnotou null
byte	8 bitové čísla od 0 do 255	string-object	podobná funkcia ako object v header súbore
double	rovnaké ako float	unsigned char	8-bit znaky od 0 do 255
float	32 bitové desatinne čísla $\pm 3.4028235E+38$	unsigned int	16 bitové kladné celé čísla od 0 do $2^{16}-1$
char	8 bit ascii tabulka	unsigned long	32 bitové kladné celé čísla od 0 do $2^{32}-1$
int	16 bitové celé čísla	void	nevracia naspäť žiadne informácie
long	32 bitové celé čísla	word	16-bit číslo bez znamienka

Tabuľka 1.2: Dátové typy

```
float AutomationShieldClass::mapFloat(float x, float in_min, float in_max
, float out_min, float out_max)
{
return (x - in_min) * (out_max - out_min) / (in_max - in_min) + out_min;
}
```

Zdrojový kód 1.2: Zdrojový kód funkcie mapFloat.

Ďalšou z funkcií použitých z knižnice AutomationShield je serialPrint. Funkcia vypisuje zvolený text na sériový monitor arduina.

```
void AutomationShieldClass::serialPrint(const char *str){
#if ECHO_TO_SERIAL // Pokial je tato funkcia povolena
    Serial.print(str); // Vypis na seriový monitor
#endif // Koniec
}
```

Zdrojový kód 1.3: Zdrojový kód funkcie serialPrint.

Popis použitých funkcií z knižnice AeroShield

Kedže na AeroShielde využívame senzor hall efektu, musíme s ním v prvom rade nadviazať komunikáciu pomocou sériovej komunikácie I²C.

Táto komunikácia funguje na princípe master-slave, kedy master je nadriadený a slave je podriadené zariadenie, s ktorým master komunikuje. Master môže naraz komunikovať s viacerými zariadeniami a to na základe jedinečných adres zariadení, ktoré sa medzi sebou striedajú v komunikácii.

Protokol I²C využíva na odosielanie a prijímanie údajov dva vodiče resp. dve linky:

- sériovú dátovú linku (SDA-serial data), cez ktorú sa posielajú údaje,
- sériovú hodinovú linku (SCL-serial clock), na ktorú arduino v pravidelných intervaloch posielá impulzy.

Hodinový pin udáva tempo komunikácie a je ovládaný mastrom. Mení stav v pravidelných impulzoch z 0 (low), na 1 (high). Pri každej takejto zmene je na dátový pin poslaný jeden bit informácie. Tieto bity nájskôr obsahujú adresu zariadenia slave s ktorým chce master komunikovať, následne sa odosielajú bity príkazov. Keď sa táto informácia celá odošle, slave vykoná požiadavku a ak je to vyžadované, môže späťne mastrovi poslať údaje. Všetky tieto bity informácií sa posielajú na linke SDA[33].

Funkcia readOneByte()

Funkcia `int AeroShield::readOneByte()`, získava 1 bajt informácií zo senzoru. Túto funkciu využívame napríklad na čítanie polohy kyvadla.

```
int AeroShield::readOneByte(int in_addr)
{
    int retVal = -1;           // Zadefinovanie pomocnej premennej
    Wire.beginTransmission(_ams5600_Address); // Zaciatoč komunikacie
    Wire.write(in_addr);      // Požiadavka na zaznamenanie uhlu
    // kyvadla
    Wire.endTransmission(); // Koniec komunikacie zo strany mastra
    Wire.requestFrom(_ams5600_Address, 1); // Ziadost na odpoved
    while (Wire.available() == 0); // Cakaj pokial odpoved nepride
    retVal = Wire.read(); // Zaznamenanie odpovede
    return retVal; // Zaslanie odpovede
}
```

Zdrojový kód 1.4: Zdrojový kód funkcie readOneByte.

Ako môžeme vidieť v kóde funkcie, master nájskôr osloví zariadenie slave, pomocou jeho adresy. Pri senzore AS5600 je adresa zariadenia v hexadecimálnej podobe 0x36. Následne zašle od výrobcu predprogramovanú žiadosť, ktorá zaznamená aktuálnu polohu magnetu resp. kyvadla. Následne je zaslaná požiadavka na odpoved zo strany slave zariadenia, ktorá je zaznamenaná a odoslaná späť na miesto, z ktorého bola funkcia privolaná.

Funkcia readTwoBytes()

Funkcia word AeroShield::readTwoBytes() je podobná predošej funkcií, s tým rozdielom, že získané sú dva bajty informácií. Na konci funkcie ešte prebieha bitový posun⁶.

```
word AeroShield::readTwoBytes(int in_adr_hi, int in_adr_lo)
{
    word retVal = -1; // Zadefinovanie pomocnej
                      premennej
    /* citanie "Low" bajtu */
    Wire.beginTransmission(_ams5600_Address); // Zaciatoč komunikacie
    Wire.write(in_adr); // Poziadavka na zaznamenanie uhlu
                        kryvadla
    Wire.endTransmission(); // Koniec komunikacie zo strany mastra
    Wire.requestFrom(_ams5600_Address, 1); // Ziadost na odpoved
    while (Wire.available() == 0); // Cakaj pokial odpoved nepride

    int low = Wire.read(); // Ulozenie prveho bajtu
    /* citanie "High" bajtu */
    Wire.beginTransmission(_ams5600_Address); // Zaciatoč komunikacie
    Wire.write(in_adr); // Poziadavka na zaznamenanie uhlu
                        kryvadla
    Wire.endTransmission(); // Koniec komunikacie zo strany mastra
    Wire.requestFrom(_ams5600_Address, 1); // Ziadost na odpoved
    while (Wire.available() == 0); // Cakaj pokial odpoved nepride

    word high = Wire.read(); // Ulozenie druhého bajtu
    high = high << 8; // Posun bitov
    retVal = high | low;

    return retVal; // Zaslanie odpovede
}
```

Zdrojový kód 1.5: Zdrojový kód funkcie readTwoBytes.

Funkcia detectMagnet()

Ďalšou dôležitou funkciou, je zistiť prítomnosť magnetu na kryvadle. Túto úlohu vykonáva funkcia int AeroShield::detectMagnet(). Využívame ju vždy pri inicializácii AeroShieldu, na to aby sme zistili či nenastali problémy s magnetom, alebo so senzorom samotným. Funkcia komunikuje so senzorom, a na základe výstupu vieme určiť či bol magnet detegovaný. Funkcia nám vráti na výstupe 1- pokiaľ sa magnet nachádza pri senzore a 0- pokiaľ magnet nie je prítomný, alebo je príliš vzdialený od senzoru.

```
int AeroShield::detectMagnet()
{
    int magStatus; // Pomocna premenna
    int retVal = 0; // Pomocna premenna
    magStatus = readOneByte(_stat); // Prebieha komunikacia so
                                    senzorom
```

⁶Bitový posun je operácia vykonávaná so všetkými bitmi binárnej hodnoty, pri ktorej sa posúvajú o určený počet miest doľava alebo doprava[34].

```

    if (magStatus & 0x20)           // Pokial je podmeinka splnena
        vrat 1, pokial nie je splnena vrat 0
    retVal = 1;

    return retVal;                // Zaslanie odpovede
}

```

Zdrojový kód 1.6: Zdrojový kód funkcie detectMagnet.

Funkcia getMagnetStrength()

Pre správnosť fungovania hall senzoru je dôležité dodržať správnu vzdialenosť magnetu od senzoru. Výrobca udáva že najideálnejšia vzdialosť je 0.5-3mm, v závislosti na sile a veľkosti magnetu. Bolo by nepraktické túto vzdialenosť merať ručne, použijeme preto funkciu `int AeroShield :: getMagnetStrength ()`. Môžeme si všimnúť že táto funkcia používa rovnaký príkaz na komunikáciu so senzorom, ako aj funkcia `detectMagnet ()` a to sice `_stat`. Z toho vyplýva že `detectMagnet ()` kontroluje nielen prítomnosť magnetu, ale aj jeho správnu vzdialenosť. Pokiaľ teda dostaneme z funkcie `detectMagnet ()` ako výsledok 1, vieme že magnet je prítomný a zároveň v ideálnej vzdialnosti. Funkcia `getMagnetStrength ()` je teda iba doplňujúcou funkciou, ktorá nám určí či sa magnet nachádza príliš blízko, alebo naopak veľmi ďaleko od senzoru.

```

int AeroShield :: getMagnetStrength ()
{
    int magStatus;                  // Pomocna premenna
    int retVal = 0;                 // Pomocna premenna
    magStatus = readOneByte(_stat); // Prebieha komunikacia so
                                    // senzorom

    if (detectMagnet () == 1)       // Pokial je splnena podmienka
        detectMagnet()
    {
        retVal = 2; // Vrat 2, magnet je v idelnej vzdialnosti
        if (magStatus & 0x10)
            retVal = 1; // Vrat 1, magnet je v prilis daleko
        else if (magStatus & 0x08)
            retVal = 3; // Vrat 3, magnet je v prilis blizko
    }

    return retVal;                // Zaslanie odpovede
}

```

Zdrojový kód 1.7: Zdrojový kód funkcie getMagnetStrength.

Funkcia getRawAngle()

Táto funkcia slúži na čítanie uhlu kyvadla. Výstupom tejto funkcie, je číslo s rozsahom 12bitov, teda číslo od 0 do 4096, ktoré udáva momentálnu polohu kyvadla.

```

word AeroShield :: getRawAngle ()
{

```

```

        return readTwoBytes(_raw_ang_hi, _raw_ang_lo); // Prebieha
        komunikacia so senzorom, ktorý rovno vrati výsledok pomocou
        príkazu return
    }
}

```

Zdrojový kód 1.8: Zdrojový kód funkcie getRawAngle.

Funkcia begin()

Prvou z funkcií mimo komunikácie s hall senzorom je `float AeroShield :: begin (bool isDetected)`, do ktorej vstupuje výsledok z funkcie `detectMagnet()` ako premenná `isDetected`. Funkcia `begin()` nastaví pin potrebný na ovládanie akčného člena, pomocou príkazu `pinMode`, ako výstup (OUTPUT). Zároveň inicializuje sériovú komunikáciu I²C. Príkaz na započatie komunikácie I²C sa pri rôznych typoch dosiek, resp. architektúr mikroradiča Arduino líši. Použijeme preto podmienku `#ifdef` za ktorou nasleduje typ architektúry daného mikroradiča a príslušný príkaz pre začiatok sériovej komunikácie I²C. V prípade Arduino UNO, je to príkaz `Wire.begin()`.

Zároveň je vo funkcií `begin()`, pomocou `if` podmienky, kontrolovaná premenná `isDetected`. Pokiaľ bol magnet detegovaný, vypíše na sériový port správu "Magnet detected" a `while`⁸ cyklus, sa pomocou príkazu `break` ukončí. Pokiaľ magnet detegovaný neboli, vypíše "Can not detect magnet", no `while` cyklus pokračuje.

```

float AeroClass :: begin(void){
    bool isDetected = AeroShield.detectMagnet(); // Detekcia magnetu
    pinMode(AERO_UPIN, OUTPUT); // Pin aktuátora

#ifdef ARDUINO_ARCH_AVR // Pre dosky s architekturov AVR
    Wire.begin(); // Pouzi objekt Wire
#elif ARDUINO_ARCH_SAM // Pre dosky s architekturov SAM
    Wire1.begin(); // Pouzi objekt Wire1
#elif ARDUINO_ARCH_SAMD // Pre dosky s architekturov SAMD
    Wire.begin(); // Pouzi objekt Wire
#endif

    if(isDetected == 0){ // Pokial magnet nie je
        detegovany
            while(1){ // While podmienka
                if(isDetected == 1){ // Pokial sa magnet deteguje
                    AutomationShield.serialPrint("Magnet detected \n");
                    break; // Koniec while podmienky
                }
                else{ // Pokial magnet nie je detegovany
                    AutomationShield.serialPrint("Can not detect magnet \n");
                }
            }
    }
}

```

Zdrojový kód 1.9: Zdrojový kód funkcie begin.

⁸Cyklus `while` sa bude opakovať nepretržite, pokiaľ sa výraz vnútri zátvoriek () nestane nepravdivým.

Funkcia calibration()

Funkcia calibration() slúži na prepočet a zaznamenanie nulovej polohy kyvadla, teda takej kedy výkon motora=0%. V ideálnom prípade by sa kyvadlo vždy po vypnutí motora vrátilo do rovnakej východzej polohy. Avšak kyvadlo je prepojené s motorom a shieldom pomocou kálov, ktoré tvoria odpor a teda kyvadlo sa vždy zastaví v inej nulovej polohe.

Do funkcie vstupuje hodnota aktuálneho uhlu kyvadla, z funkcie getRawAngle() ako premenná RawAngle. Funkcia na začiatok vypíše text "Calibration running..." a motorček zapneme na štvrt sekundy na výkon 20%. Kyvadlo sa začne kývať a počkáme štyri sekundy, pokiaľ sa ustáli. Keď je kyvadlo ustálené zaznamenáme jeho hodnotu do premennej startangle. Následne prebieha for cyklus ktorý zvukovo informuje o dokončenej kalibrácii pomocou troch pípnutí.

```
float AeroShield::calibration(word RawAngle) {
    AutomationShield.serialPrint("Calibration running...\n");
    startangle=0; // Vynulovanie premennej
    analogWrite(AERO_UPIN,50); // Spustenie motora na vykon 20%
    delay(250); // Cakaj 0.25s
    analogWrite(AERO_UPIN,0); // Vypnutie motora
    delay(4000); // Cakaj 4s

    startangle = RawAngle; // Uloz hodnotu nuloveho uhla
    analogWrite(AERO_UPIN,0); // Poistne vypnutie motora
    for(int i=0;i<3;i++){ // Funkcia na zvukovu signalizaciu
        analogWrite(AERO_UPIN,1); // Zapnutie motora
        delay(200); // Cakaj
        analogWrite(AERO_UPIN,0); // Vypnutie motora
        delay(200); // Cakaj
    }

    AutomationShield.serialPrint("Calibration done");
    return startangle; // Vrat hodnotu
}
```

Zdrojový kód 1.10: Zdrojový kód funkcie calibration.

Funkcia convertRawAngleToDegrees()

Ako sme už spomínali v sekcií 1.1.1, zaznamenaná hodnota uhlu kyvadla je v rozmedzí od 0 do 4096 a tieto hodnoty sú priamo úmerné zo stupňami od 0° do 360°. 1° predstavuje hodnotu približne 11.77 vo formáte raw. Funkcia teda prenásobí raw hodnotu číslom $\frac{360}{4096} = 0.087$ °. Výsledkom rovnice je prepočítaný uhol v stupňoch.

```
float AeroShield::convertRawAngleToDegrees(word newAngle) {
    float ang = newAngle * 0.087; // 360/4096=0.087 x rawHodnota
    return ang; // Vrat hodnotu
}
```

Zdrojový kód 1.11: Zdrojový kód funkcie convertRawAngleToDegrees.

Funkcia referenceRead()

Funkcia referenceRead() slúži na čítanie hodnoty z potenciometra, ktorý sa nachádza na shielde, a jeho následne premapovanie do percentuálnej podoby. Potenciometer využívame na manuálne ovládanie AeroShieldu a v ďalších funkciách, sa využíva hlavne jeho percentuálna hodnota. Vrátená hodnota je typu float, v škále od 0.0% do 100.0%.

```
float AeroShield::referenceRead(void) {
    referencePercent = AutomationShield.mapFloat(analogRead(
        AERO_RPIN), 0.0, 1024.0, 0.0, 100.0);
    // Premapovanie originalnej hodnoty 0.0-1023 na
    // percentualny rozsah 0.0-100.0
    return referencePercent;           // Vrat percentualnu
                                      // hodnotu
}
```

Zdrojový kód 1.12: Zdrojový kód funkcie referenceRead.

Funkcia actuatorWrite()

Na ovládanie motora resp. jeho otáčok, používame funkciu actuatorWrite(). Do funkcie vstupuje percentuálna hodnota výkonu motora. Táto hodnota je premapovaná na PWM signál, ktorý využívame na ovládanie motora. Tento signál následne vstupuje do ochranej funkcie constrainFloat(), ktorá zabezpečí aby sa hodnota PWM signálu mohla pohybovať len v rozmedzí od 0.0 do 255.0.

```
void AeroShield::actuatorWrite(float PotPercent) {
    float mappedValue = AutomationShield.mapFloat(PotPercent,
        0.0, 100.0, 0.0, 255.0);
    // Vstupna percentualna hodnota 0.0-100.0 premapovana na
    // hodnoty 0.0-255.0
    mappedValue = AutomationShield.constrainFloat(mappedValue,
        0.0, 255.0);
    // Bezpecnostna funkcia obmedzenia premapovanej hodnoty
    analogWrite(AERO_UPIN, (int)mappedValue); // Zapisanie
                                                // hodnoty na pin
}
```

Zdrojový kód 1.13: Zdrojový kód funkcie actuatorWrite.

Funkcia currentMeasure()

Poslednou funkciou AeroShieldu je currentMeasure(). Funkcia slúži na zaznamenávanie prúdu, ktorý odoberá motor kyvadla. Fungovanie snímača je opísané v sekcií 1.1.1. Funkcia slúži na prepočítanie zaznamenanej hodnoty napäťia na prúd. Pre presnejšie výsledky merania, využívame priemernú hodnotu prúdu, ktorú získavame pomocou for cyklu.

Nepresnosť vzniká v dôsledku ovládania motora PWM signálom. Tým že je motor prerušovane vypínaný a zapínaný, kolísa aj veľkosť odoberaného prúdu.

Výsledná hodnota prúdu prechádza úpravami, pomocou dvoch korekčných premeníných, ktoré sme získali vďaka meraniam prúdu pomocou multimetra, a následným porovnaním týchto hodnôt oproti hodnotám zo senzora.

Prevod z napäťa na prúd robíme podľa pokynov uvedených v zdrojovom dokumente senzoru. Pri tomto prevode využívame hodnotu shunt rezistora RS , ako aj hodnotu rezistora RL .

```

float AeroShield::currentMeasure(void){
    for(int i=0 ; i<repeatTimes ; i++){           // For cyklus
        voltageValue= analogRead(VOLTAGE_SENSOR_PIN);
        // Citanie hodnoty zo senzoru INA169
        voltageValue= (voltageValue * voltageReference) / 1024;
        // Mapovanie hodnoty zo senzoru, na realnu hodnotu napatia(
            // referencne napatie je 5V)
        current= current + correction1-(voltageValue / (10 * ShuntRes));
        // Vzorec na vypocet prudu
        // Is = (Vout x 1k) / (RS x RL)
    }

    float currentMean= current / repeatTimes;
    // Vypocet priemernej hodnoty
    currentMean= currentMean-correction2;           // Korekcia
    if(currentMean < 0.000) currentMean= 0.0;
    // Korekcia nulovej hodnoty
    current= 0;          // Vynulovanie pomocnych premennych
    voltageValue= 0;      // Vynulovanie pomocnych premennych
    return currentMean; // Vrat hodnotu prudu v amperoch
}

```

Zdrojový kód 1.14: Zdrojový kód funkcie currentMeasure.

1.2.2 MATLAB

V tomto programe, rovnako ako pri Arduino IDE, vieme príkazy a funkcie zapisovať do knižnice, odkiaľ si potom jednotlivé položky voláme do hlavného kódu. Z toho dôvodu bola zostavená knižnica `AeroShield.m`. Knižnice v MATLABE môžu mať podobu súborov `Header` a `Source`, alebo funkcie zapíšeme len do jedného súboru s koncovkou `.m`.

V našom prípade sa jedná o jeden súbor na ktorého začiatku definujeme triedu súboru, pomocou príkazu `classdef AeroShield < handle ... end`. Za príkazom `classdef` nasleduje názov našej triedy `AeroShield` a porovnávací symbol `<`, za ktorým nasleduje "super trieda" `handle`. Super, alebo nad-trieda `handle` je abstraktná trieda, ktorej inštancia sa nedá vytvoriť priamo. Táto trieda sa využíva na odvodenie iných tried, ktoré sú už konkrétnymi triedami, ktorých inštancie sú objektmi `handle`.

Každá trieda môže následne obsahovať jeden alebo viacero triednych blokov:

- `Properties(atribúty) ...end-` Vymedzuje blok kódu, ktorý definuje vlastnosti s rovnakými nastaveniami atribútov.
- `Methods(atribúty) ...end-` Má rovnaký názov ako trieda v ktorej sa nachádza a vracia inicializovaný objekt triedy.
- `Events(atribúty) ...end-` Vymedzuje blok kódu, v ktorom nastane istá preddefinovaná udalosť, napr. zmena stavu.

- Enumeration(atribúty)...end- Vytvorí zoznam hodnôt/premenných.

V AeroShield knižnici máme definované dva triedne bloky typu properties. V prém bloku sa nachádzajú objekty pre dosku arduino a hall senzor. V druhom bloku definujeme všetky premenné ktoré budeme ďalej v kóde využívať. Jedná sa hlavne o názvy a čísla pinov, ktoré používame na čítanie alebo zapisovanie hodnôt, ako aj o premenné, na výpočet prúdu odoberaného motorom.

```
classdef AeroShield < handle

properties
    arduino; % objekt dosky arduino
    as5600; % objekt hall senzoru
end

properties(Constant)
    AERO_UPIN = 'D5'; % pin aktuatora
    AERO_RPIN = 'A3'; % pin potenciometra
    VOLTAGE_SENSOR_PIN = 'A2'; % pin na meranie prudu

    voltageReference = 5.0; % referencne hodnota napatia
    ShuntRes = 0.1; % hodnota shunt rezitora v ohmoch
    correction1 = 4.1220; % korekcia
    correction2 = 0.33; % korekcia
    repeatTimes = 50; % pocet cyklov pre vypocet priemernej
    hodnoty prudu
end
```

Zdrojový kód 1.15: Knižnica AeroShield.m properties.

Nasleduje triedy blok **methods**, v ktorom sú všetky funkcie, ktoré budeme volať do hlavnej časti kódu. Keďže logika funkcií je rovnaká ako pri súbore **Source**, nebudem si funkcie vysvetľovať po jednom. Dôležité je poznamenať, že pokial nám do funkcie nejaká premenná vstupuje, zapisujeme ju v zátvorke za príkazom funkcie **function** nazovFunkcie(AeroShieldObject, vstupnaPremenna). Naopak, pokial z funkcie nejaká premenná vychádza, túto premennú deklarujeme pred názvom funkcie **function** vystupnaPremenna = nazovFunkcie(AeroShieldObject). Pre pochopenie si ešte ukážeme časť kódu **function begin(AeroShieldObject)**, v ktorej prebieha tvorba objektov pre arduino a hall senzor. Proces odvolávania sa na objekt **arduino**, funguje cez názov triedy **AeroShield**, v ktorom sa odvolávame na **Object**. Príkaz **arduino()** slúži na vyhľadanie dosky arduino, pripojenej do počítača a nastavenie parametrov potrebnych na komunikáciu. Rovnako sa odvolávame aj na objekt **as5600**, ktorý vytvoríme pomocou príkazu **device()**, v ktorom zadávame objekt dosky arduino, a adresu zariadenia I²C. Poslednou funkciou je vypísanie správy a konfiguráciu pinu **AERO_UPIN** ako výstup.

```
function begin(AeroShieldObject) % funkcia inicializacie
    AeroShieldObject.arduino = arduino(); % tvorba arduino objektu
    AeroShieldObject.as5600 = device(AeroShieldObject.arduino, 'I2CAddress', 0x36); % tvorba objektu sonzoru
    configurePin(AeroShieldObject.arduino, AeroShieldObject.AERO_UPIN, 'DigitalOutput') % konfiguracia pinu ako vystup
    disp('AeroShield initialized.') % vypis spravu
```

end

Zdrojový kód 1.16: Knižnica AeroShield.m properties.

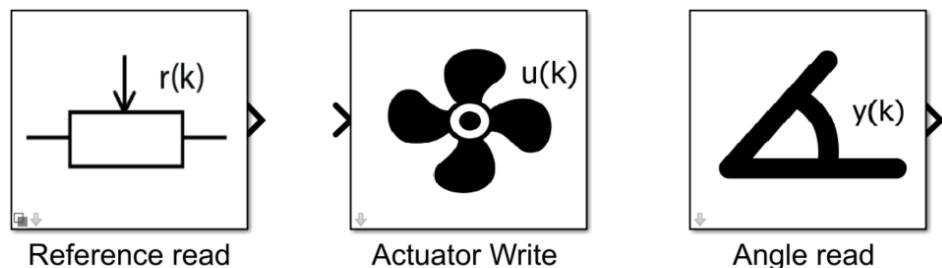
Zostávajúce funkcie, ako aj celý zdrojový kód knižnice `AeroShield.m` sa nachádza v prílohe na strane xi.

1.2.3 Simulink

Simulink, narozenie od MATLABU alebo Arduino IDE, využíva grafické programátoriské prostredie, ktoré je založené na prepájaní funkčných blokov, vyberaných z knižnice. Pre prácu s Arduinom, v rámci Simulinku, existuje knižnica *Simulink Support Package for Arduino Hardware*, ako aj knižnica AeroLibrary obr.1.15, ktorú využívame priamo pre funkcie AeroShieldu. V knižnici AeroLibrary sa nachádzajú tieto bloky:

- Reference read- čítanie referenčnej hodnoty,
- Actuator write- ovládanie akčného členu,
- Angle read- meranie výstupu,
- AeroShield- súhrn blokov na zapisovanie akčného zásahu ako aj meranie reguloanej veličiny.

Jednotlivé bloky majú svoju vlastnú vnútornú štruktúru a nastavenia, ktoré sa dajú meniť priamo v bloku, alebo pomocou masky bloku. Maska je akési grafické okno, ktoré sa zobrazí po kliknutí na blok. Toto okno zobrazuje informácie o funkcionalite bloku, ako aj prvky, pomocou ktorých sa dá nastaviť vnútorná štruktúra bloku. Pre lepšiu orientáciu medzi blokmi, má každý priradený vlastný ilustračný obrázok. Bližšie si o tvorbe masky povieme v časti 1.2.3.



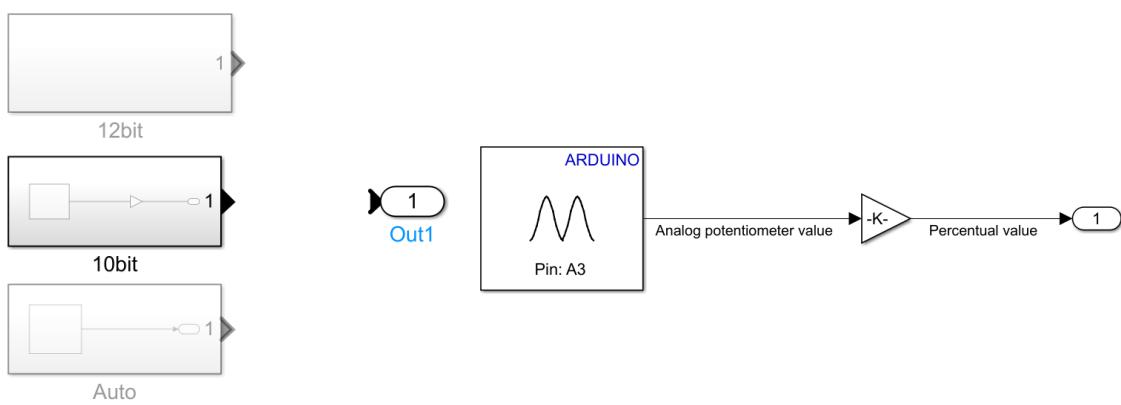
Obr. 1.15: Knižnica AeroLibrary.

Blok **Actuator Write** má najjednoduchšiu vnútornú štruktúru. Skladá sa z funkcie **Constrain**, ktorá prepočíta percentuálnu hodnotu vstupu na výstupný 8-bitový PWM signál. Tento signál posielá na Shield blok PWM, z knižnice *Simulink Support Package for Arduino Hardware*.

Reference read

V časti **Reference read** máme na výber možnosť manuálnej, alebo automatickej trajektórie. Používateľ si z týchto možností vyberie pomocou tlačidiel v maske bloku. Vnútorná štruktúra sa ďalej skladá z troch podsystémov, z ktorých dva, pre manuálnu trajektóriu obr.1.16, sú takmer totožné. Rozdiel medzi nimi je taký, že pri použití niektorých druhov Arduina¹¹ je pri použití bloku **Analog input**, výstup v tvare 12-bitového čísla, narozenie od 10-bitového, ako je tomu pri ostatných podporovaných modeloch Arduina. Načítaná hodnota je ďalej upravovaná do percentuálnej podoby, pomocou prenásobenia konštantou v bloku **gain**.

Podsystém pre automatickú trajektóriu obr.1.17 má viacero možností nastavenia. Ponúka možnosť zmeny pola referenčnej trajektórie, dĺžku trvania jednotlivých sekcií ako aj zmenu vzorkovacieho času. Viac informácií o bloku **Reference read** nájdeme v časti 1.2.3 na strane 34.



(a) Tri podsystémy bloku **Reference read**.

(b) Podsystém pre 10-bitovú logiku.

Obr. 1.16: Reference read- Simulink.

V MATLAB **function** bloku, ktorý sa nachádza v podsystéme pre automatickú trajektóriu, nájdeme kód 1.17. Funguje na veľmi podobnom princípe ako v príklade 1.2.2. Tento funkčný blok má 4 vstupy, z ktorých 3 zadávame v maske bloku a 2 výstupy. Funkcia kontroluje čas, ako dlho sú spustené jednotlivé úseky automatickej trajektórie. Robí tak pomocou bloku **Clock**, ktorý udáva aktuálny čas od spustenia simulácie. Pokiaľ je úsek trajektórie spustený dlhšie ako zadávame v maske bloku, funkcia automaticky prejde na ďalší úsek. Pokiaľ bola automatická trajektória ukončená, funkcia zastaví prebiehajúcu simuláciu.

```
function [r, stopFlag] = fcn(clock, refVector, sectionLength, Ts)
persistent counter;
pauseTime = 2;

if isempty(counter)
counter = 1;
end
if clock > pauseTime
```

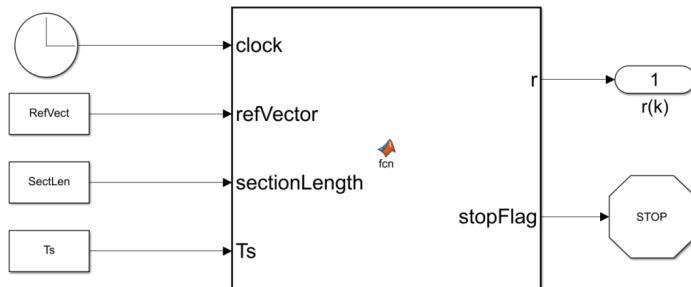
¹¹Due, MKR1000, MKR1010, MKRZero, Nano 33IoT.

```

RefLength = length( refVector );
SectionDuration = sectionLength * Ts;
if (clock - pauseTime) <= (SectionDuration * counter)
r = refVector(counter);
stopFlag = 0;
else
counter = counter + 1;
if counter <= RefLength
r = refVector(counter);
stopFlag = 0;
else
r = refVector(counter-1);
stopFlag = 1;
end
end
else
r = refVector(1);
stopFlag = 0;
end
end

```

Zdrojový kód 1.17: MATLAB function blok autiomatická trajektoria.

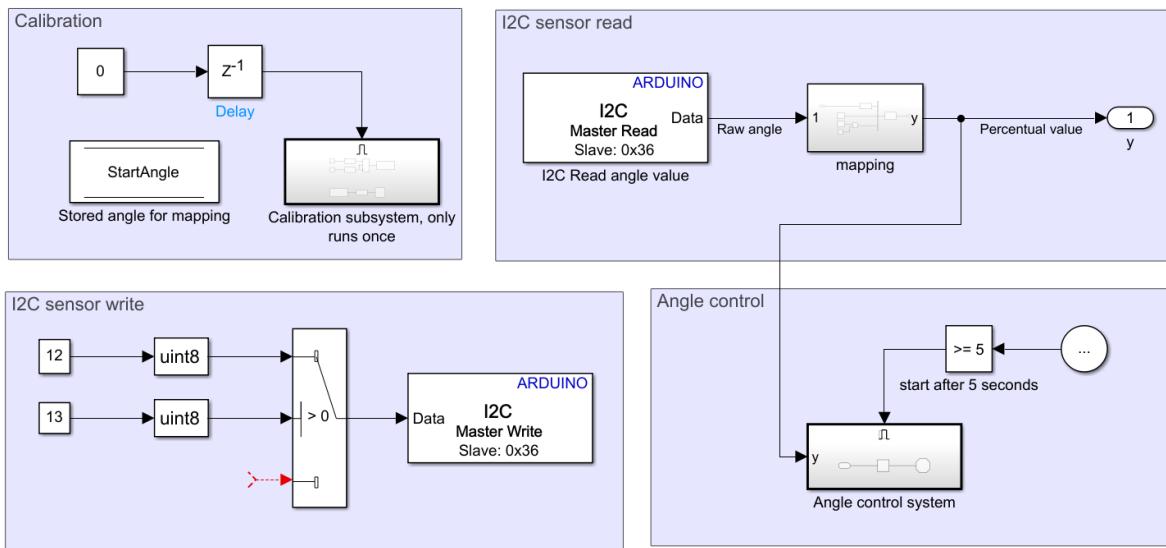


Obr. 1.17: Automatická trajektória- Simulink.

Angle read

Tento blok obr.1.18 slúži na čítanie uhlu kyvadla, ako aj pre bezpečnostnú kontrolu uhlu kyvadla. Jeho vnútorná štruktúra sa dá rozdeliť na štyri menšie celky:

- zapisovanie príkazov na senzor,
- čítanie uhlu kyvadla zo senzoru,
- kalibrácia nulového uhlu kyvadla,
- kontrola uhlu kyvadla.



Obr. 1.18: Angle read- Simulink.

I2C zápis Blok I2C Write obr.1.18(ľavý dolný roh), z knižnice *Simulink Support Package for Arduino Hardware*, zapisuje na senzor striedavo hodnoty 12 a 13, ktoré slúžia ako príkaz pre čítanie a následné odoslanie hodnoty pootočenia kyvadla. Prepínanie medzi týmito dvomi hodnotami zabezpečuje automatický prepínač (Automatic switch).

I2C čítanie Samotné čítanie uhlu zabezpečuje blok I2C Read obr.1.18(pravý horný roh), ktorého výstupom je 12-bitové číslo, predstavujúce aktuálne natočenie kyvadla. Tento výstup musíme previesť na uhlovú výchylku v rozmedzí od 0 do 360° . Tento prevod sa vykonáva v mapovacom podsystéme obr.1.19. Mapovanie funguje na rovnomnom princípe ako v príklade 2.1.2. Jednotlivé premenné vchádzajú do bloku MUX, ktorý slúži ako multiplexer¹². Signál z multiplexoru vstupuje do fcn bloku, ktorý prepája Simulink s MATLABOM. Zdrojový kód 1.18 zobrazuje funkciu vo vnútri tohto bloku.

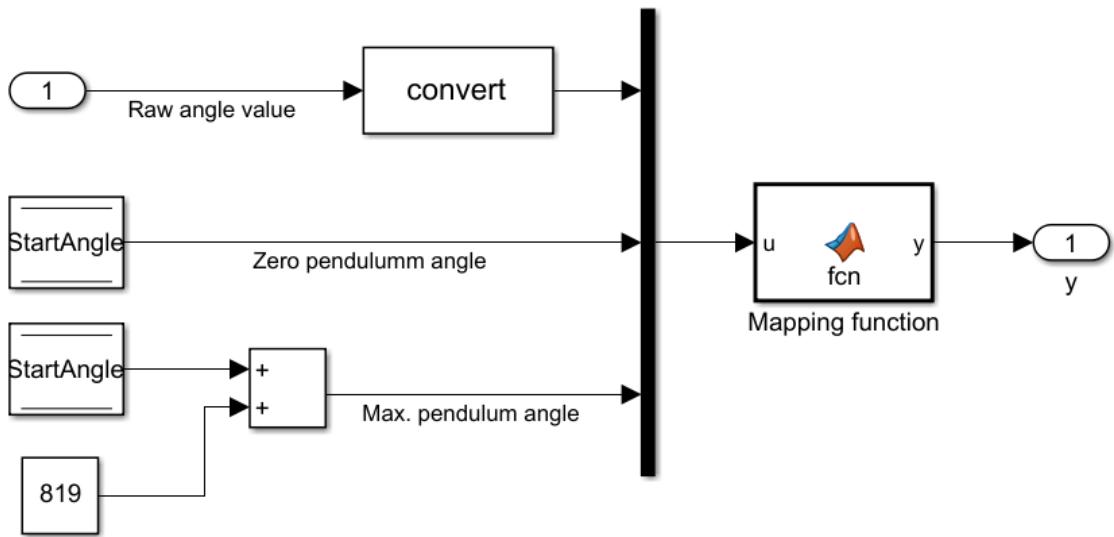
```
function y = fcn(u)
angMin=0;
angMax=72;
y = ((u(1) - u(2)) * (angMax - angMin)) / (u(3) - u(2)) + angMin
;
```

Zdrojový kód 1.18: Mapovacia funkcia vo fcn bloku.

Proces získania premennej **StartAngle**, ktorú využívame pri mapovaní, je opísaný v nasledujúcom odstavci.

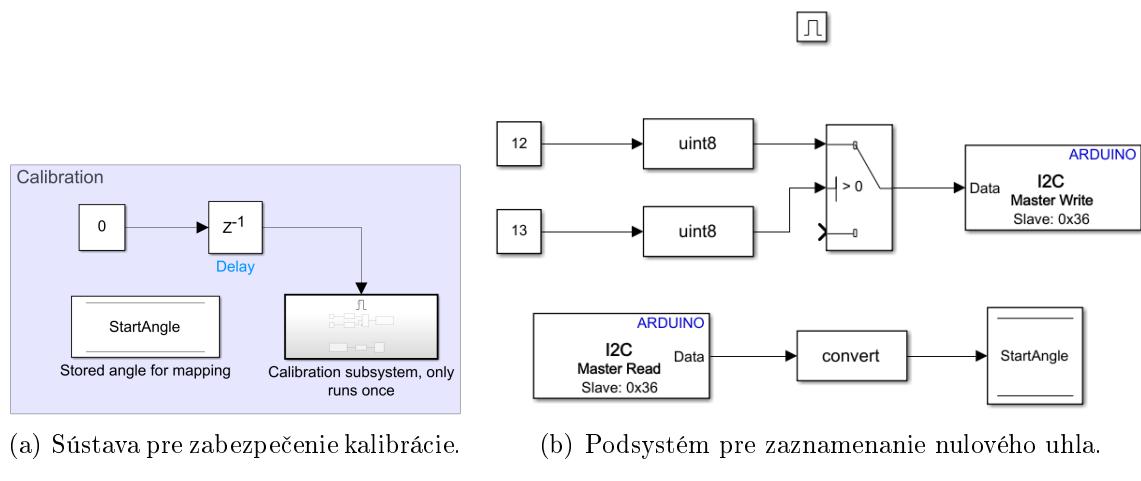
Kalibrácia Kalibrácia prebieha vždy len raz a to pri spustení simulácie. V podsystéme (Enabled Subsystem) obr.1.20.a, ktorý je spustený len ak je do neho privedený signál, sa nachádzajú funkcie pre zapisovanie a čítanie informácií zo senzoru. Uhlopis

¹²Zariadenie, ktoré vyberá medzi viacerými analógovými alebo digitálnymi vstupnými signálmi a tieto preposiela na jednu výstupnú linku.



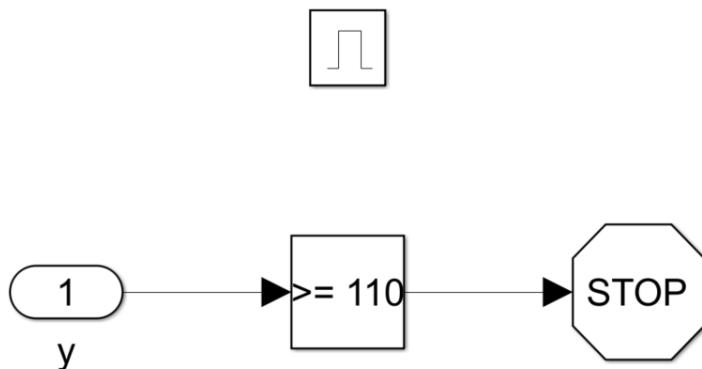
Obr. 1.19: Mapovanie uhlu kyvadla- Simulink.

ktorý je týmto podsystémom zaznamenaný pri spustený simulácie, sa uloží v bloku **Data store Write**, ako premenná **StartAngle** obr.1.20.b.



Obr. 1.20: Kalibrácia- Simulink.

Kontrola uhlu Kontrola uhlu kyvadla prebieha v podsystéme, ktorý je spustený päť sekúnd po začatí simulácie obr.1.18(pravý dolný roh). Časový posun je použitý z dôvodu občasného šumu premenných počas začiatku simulácie. Tento šum môže spôsobiť nechcené výkyvy hodnoty uhlu kyvadla, ktoré by splnili podmienku v podsystéme a tým by zastavili priebeh simulácie. V podsystéme obr.1.21, sa nachádza blok **Compare To Constant**, ktorý porovnáva hodnotu uhlu kyvadla, s daným maximálnym uhlom kyvadla. Táto hodnota je v našom prípade 110° . Pokiaľ kyvadlo dosiahne väčší uhol, ako je uhol dovolený, blok **Stop Simulation** zastaví simuláciu.



Obr. 1.21: Pod systém na kontrolu uhlu kyvadla.

Tvorba masky

Maska slúži ako vlastné používateľské rozhranie bloku resp. podsystému. Maskovaním bloku zapúzdrime blokovú schému tak, aby mala podľa potreby, vlastné dialógové okná s nastaviteľnými parametrami, popisy bloku, výzvy na zadanie parametrov alebo texty s nápovedami. Vďaka maske blokov, sa tieto viac podobajú na bloky, ktoré nájdeme v predprogramovaných knižniciach Simulinku. Masku upravujeme pomocou editora, ktorý spustíme pravým kliknutím na blok, s následným výberom možnosti **Mask**. V maske sa nachádzajú štyri hlavné editovacie rozhrania.

Prvým z nich je záložka **Icon & Ports**. V tomto okne upravujeme vzhľad bloku. Teda jeho zobrazenie, rotáciu, inicializáciu, alebo ikonu bloku. Pre všetky bloky v knižnici AeroLibrary, boli vytvorené vlastné ikony, ktoré sa do masky vkladajú príkazom `image('assets/Pote.png')`.

Ďalšou v poradí je záložka **Parameters & Dialog**. V tejto sekcií nastavujeme vzhľad a funkcie masky, ktorá sa zobrazí po kliknutí na blok. V prípade bloku **Reference read**, máme v maske na výber voľbu automatickej alebo manuálnej referenčnej trajektórie. Medzi týmito možnosťami si vyberáme pomocou tlačidiel (Radio button) obr.1.23, ktorých výber taktiež formátuje zobrazenie masky bloku.

Pokiaľ si z tlačidiel vyberieme manuálnu trajektóriu, zobrazí sa ďalšia možnosť výberu, a to je model dosky Arduino. Jedná sa o roletové, alebo "popup" menu, v ktorom si vyberieme typ dosky s ktorou používame AeroShield. Dôvod tohto výberu je opísaný v časti 1.2.3, na strane 30. Pokiaľ si zvolíme Automatickú referenčnú trajektóriu, možnosť voľby dosky sa nezobrazí. Namiesto toho sa zobrazia možnosti nastavenia automatickej trajektórie. Táto funkcia je vykonávaná automaticky, zdrojovým kódom 1.19, ktorý sa nachádza v časti **Callback**, v nastaveniach tlačidiel voľby trajektórie.

```

ref = get_param(gcb,'ref');
if strcmp(ref(1), 'M')
set_param(gcb,'MaskVisibilities',{ 'on'; 'on'; 'off'; 'off'; 'off'; 'off' }) ,
else
set_param(gcb,'MaskVisibilities',{ 'on'; 'off'; 'on'; 'on'; 'on'; 'on' }) ,
end

```

Zdrojový kód 1.19: Callback funkcia.

Pomocou príkazu `get_param`, sa nám do premennej `ref`, uloží výber tlačidla v maske. Príkaz `gcb` vráti cestu daného bloku v aktuálnom systéme. Podmienka `if` kontrolouje ktorú z možností sme si vybrali, pomocou príkazu `strcmp`, ktorý porovnáva prvé písmeno z premennej `ref`, s písmenom 'M' (Manual). Pokiaľ sa tieto zhodujú, príkaz `set_param`, spolu s príkazom `MaskVisibilities`, nastaví polia tlačidiel ako viditeľné, teda s hodnotou 'on'. Pokiaľ sa písmená nezhodujú a teda bola vybraná trajektória automatická, možnosť výberu dosky Arduino, je skrytá príkazom 'off' a namiesto nej, sa zobrazia ďalšie štyri editovateľné polia.

`Initialization` tvorí tretiu záložku v maske. V tejto záložke môžeme pridať MATLAB kód, ktorý ovláda a formátuje podsystémy. V maske bloku `Reference read`, máme v tejto záložke vložený kód 1.20. Podľa pravdivosti `if` podmienky, príkaz `set_param(gcb, 'LabelModeActiveChoice')` aktivuje podsystém bloku, ktorého názov je uvedený ako tretí parameter v zátvorke príkazu.

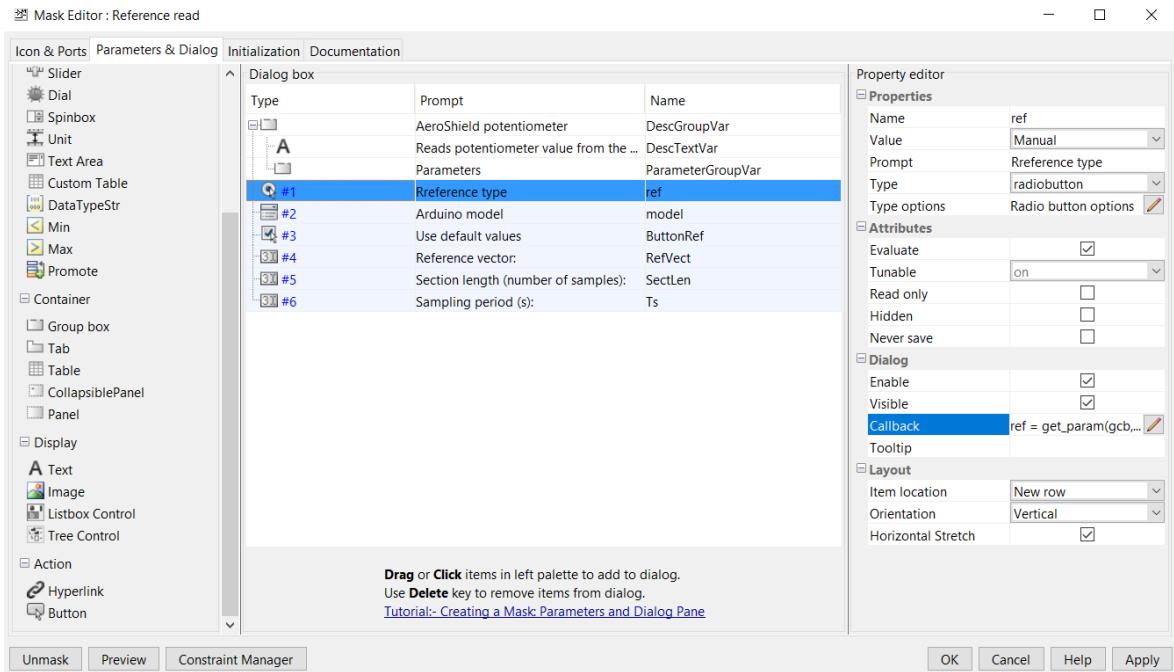
```

ref = get_param(gcb, 'ref');
model = get_param(gcb, 'model');
if strcmp(ref, 'Manual')
    if strcmp(model, 'Due')
        set_param(gcb, 'LabelModeActiveChoice', '12bit');
    elseif strcmp(model, 'MKR1000')
        set_param(gcb, 'LabelModeActiveChoice', '12bit');
    elseif strcmp(model, 'MKR1010')
        set_param(gcb, 'LabelModeActiveChoice', '12bit');
    elseif strcmp(model, 'MKRZero')
        set_param(gcb, 'LabelModeActiveChoice', '12bit');
    elseif strcmp(model, 'Nano 33 IoT')
        set_param(gcb, 'LabelModeActiveChoice', '12bit');
    elseif strcmp(model, 'Others')
        set_param(gcb, 'LabelModeActiveChoice', '10bit');
    end
end
if strcmp(ref, 'Automatic')
    set_param(gcb, 'LabelModeActiveChoice', 'Auto');
end

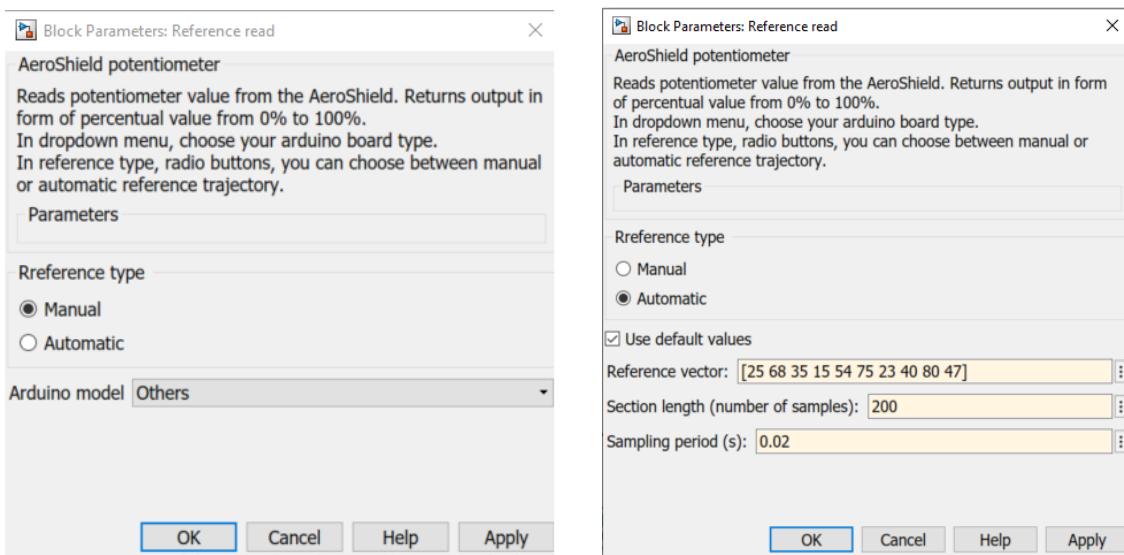
```

Zdrojový kód 1.20: Initialization- maska Reference read.

Pre lepšie porozumenie blokov a ich funkcií, bol v API Simulink zostavený inštruktažny príklad `AeroShieldOpenLoop` obr.1.24. Tento príklad funguje na princípe merania hodnoty bežca potenciometra na Shielde, a následného zapisovania nameranej hodnoty na akčný člen sústavy. Zároveň je meraný uhol v ktorom sa kyvadlo nachádza. Obe tieto hodnoty sú priebežne zobrazované na grafe.



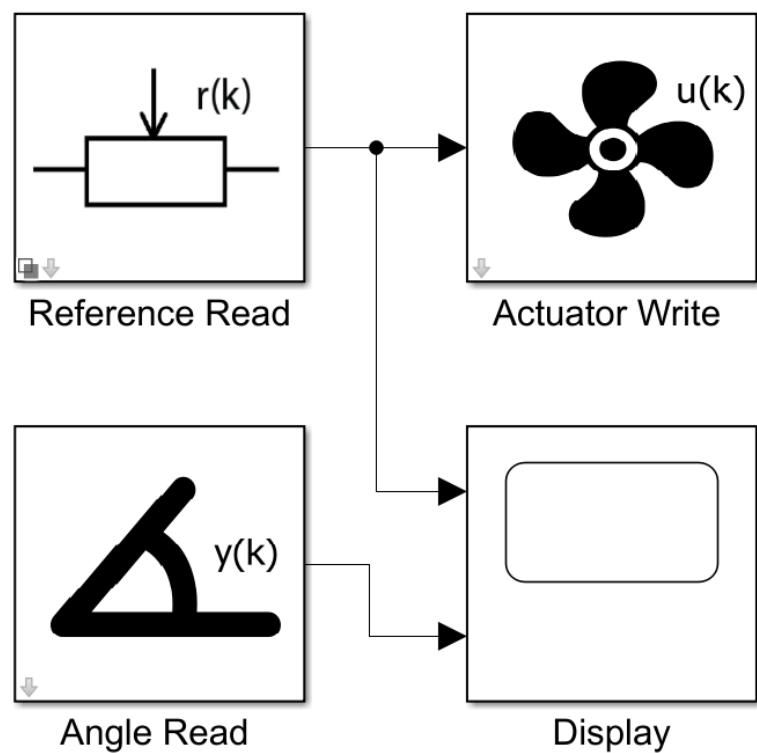
Obr. 1.22: Nastavenia parametrov masky bloku Reference read.



(a) Nastavenia masky pri výbere manuálnej trajektórie.

(b) Zobrazenie masky pri výbere automatickej trajektórie.

Obr. 1.23: Reference read- Simulink.



Obr. 1.24: AeroShield_OpenLoop.

2 Didaktické príklady

Pre AeroShield bolo v prostredí Arduino IDE, MATLAB a Simulink vytvorených niekoľko vzorových programov, ktoré demonštrujú všetky jeho funkcie. Programy sú rozdelené do dvoch veľkých skupín, konkrétnie programy v otvorenej slučke bez späťnej väzby a programy v uzavretej slučke so spätnou väzbou.

Ich rozdiel spočíva v tom že pri riadení bez späťnej väzby, hovoríme o ovládaní systému, kedy sa snažíme dosiahnuť žiadané hodnoty výstupov bez späťnej informácie o vykonaní procesu, alebo o jeho hodnote. V prípade riadenia so spätnou väzbou sa jedná o reguláciu. Pri regulácii sa kontroluje bezprostredný účinok riadenia, ktorý sa porovnáva so žiadanou hodnotou výstupu a na vyrovnanie ich vzájomnej chyby, sa okamžite vykonáva zásah do vstupných veličín.

2.1 Programy v otvorenej slučke, bez späťnej väzby

2.1.1 Arduino IDE

Ako prvý príklad si ukážeme program s názvom `AeroShield_OpenLoop.ino` napísaný v prostredí Arduino IDE. Hlavnou ideou tohto programu je jednoduché ovládanie otáčok motorčeka, pomocou potenciometra. Na začiatku programu inicializujeme hlavnú knižnicu AeroShieldu pomocou príkazu `#include "AeroShield.h"`. Následne deklarujeme premenné, ktorých hodnoty budú vypisované na sériový monitor.

```
float startAngle=0;           // Premenna pre nulovy uhol
float lastAngle=0;            // Premenna pre maximalny uhol
float pendulumAngle;          // Uhol natocenia kyvadla
float referencePercent;       // Hodnota potenciometra
float CurrentMean;           // Hodnota prudu odoberaneho
motorom
```

Zdrojový kód 2.1: AeroShield open loop dekleracia.

Nasleduje časť `setup()`, v ktorej ako prvé, prebehne nastavenie rýchlosťi sériovej komunikácie `Serial.begin(115200)`. Číslo 115 200 predstavuje počet zmien, stavu z 0 na 1 resp. zo stavu high na stav low, za sekundu. Toto tempo signálnej rýchlosť nazývame `baud rate`. Nasleduje funkcia `AeroShield.begin()`, ktorá sleduje prítomnosť magnetu, a pred nastaví potrebné premenné a funkcie pinov. Poslednou funkciou je kalibrácia kyvadla `AeroShield.calibration()`, spolu s výpočtom začiatočného a koncového uhla kyvadla.

```
void setup() {                                // Setup prebehne len jeden krat
```

```

Serial.begin(115200);           // Zaciatok seriovej
                               komunikacie
AeroShield.begin();            // Inicializacia AeroShieldu
startangle = AeroShield.calibration(AeroShield.
                                     getRawAngle());    // Kalibracia kyvadla
lastangle= startangle+1024;   // Kalkulacia uhu kyvadla
                             pre map function
}

```

Zdrojový kód 2.2: AeroShield open loop setup().

V cykle `loop()` prebehne ako prvé mapovanie uhu kyvadla pomocou funkcie `AutomationShield.mapF1`. Nasleduje čítanie hodnoty potenciometra, ktorá slúži na ovládanie akčného člena pomocou funkcie `AeroShield.actuatorWrite()`. Na sériový súradnicový zapisovač sa modrou farbou vykreslí uhol kyvadla, červenou farbou hodnota potenciometra a zelenou veľkosť prúdu odoberaného motorom obr.2.1.

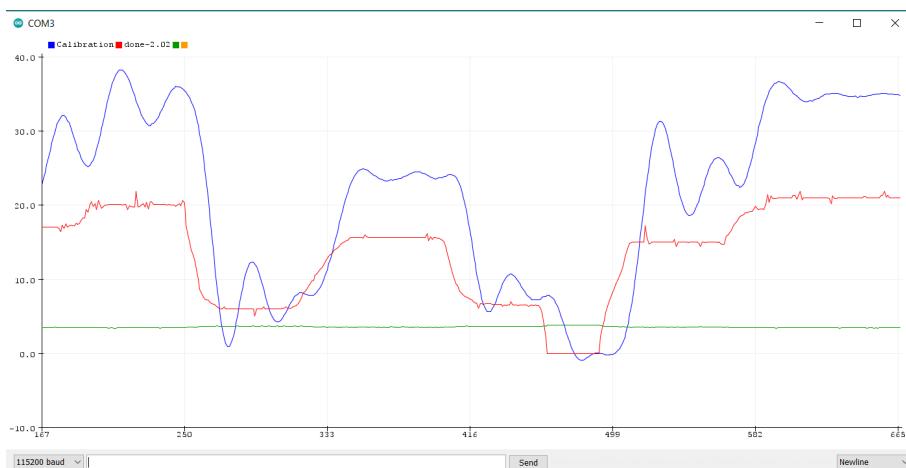
```

void loop() {
    referencePercent= AeroShield.referenceRead(); // Citanie
                                                   potenciometra
    AeroShield.actuatorWrite(referencePercent); // Pohyb
                                                 akcneho clenu
    CurrentMean= AeroShield.currentMeasure(); // Meranie
                                                 prudu

    Serial.print(pendulumAngle);
    Serial.print(" ");
    Serial.print(referencePercent);
    Serial.print(" ");
    Serial.print(CurrentMean);
    Serial.println(" ");
}

```

Zdrojový kód 2.3: AeroShield open loop loop().



Obr. 2.1: Výstup z programu AeroShieldOpenLoop.ino.

2.1.2 MATLAB

V príklade `AeroShieldOpenLoop.m` si ukážeme výhody a možnosti zobrazovania výstupov, v prostredí MATLAB. Výstupy vieme zobrazovať nielen na sériovom monitore, alebo súradnicovom zapisovači, ale máme možnosť tvoriť grafy, tabuľky a tieto výstupy upravovať a ukladať.

Na začiatku kódu vymazame všetky premenné a objekty, pomocou série príkazov `Clear all`, `clc`. Následne načítame knižnicu AeroShieldu a vykonáme funkciu `AeroShield.begin()`. Kód pokračuje kalibráciou nulového uhlu kyvadla, zadefinovaním premenných na počítanie času, ako aj premenných na ukladanie hodnôt potenciometra a uhlu kyvadla.

```
% vymazanie premennych a objektov
clear all
clc

% nacitanie kniznice AeroShieldu
AeroShield=AeroShield;
% vytvorenie objektov arduino , as5600
AeroShield.begin();
% kalibracia
startangle= AeroShield.calibration();
lastangle=startangle+2048;

% premenne na pocitanie casu
time = 0;
count = 0;
angle = 0; % uhol kyvadla
potentiometer = 0; % hodnota potenciometra
```

Zdrojový kód 2.4: AeroShield open loop inicializacia.

Ďalej pokračujeme s definovaním vlastností grafu na zobrazovanie hodnôt potenciometra a uhlu kyvadla. Zvolili sme spôsob kontinuálneho vykreslovania grafu, kedy obe strany grafu zobrazujú rozdielne premenné a rozsahy, ktoré sa prispôsobujú veľkosti premenných. Zapisovanie premenných na rôzne strany grafu funguje vďaka príkazom, `yyaxis right` a `yyaxis left`, za ktorými nasleduje samotné vykreslenie grafu pomocou príkazu `plot()`. Volíme si taktiež názvy osí grafu a jeho legendu. Medzi vykresleniami jednotlivých premenných použijeme príkaz `hold on`, ktorý zabezpečí zapísanie premenných do jedného grafu. Na konci kódu je príkaz `tic` ktorý začne počítať prejdený čas, od jeho spustenia.

```
yyaxis right % set plotting to right axis
plotGraph = plot(time,angle,'-r') % plotting angle value
ylabel('Angle (degree)', 'FontSize',15); % label settings
xlabel('Time (s)', 'FontSize',15); % label settings
hold on % hold on makes sure all of the variables are
        plotted

yyaxis left % Set plotting to left
axis
plotGraph1 = plot(time,potentiometer,'-b') % plotting
        potentiometer value
title('Pendulum plot','FontSize',15); % title settings
```

```

ylabel('Percent', 'FontSize', 15) % label settings
legend('Potentiometer value', 'Pendulum angle') % legend for
plots
grid('on'); % grid for plot 'off' to turn
off grid

tic % time keeping

```

Zdrojový kód 2.5: AeroShield open loop grafy.

Nasleduje while cyklus, ktorý je ukončený zatvorením vykreslovaného grafu. V cykle najskôr čítame hodnotu potenciometra pomocou príkazu `AeroShield.referenceRead()` a túto hodnotu zapisujeme na akčný člen príkazom `AeroShield.actuatorWrite()`. Nasleduje čítanie uhlia kyvadla `AeroShield.getRawAngle()`, za ktorím prebehne mapovanie premennej z hodnoty raw na stupne. Premenná `count` slúži na počítanie počtu prejdených cyklov, na tvorbu usporiadaneho radu premenných, ako aj na vykreslovanie pohyblivej x-ovej osi grafu obr.2.2. Ľavá stupnica grafu je stacionárna a zobrazuje hodnotu potenciometra, pravá stupnica zobrazuje uhol kyvadla v stupňoch a svoje rozpätie zväčšuje, alebo zmenšuje v závislosti na výchylke kyvadla. Na konci programu ešte nájdeme if podmienku, ktorá kontroluje uhol kyvadla, ktorý ak nadobudne hodnotu väčšiu ako 110° , proces sa automaticky ukončí a vypíše sa upozornenie. Posledný príkaz `clear AeroShield.arduino` vymaže objekt `arduino` a pripraví MATLAB na spustenie ďalšieho programu.

```

while ishandle(plotGraph) % loop will run until plot is
closed

pwm = AeroShield.referenceRead(); % read potentiometer value
AeroShield.actuatorWrite(pwm); % actuate
RAW= AeroShield.getRawAngle(); % read raw angle value
angle_ = mapped(RAW, startangle , lastangle , 0, 180); % map raw
value to degree

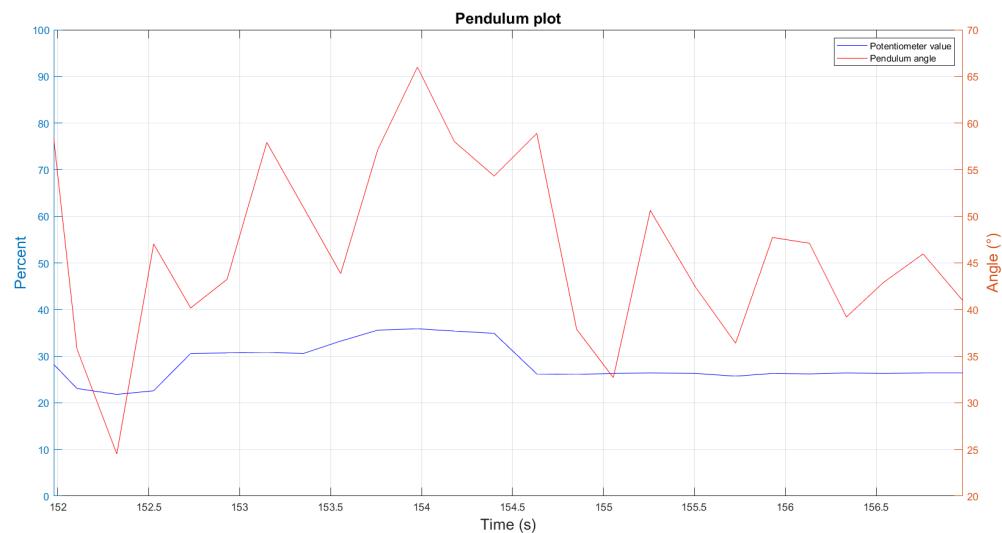
count = count + 1; % cycle counter
time(count) = toc; % time keeping
angle(count) = angle_(1); % angle value in
time
percenta= mapped(pwm, 0.0 , 5.0 , 0.0 , 100.0); % map pwm to
percent
potentiometer(count) = percenta(1); % pententiometer
value in time
set(plotGraph, 'XData', time , 'YData' , angle); % plot first data
set(plotGraph1, 'XData', time , 'YData' , potentiometer); % plot second
data
axis([time(count)-5 time(count) 0 100]); % "running" x
axis settings

if (angle_ > 110) % if angle of
pendulum bigger than 110degree
AeroShield.actuatorWrite(0.0); % stop the motor
disp('Angle of pendulum too high. AeroShield is turned off')
break % stop the program
end
end

```

```
clear AeroShield.arduino;
```

Zdrojový kód 2.6: AeroShield open loop, while cyklus.

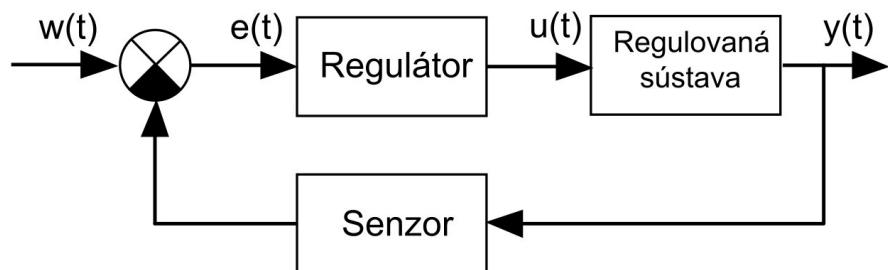


Obr. 2.2: Výstup z programu AeroShieldOpenLoop.m.

3 PID regulácia

Skôr ako si ukážeme príklady s využitím PID regulátora, musíme si vysvetliť ako PID regulátor funguje. Základom je získavanie informácií o sledovanej resp. riadenej veličine, za pomoci senzoru a jej porovnávanie s hodnotou žiadanej. Vďaka tomu že získavame informácie o výstupe, ktoré aktívne využívame na riadenie akčného člena, môžeme hovoriť o spätnoväzbovom riadení. Spätnoväzbové riadenie, je teda také riadenie, ktoré ovplyvnuje sústavu na základe aktuálne získaných informácií o stave, v ktorom sa sústava nachádza. Pri takomto riadení existuje množstvo algoritmov, ktoré ovládajú správanie sa systému. Medzi tieto algoritmy patrí napríklad: Modelové prediktívne riadenie (MPC), Lineárno-kvadratické riadenie (LQ), Lineárne riadenie s premenlivým parametrom (LPV), Proporcionálno-integračno-derivačné riadenie (PID)...

PID regulátor obr.3.1., ovplyvňuje akčné zásahy do sústavy $u(t)$ na základe zaznamenaných výstupných informácií $y(t)$. Veľkosť akčného zásahu $u(t)$ vypočítame na základe rozdielu medzi požadovanou hodnotou $w(t)$ a hodnotou reálnou $y(t)$. Tento rozdiel označujeme tiež ako regulačná odchýlka $e(t)$. Písmeno "t" v zátvorkách predstavuje, časovú závislosť premenných.



Obr. 3.1: Schéma riadenia PID regulátorom.

Skratka PID odzrkadľuje zložky, z ktorých sa regulátor skladá:

- **P-** označuje tzv. proporcionálnu zložku. Akčný zásah smerujúci do sústavy $u(t)$, je priamo úmerný veľkosti regulačnej odchýlky $e(t)$. K_p predstavuje proporcionálnu konštantu, ktorou násobíme regulačnú odchýlku na získanie požadovaného vstupu rov.3.1.

$$u(t) = K_p e(t) \quad (3.1)$$

Konštantu K_p je veľmi dôležitou pri nastavovaní parametrov PID regulátora, lebo znižuje trvalú regulačnú odchýlku. Zmenou proporcionálnej zložky vieme taktiež

urýchliť, alebo spomaliť nábeh na požadovanú hodnotu. Pre malé hodnoty K_p je nábeh pomalý, a regulačná odchýlka veľká. Pri zvyšovaní hodnoty K_p sa regulačná odchýlka zmenšuje, zrýchľuje sa nábeh, no zároveň narastá nežiadúce kmitanie sústavy. Zvyšovanie hodnoty K_p má svoj limit, za ktorým sa sústava dostáva za hranicu stability a ďalšia regulácia nie je možná.

- **I-** predstavuje integračnú zložku. Integrálny riadiaci člen je priamo úmerný veľkosti ako aj trvaniu chyby. Pokiaľ má regulovaná veličina menšiu hodnotu ako je požadovaná, integrálna časť PID sa zväčšuje. Naopak pokiaľ je hodnota regulovanej veličiny nižšia ako hodnota požadovaná, integrálna časť PID klesá. Tento údaj sa následne vynásobí integračnou konštantou K_i a je pripočítaný ku hodnote akčného zásahu rov.3.2. Príliš veľká hodnota K_i má za následok zvyšovanie nežiaduceho kmitania resp. nad a pod regulovania.

Veľkosť integrálnej zložky PID regulátora ovplyvňuje regulačnej odchýlky, doba jej trvania, ako aj integračná konštantu K_i .

$$u(t) = K_i \int_0^t e(\tau) d\tau \quad (3.2)$$

- **D-** reprezentuje derivačnú zložku, ktorá predpovedá správanie sa systému, pomocou derivácie regulačnej odchýlky v čase. Rýchlosť zmeny je následne prenásobená derivačnou konštantou K_d . Vstup je teda počítaný podľa rov.3.3. Derivačná zložka PID regulátora zmierňuje kmitanie systému, zároveň však spomaľuje rýchlosť odozvy na skokovú zmenu. V praxi sa derivačná zložka veľmi nepoužíva a využívaný je P alebo PI regulátor.

$$u(t) = K_d \frac{de(t)}{dt} \quad (3.3)$$

Pre lepšie chápanie je v tab.3.1[35] zhrnutý vplyv jednotlivých zložiek PID, na odozvu systému v uzavorennej slučke a to pri zvyšovaní hodnoty K_p , K_i a K_d .

	Regulačná odchýlka	Rýchlosť ustálenia	Prekmit	Rýchlosť odozvy	Stabilita
K_p	znižuje	malý vplyv	zvyšuje	zvyšuje	znižuje
K_i	znižuje	znižuje	zvyšuje	zvyšuje	znižuje
K_d	malý vplyv	zvyšuje	znižuje	znižuje	zvyšuje

Tabuľka 3.1: Odozva systému na zmenu konštánt.

Spojením jednotlivých samostatných zložiek, získame kompletný vzťah pre PID regulátor rov.3.4.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \quad (3.4)$$

$$u(t) = K_p \left(e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T_d \frac{de(t)}{dt} \right) \quad (3.5)$$

Rovnica 3.4 predstavuje jeden z možných tvarov zápisu vzťahu pre PID regulátor. V praxi sa často využíva tvar rov.3.5, z dôvodu lepšej interpretácie parametrov ktoré rovnici tvoria. T_i predstavuje integračnú a T_d derivačnú časovú konštantu, pričom ich vzťah s konštantami K_i a K_d je v tvare $T_i = (K_p/K_i)$ a $T_d = (K_d/K_p)$. Jednotlivé zložky v zátvorke tvoria novú a samostatnú regulačnú odchýlku, ktorá je ešte násobená konštantou K_p . Derivačná zložka predpovedá hodnotu regulačnej odchýlky T_d sekúnd(vzoriek) do budúcnosti a integračná zložka sa snaží korigovať súčet hodnoty regulačných odchýlok do T_i sekúnd(vzoriek)[35].

Predchádzajúce rovnice PID regulátorov fungujú pri spojitych procesoch. Avšak pri implementácii PID pomocou číslicových regulátorov, musíme rovnicu transformovať do jej diskrétnej podoby rov.3.6.

$$u(kT) = K_p \left(e(kT) + \frac{T}{T_i} \sum_{i=0}^k e(iT) + \frac{T_d}{T} [e(kT) - e[(k-1)T]] \right) \quad (3.6)$$

Diskrétna forma PID regulátora je využívaná z toho dôvodu že Arduino resp. počítače nie sú schopné nepretržito zaznamenávať merané hodnoty. Dáta sú preto spracovávané v diskrétnych okamihoch kT , ktorým hovoríme vzorky. Proces získavania takýchto vzoriek v istej pravidelnom frekvencii, nazývame vzorkovanie. Vzorkovanie môže mať rýchlosť niekoľko minút, pri pomali sa meniacich systémoch, až po mikrosekundy, pri veľmi rýchlych systémoch. Rovnicu v tvare 3.6 využíva na implementáciu PID regulátora knižnica `AutomationShield` a teda je aj súčasťou didaktických príkladov pre `AeroShield`.

3.1 Programy v uzavorennej slučke, so spätnou väzbou

3.1.1 Arduino IDE

V tomto príklade využívame na riadenie PID regulátora, vopred pripravenú knižnicu `PIDAbs`, ktorá je volaná z knižnice `AeroShield`. Za účelom vzorkovania, využívame knižnicu `Sampling`, ktorú načítame do príkladu príkazom `#include <Sampling.h>`. Pri voľbe referenčnej trajektórie máme na výber z dvoch možností. Prvou je voľba manuálnej trajektórie, ktorej referenčnú hodnotu nastavujeme pomocou potenciometra na shielde. Druhou voľbou je automatická trajektória, ktorá má vopred naprogramované referenčné hodnoty. Voľbu trajektórie meníme zmenou hodnoty premennej `MANUAL` z 0 na 1 v príkaze `#define MANUAL 0`. Parametre regulátora K_p , T_i a T_d slúžia ako symbolické parametre, pre lepší prehľad. Ich hodnotu zapisujeme do PID knižnice pomocou metódy `PIDAbs.setKp(KP)`. Vzorkovacia períoda T_s je nastavená na hodnotu troch milisekúnd. Períoda vzorkovania je priradená riadiacemu algoritmu PID, ako aj knižnici na vzorkovanie.

```
#define KP 1.7          // PID Kp konstanta
#define TI 3.8           // PID Ti konstanta
#define TD 0.25          // PID Td konstanta
```

```

float startAngle=0;           // Premenna pre nulovy uhol kyvadla
float lastAngle=0;            // Premenna pre mapovanie uhlu kyvadla
float pendulumAngle;          // Realna hodnota uhlu kyvadla

unsigned long Ts = 3;          // Vzorkovacia perioda
unsigned long k = 0;            // Index vzorky
bool nextStep = false;         // Povolenie kroku vzorky
bool realTimeViolation = false; // Premenna pri poruseni
                               // vzorkovania

int i=i;                      // Index referencnej hodnoty
int T=1000;                     // Dlzka sekcie dana poctom vzoriek
float R[]={45.0,23.0,75.0,32.0,58.0,10.0,35.0,
           19.0,9.0,43.0,23.0,65.0,15.0,80.0}; // Referencna
                                               // trajektoria kyvadla
float r = 0.0;                  // Referencia (Uhол ktorý chceme
                               // dosiahnut)
float y = 0.0;                  // Vystup (Realny uhol kyvadla)
float u = 0.0;                  // Vstup (Výkon motora)

```

Zdrojový kód 3.1: Načítanie knižníc a premenných do programu.

V organizačnej funkcií setup sa nastaví rýchlosť sériovej komunikácie, spolu s inicializáciou a kalibráciou AeroShieldu. Zároveň sa nastavia hodnoty PID regulátora, ako aj rýchlosť vzorkovania.

```

void setup() {
    Serial.begin(250000);           // Zaciatok seriovej
                                    // komunikacie
    AeroShield.begin();             // Inicializacia
    startAngle = AeroShield.calibration(AeroShield.
                                         getRawAngle()); // Kalibracia
    lastAngle=startAngle+1024;      // Vypocet uhlu pre
                                    // mapovanie
    Sampling.period(Ts*1000);       // Vzorkovacia
                                    // perioda
    PIDAbs.setTs(Sampling.samplingPeriod); // Vzorkovacia
                                             // perioda
    Sampling.interrupt(stepEnable);
                                    // Nasatavenie nazvu funkcie stepEnable, v
                                    // kniznici sampling
}

```

Zdrojový kód 3.2: Organizačná funkcia setup.

Funkcia `stepEnable()` je volaná z knižnice `sampling`, v intervale zadanom na začiatku programu, ako vzorkovacia períoda. Slúži na kontrolu postupnosti vzoriek a povolenie spustenia nasledujúcej vzorky. Táto funkcia je volaná vždy len na začiatku jednotlivých vzoriek. Pokiaľ je teda v trvaní jednej vzorky spustená viacero krát, vieme povedať že nastala chyba vzorkovania. Pri takejto chybe sa vypne motor a pomocou príkazu `while(1)`, je ukončené vykonávanie programu.

Pokiaľ nedošlo ku chybe vzorkovania, funkcia povolí vykonanie nasledujúcej vzorky, zmenou hodnoty premennej `nextStep`, na hodnotu "true" teda 1.

```

void stepEnable() {
    if(nextStep == true) {           // Pokial predosla vzorka
        stale trva
}

```

```

        realTimeViolation = true; // Nastala chyba
        vzorkovania
        Serial.println("Real-time samples violated.");
        // Vypis chybovu hlasku
        analogWrite(5,0);           // Vypni motor
        while(1);                  // Ukonci vykonavanie
        programu
    }
    nextStep = true;             // Povol nasledujuci vzorku
}

```

Zdrojový kód 3.3: Funkcia stepEnable().

V organizačnej funkcií `loop()`, je ako prvá, kontrolovaná podmienka `if (pendulumAngle > 120)`. Táto kontrola slúži ako ochranný mechanizmus, pred pretočením kyvadla o príliš veľký uhol. Ďalej je v rámci vzorkovania volaná funkcia `step()`. V if podmienke je testovaná premenná `nextStep`. Pokiaľ táto premenná nadobudne hodnotu "true", teda 1, podmienka sa splní a vykoná sa funkcia `step()`, za ktorou sa premennej `nextStep` priradí hodnota "false", teda 0.

```

void loop() {
    if(pendulumAngle>120){           // Bezpecnostna podmienka
        kyvadla
        AeroShield.actuatorWrite(0); // Pokial je uhol
        vacsi ako 120
        while(1);                  // stupnov, motor sa
        vypne
    }
    if (nextStep) {                  // Pokial nextStep == 1
        step();                     // Spusti funciu step()
        nextStep = false;           // Vynuluj premennu
    }
}

```

Zdrojový kód 3.4: Organizačná funkcia `loop()`.

Funkcia `step()` vykonáva samotné meranie, ovládanie a výpočty potrebné pri riadení systému pomocou PID regulátora. Na začiatku funkcie sa zvolí buď manuálna, alebo automatická trajektória. Pri automatickej dráhe je dôležité, vedieť kedy bol dosiahnutý koniec predprogramovanej trajektórie. Táto kontrola je vykonávaná pomocou porovnávania veľkostí premennej `i`, ktorá zaznamenáva počet vykonaných sekcií trajektórie, oproti veľkosti pola `R[]`, v ktorom sú zapísané hodnoty jednotlivých sekcií. Príkaz `sizeof(R)/sizeof(R[0])`, vráti počet prvkov pola `R[]`. Zároveň sa kontroluje dĺžka chodu sekcie. Pokiaľ výraz `k % (T*i)`, dosiahne hodnotu 0, nastaví sa ako trajektória nasledujúca sekcia.

Následne je mapovaný uhol kyvadla na percentuálnu hodnotu od 0% do 100%, ktorá je uložená ako premenná `y`. Veľkosť regulačnej odchýlky, obmedzenie integračného nasýtenia⁹(angl. anti-windup), ako aj hodnotu saturácie systému¹⁰, zadávame do algo-

⁹K nasýteniu integračnej zložky dochádza, v prípade že akčný člen nie je schopný dosiahnuť požadovanú referenčnú hodnotu. V takom prípade začne hodnota integračná zložka nekontrolovaťne stúpat.

¹⁰Ked' sa hodnota spätnoväzbového riadiaceho systému dostane do oblasti nasýtenia ktorejkoľvek z jeho zložiek, zmena vstupu nasýtenej zložky nespôsobí žiadnu zmenu na jej výstupe. Zosilnenie vtedy dosahuje nulové hodnoty.

ritmu na výpočet akčného zásahu v tvare PIDAbs.compute(r-y,minSaturacia,maxSaturacia,antiWindupMin,antiWindupMax);

```

void step() {
    #if MANUAL // Pokial je zvolena manualna trajektoria
        r = AeroShield.referenceRead(); // Referencna hodnota
        z potenciometra
    #else
        if(i>(sizeof(R)/sizeof(R[0]))) { // Pokial automaticka
            trajektoria skoncila
            analogWrite(5,0); // Vypni motor
            while(1); // Zastav program
        } else if (k % (T*i) == 0) { // Pokial je dosiahnutý
            koniec sekcie trajektorie
            r = R[i]; // Postup na dalsiu sekciu
            i++; // Pripracuj postup o jednu
            sekciu
        }
    #endif

    y= AutomationShield.mapFloat(AeroShield.getRawAngle(), startangle, lastangle, 0.00, 100.00);
    // Mapovanie uhlu kyvadla na percenta
    u = PIDAbs.compute(r-y,0,100,0,100); // Vypočet PID
    AeroShield.actuatorWrite(u); // Aktuator

    Serial.print(r); // Referencna hodnota
    Serial.print(" , ");
    Serial.print(y); // Vystup
    Serial.print(" , ");
    Serial.println(u); // Akcny zasah
    k++; // Pocitadlo vzoriek
}

```

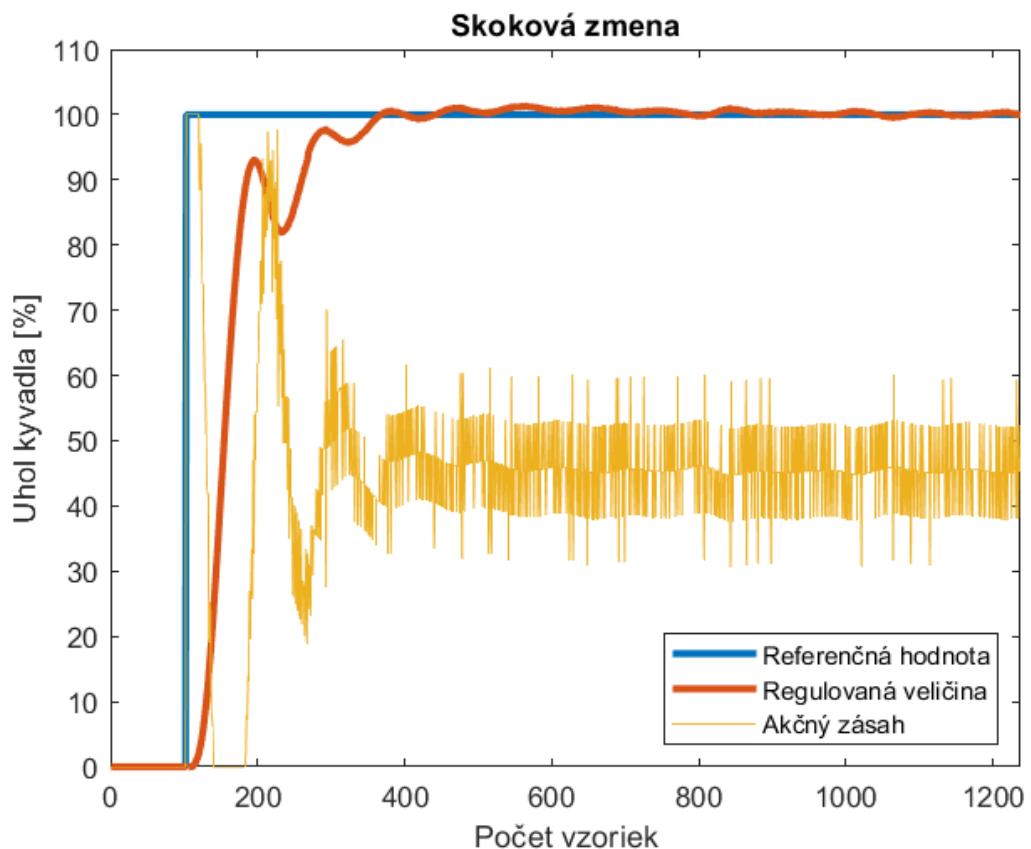
Zdrojový kód 3.5: Funkcia step().

Výstupy

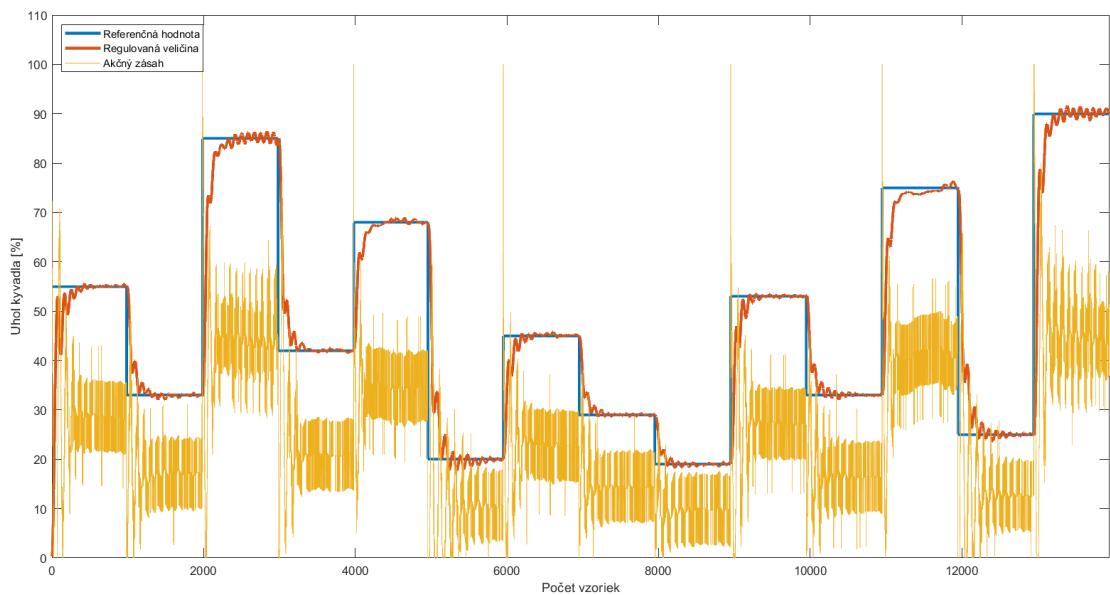
Všetky výstupy z API Arduino IDE, boli zaznamenávané programom CoolTerm a následne vykreslené do grafov v prostredí MATLAB.

Na obr.3.2, vidíme reakciu systému na skokovú zmenu z nulovej referenčnej hodnoty na hodnotu maximálnu, teda 100%. Pomocou sledovania odozvy systému vieme lepšie nastaviť parametre PID regulátora. Systém nadobudne 1% regulačnú odchýlku v priebehu 500 vzoriek, čo znamená čas približne jeden a pol sekundy.

Manuálna trajektória obr.3.4, má oproti automatickej trajektórii obr.3.3, väčšie rozdiely v hodnote akčného zásahu. Je to spôsobené kontinuálnou zmenou referenčnej hodnoty, ako aj miernym šumom signálu z potenciometra. V rámci obr.3.4 si ešte môžeme všimnúť časť medzi vzorkami 5000-6000, ktorá je priblížená na obr.3.5. Jedná sa o manuálne zavedenie výchylky, pomocou buchnutia do kyvadla. Systém na takúto zmenu reaguje zmenou akčného zásahu a regulačnú odchýlku menšiu ako 2%, nadobúda v priebehu jednej sekundy.



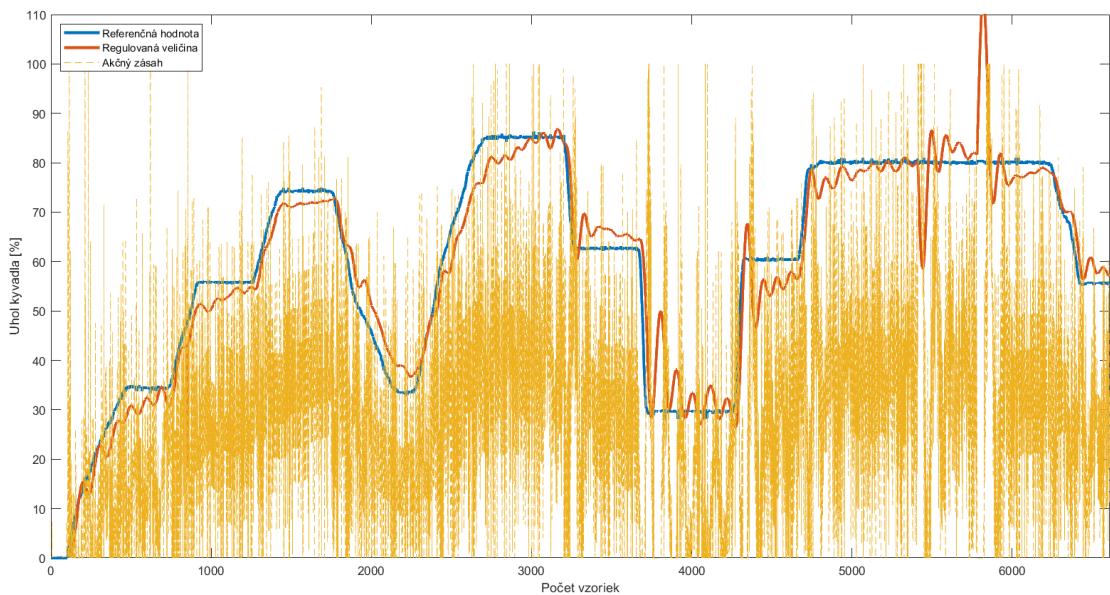
Obr. 3.2: Reakcia systému na jednotkový skok.



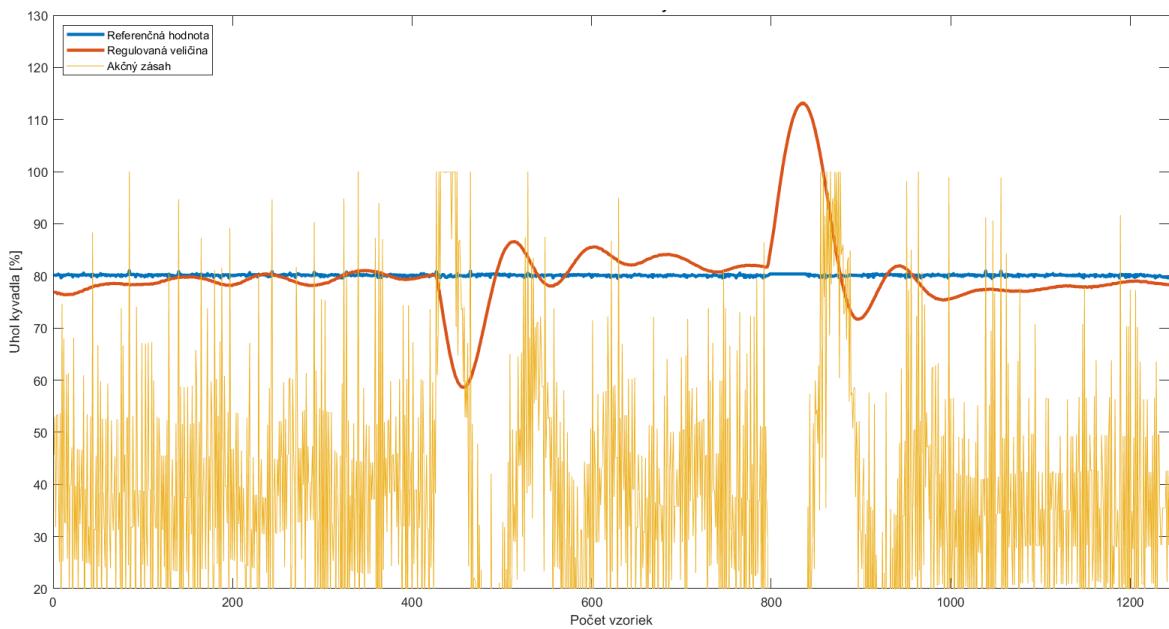
Obr. 3.3: Automatická trajektória.

3.1.2 MATLAB

Príklad na ukážku fungovania PID regulátora bol taktiež vytvorený v prostredí MATLAB a simulink. Princíp fungovania PID regulátora je rovnaký ako pri Arduino



Obr. 3.4: Manuálna trajektória.



Obr. 3.5: Manuálne zavedenie výchylky.

IDE, avšak mení sa spôsob vzorkovania a nastavovania parametrov PID. V prípade MATLABU využívame, na výpočet akčného zásahu, knižnicu PID.m, ktorá bola taktiež vytvorená v rámci programu AutomationShield. Na nastavenie parametrov PID slúži príkaz `PID.setParameters(Kp, Ti, Td, Ts)`.

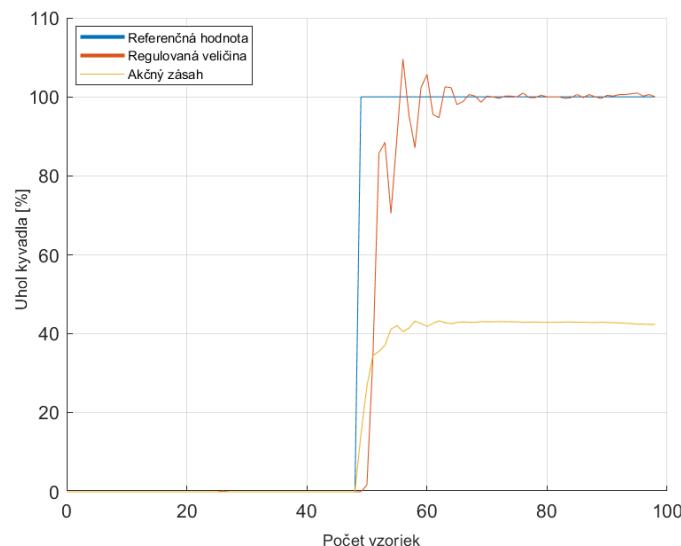
Na vzorkovanie využívame funkcie `TIC` a `TOC`, ktoré merajú prejdený čas. Funkcia `TIC` zaznamenáva aktuálny čas a funkcia `TOC` používa zaznamenanú hodnotu na výpočet uplynulého času. Ak je splnená podmienka `if (toc>=Ts*k)`, povolí sa posun na nasledujúcu vzorku, pričom premenná `k`, udáva počet vykonaných vzoriek. Dáta sú

postupne zapisované do poľa $\text{PIDresponse}(k, :) = [r \ y \ u]$, a toto pole je vykreslované pomocou funkcie $\text{plotLive}(\text{PIDresponse}(k, :))$. Na konci sú všetky dáta uložené, a teda sú prístupné aj po ukončení programu. Kompletný zdrojový kód sa nachádza v prílohe na strane xiii.

Regulácia PID v prostredí MATLAB potrebuje pre svoje fungovanie pomerne vysoký výpočtový výkon. Z tohto dôvodu je períoda vzorkovania, oproti Arduino IDE, rádovo nižšia a dosahuje hodnoty maximálne piatich vzoriek za sekundu tj. $T_s=0.2\text{s}$. Takáto rýchlosť vzorkovania je hraničná a teda pri pomalšom vzorkovaní systém nedosahuje ideálne vlastnosti. Výpočtová náročnosť sa dá znížiť, zamedzením vykreslovania grafu, alebo odstránením možnosti ukladania zaznamenaných dát.

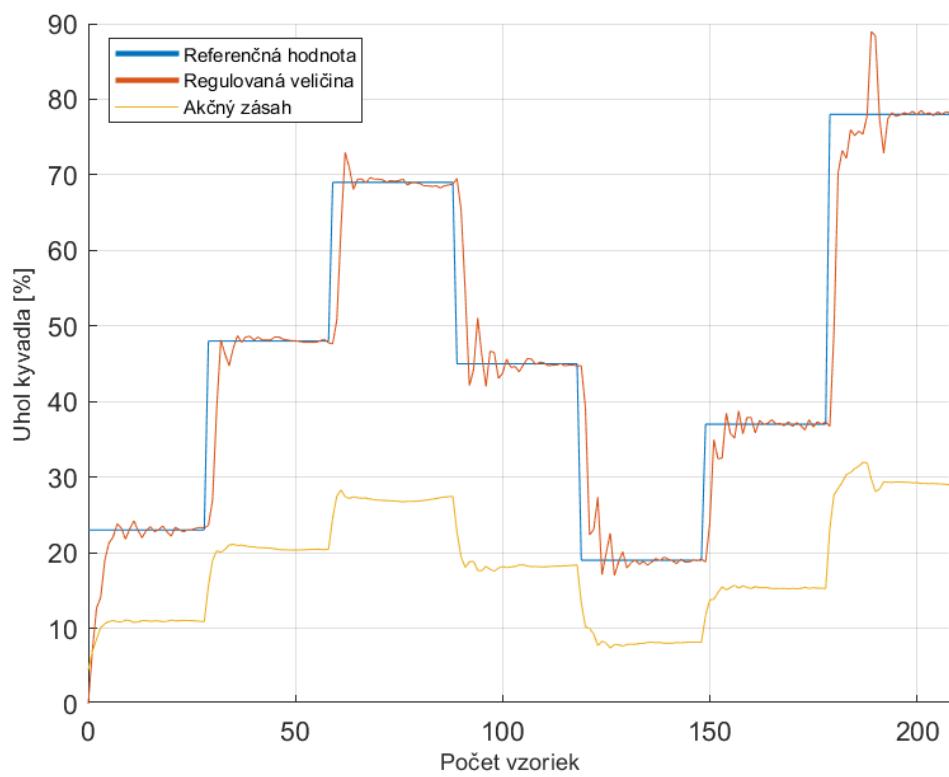
Výstupy

Všetky výstupy z príkladu 3.1.2, majú priamu funkciu vykreslovania grafov spolu s legendou výstupov. Tieto grafy majú takiež definovaný rozsah zobrazovaných hodnôt na oboch osiach, ako aj pomenovania týchto osí. Na obr.3.6 vidíme reakciu systému na jednotkový skok. Obrázok 3.7 zobrazuje automatickú trajektóriu referenčnej hodnoty a obr.3.8 trajektóriu manuálnu.

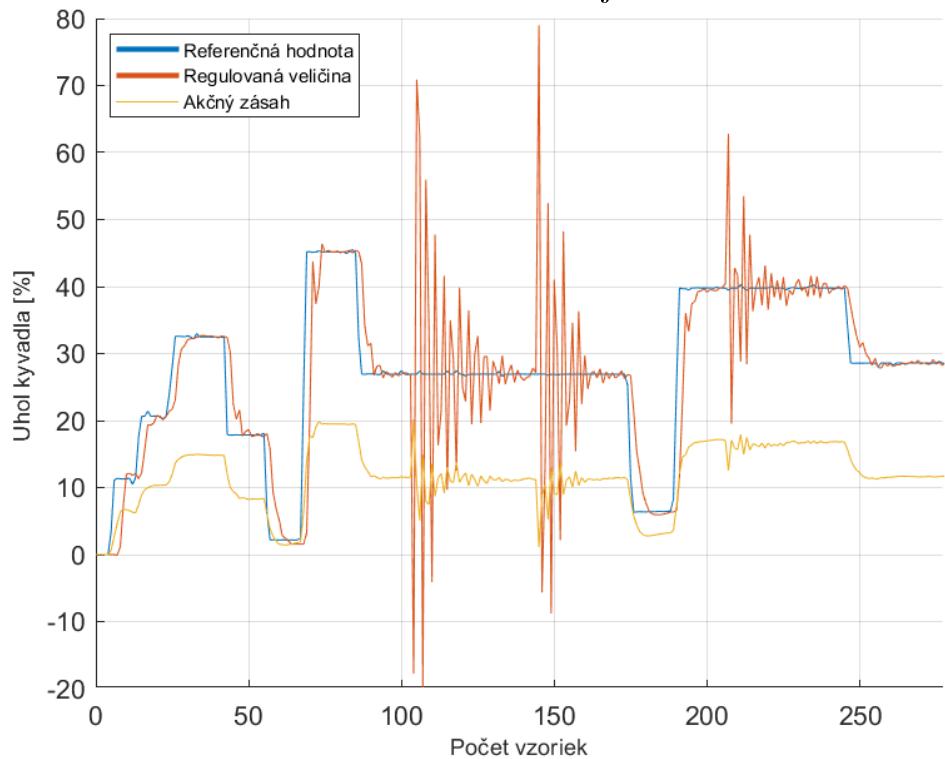


Obr. 3.6: Reakcia systému na jednotkový skok.

Pri manuálnej trajektórii bola trikrát vnesená veľká regulačná odchýlka a to pomocou úderu do kyvadla. Ako je vidieť z grafu 3.8, ustálenie systému prebieha oveľa pomalšie ako v príklade 3.1.1. Oscilácia je pomerne vysoká a pretrváva po dobu cca 35 vzoriek, čo predstavuje približne 7 sekúnd. Táto skutočnosť je spôsobená pomalšou reakciou PID regulátora, na veľkú regulačnú odchýlku. Dlhší čas potrebný na ustálenie kyvadla je spôsobený pomalším vzorkovaním, ako aj nie ideálne nastavenými hodnotami PID regulátora.



Obr. 3.7: Automatická trajektória.



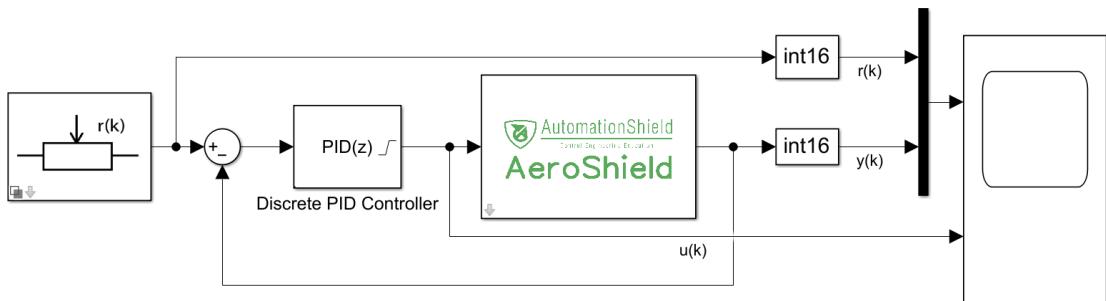
Obr. 3.8: Manuálna trajektória.

3.1.3 Simulink

Vzhľadovo pôsobí príklad PID riadenia v API Simulink jednoduchšie, ako tomu bolo pri tom istom riadení v MATLABe alebo Arduino IDE. Predpripravené bloky z knižnice AeroLibrary stačí v príklade logicky pospájať, a následne zvoliť vhodné parametre v maskách blokov. Regulovanú sústavu v tomto príklade predstavuje blok **AeroShield**, do ktorého vstupuje z bloku **Reference read** percentuálna hodnota referenčnej trajektórie. Z bloku **AeroShield** máme ako výstup uhol kyvadla, ktorý využívame na výpočet regulačnej odchýlky.

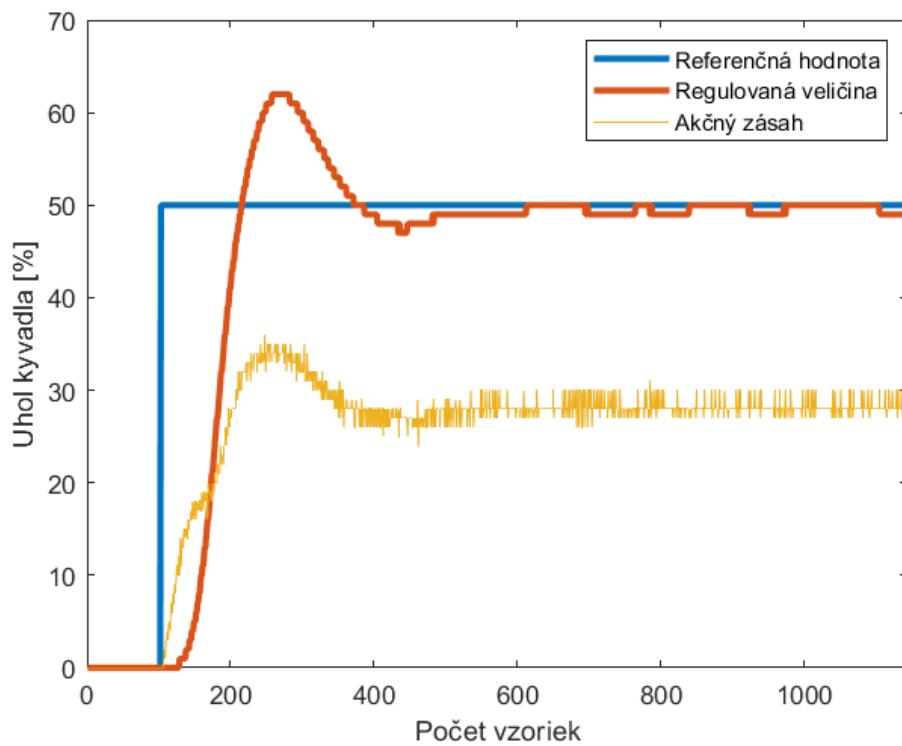
Blok **Discrete PID Controller** predstavuje riadiaci systém PID regulátora. Hodnoty jednotlivých zložiek regulátora sú: $P=0.011$, $I=200$, $D=8$. Matematická reprezentácia výpočtového algoritmu ideálneho PID regulátora, je v tvare rov.3.7:

$$P \left(1 + I * T_s \frac{1}{z - 1} + D * \frac{1}{T_s} \frac{z - 1}{z} \right) \quad (3.7)$$

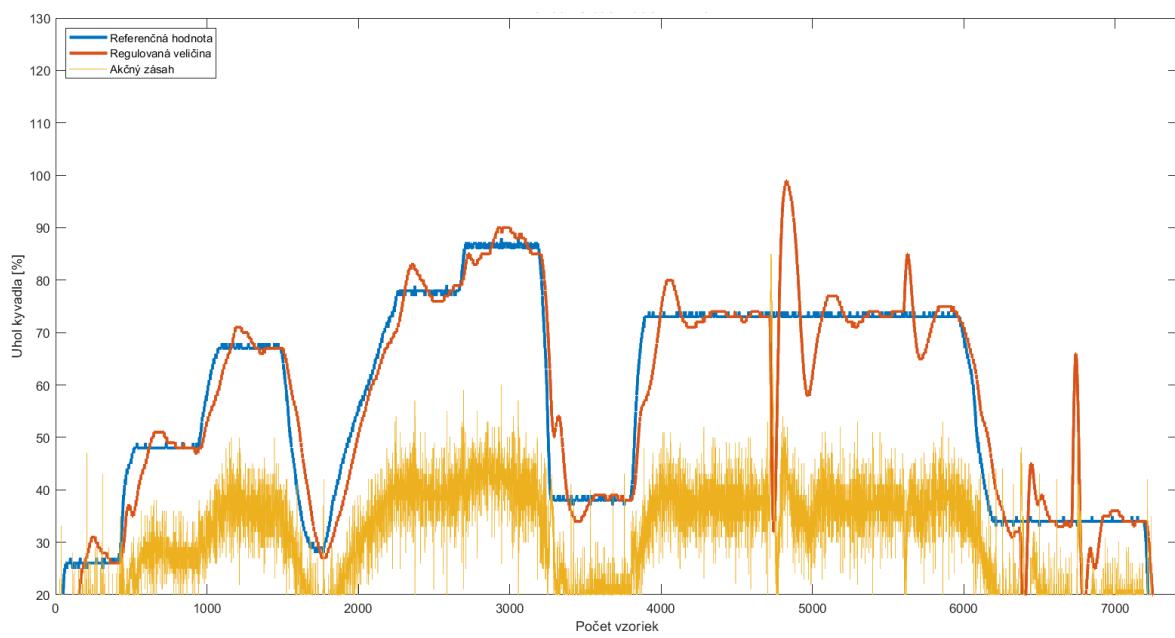


Obr. 3.9: Ukážka riadenia systému pomocou PID regulátora v API Simulink.

Výstupy



Obr. 3.10: Reakcia systému na skokovú zmenu.



Obr. 3.11: Manuálna trajektória.

4 Záver

Príkladmi z kapitoly 2 a 3, sme potvrdili funkčnosť AeroShieldu v API Arduino IDE, MATLAB a Simulink. AeroShield je teda použiteľný ako didaktická pomôcka a to aj napriek niektorým nedostatkom na softvérovom, ako aj hardvérovom rozhraní.

Na AeroShielde je zaznamenaný prúd, ktorý odoberá akčný člen sústavy. Toto meranie je pri malých zmenách prúdu pomerne nepresné a to z dôvodu ovládania motora PWM signálom. V budúcej verzii AeroShieldu môže byť použitý iný druh napájania motora, čo by malo za následok aj vylepšenie presnosti merania prúdu. Takto nameenaný prúd by sa dal následne využívať na presnejšie riadenie výkonu motora, alebo na implementáciu PID regulácie na základe prúdu a nie uhlu kyvadla.

Ďalším problémom pri PID regulácii AeroShieldu, je jeho zložité a dlhotrvajúce nastavenie parametrov. Pri príklade 3.1.2 na strane 51, bolo nastavenie parametrov najzložitejšie a strávil som pri ňom niekoľko desiatok hodín. Problémom bolo pomalšie vzorkovanie a teda aj malá zmena parametrov, spôsobila veľké zmeny na výstupe.

Ďalším z problémov bolo priame prepojenie motora so Shieldom. V prípade nechceného pretočenia ramena kyvadla, sa napájacie káble zapletú na rameno a to spôsobí zamedzenie ďalšieho otáčania. Všetky didaktické príklady súčasťne majú implementovanú softvérovú ochranu proti takému pretočeniu, avšak táto nie je 100% účinná. Tento problém by vyriešila realizácia napájania pomocou konektora so zbernými krúžkami, avšak takýto konektor stojí v priemere 15€ a teda jeho aplikácia v nízko nákladovej učebnej pomôcke je otázna. Zároveň pomocou magnetu uloženého na konci kyvadla meriame jeho uhol. Použitý konektor by preto musel mať stredovú časť s možnosťou pripojenia magnetu, alebo by sa uhol kyvadla merať iným spôsobom.

Medzi vylepšenia nasledujúceho modelu AeroShieldu môžeme zaradiť úplnú zmenu podporného systému kyvadla. Uchytenie pomocou dvoch otočených V konštrukcií, prepojených priečkou, by umožňovalo meranie natočenia na jednej strane priečky a druhou stranou by bolo realizované napájanie motora. Zaujímavým experimentom by bola možnosť hardvérovej zmeny smeru otáčania motora, pomocou prepínača. Pokiaľ by bola rýchlosť zmeny otáčania dostatočne rýchla (rádovo niekoľko desiatok milisekúnd), dať by sa realizovať príklad otočeného kyvadla, kedy je rameno kyvadla držané pomocou regulátora vo vzpriamenej polohe.

Medzi ďalšie vylepšenia pre budúcu prácu s AeroShieldom, môžeme zaradiť reguláciu pomocou iného algoritmu ako bol PID. Medzi ne môžeme zaradiť modelové prediktívne riadenie (MPC), Lineárno-kvadratické riadenie (LQ), Lineárne riadenie s premenlivým parametrom (LPV)... Pomocou týchto algoritmov by sme mohli dosiahnuť lepšie vlastnosti systému a tým pádom presnejšie riadenie.

Všetky chyby a nedokonalosti dizajnu, ako aj neschopnosť dokonalého nastavenia

uhlu kyvadla pri PID regulácii, neznemožňujú kvalitnú výuku s použitím AeroShieldu. Sú to skôr nápady a možnosti vylepšenia, ktoré môžu byť v budúcnosti na AeroShield implementované.

Literatúra

- [1] Arduino uno r3 development board microcontroller for diy project. Store. Online., -. -. , <https://sunhokey.en.made-in-china.com/product/bjkxIyAKQdhF/China-Arduino-Uno-R3-Development-Board-Microcontroller-for-DIY-Project.html>.
- [2] Arduino mega 2560 rev3. Store. Online., -. -. , <https://store.arduino.cc/collections/boards/products/arduino-mega-2560-rev3>.
- [3] Petr Horáček. Laboratory experiments for control theory courses: A survey. *Annual Reviews in Control*, 24:151–162, 2000.
- [4] Ed Edwards. All about position sensors. article. Online., -. 2021, <https://www.thomasnet.com/articles/instruments-controls/all-about-position-sensors>.
- [5] Mila Mary Job and P. Subha Hency Jose. Modeling and control of mechatronic aeropendulum. In *2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, pages 1–5, 2015.
- [6] Eniko T. Enikov and Giampiero Campa. Mechatronic aeropendulum: Demonstration of linear and nonlinear feedback control principles with matlab/simulink real-time windows target. *IEEE Transactions on Education*, 55(4):538–545, Nov 2012.
- [7] Spracovanie meraných údajov, modelovanie a identifikácia mechatronického systému aerokyvadlo.
- [8] Návrh konštrukcie, riadiacich prvkov a algoritmov mechatronického systému aerokyvadlo.
- [9] Fanavaran Sharif. Aero pendulum control system (pr22). Store. Online., 2021. 2021, <https://www.zoodel.com/en/product/ZP22344/Aero-Pendulum-Control-System-PR22>.
- [10] Gergely Takacs. Automationshield. Wiki. Online., 2021. 13.7.2021, <https://github.com/gergelytakacs/AutomationShield/wiki>.
- [11] Gergely Takacs. Automationshield. Code. Online., 2021. 23.12.2021, <https://github.com/gergelytakacs/AutomationShield>.

- [12] Saroja Dhanapal and Evelyn Wan Zi Shan. A study on the effectiveness of hands-on experiments in learning science among year 4 students. In -, 2013.
- [13] Harry Baggen. The javelin stamp. *Elector Electronics*, 1(1):0, 2003.
- [14] Arduino uno rev3. Info. Online., 2021. 2021, <https://store.arduino.cc/products/arduino-uno-rev3>.
- [15] Texas instruments tps56339 buck converters. store. Online., 2021. 2021, <https://www.mouser.ee/new/texas-instruments/ti-tps56339-buck-converters/>.
- [16] Arduino. Overview of the arduino uno components. article. Online., 2021. 2021, <https://docs.arduino.cc/tutorials/uno-rev3/intro-to-board>.
- [17] Arduino uno r3 - schematic with ch340. article. Online., 2021. 2021, http://electronoobs.com/eng_arduino_tut31_sch3.php.
- [18] DANIELLE COLLINS. What are coreless dc motors? article. Online., 2018. 09.10.2021, <https://www.motioncontrolltips.com/what-are-coreless-dc-motors/>.
- [19] Komatsu Yasuhiro, Tur-Amgalan Amarsanaa, Yoshihiko Araki, Syed Abdul Kadir Zawawi, and Takamura Keita. Design of the unidirectional current type coreless dc brushless motor for electrical vehicle with low cost and high efficiency. In *SPEEDAM 2010*, pages 1036–1039, 2010.
- [20] Pmv45en2. shop. Online., 2021. 2021, <https://www.nexperia.com/products/mosfets/small-signal-mosfets/PMV45EN2.html>.
- [21] 7mm diameter 720 coreless motor for quadcopter. shop. Online., 2021. 2021, <https://www.elecrow.com/7mm-diameter-720-coreless-motor-for-quadcopter.html>.
- [22] SHAWN HYMEL. Ina169 breakout board hookup guide. article. Online., 2021. 2021, <https://learn.sparkfun.com/tutorials/ina169-breakout-board-hookup-guide/all>.
- [23] Ina169na/250. store. Online., 2021. 2021, <https://www.ti.com/store/ti/en/p/product/?p=INA169NA/250>.
- [24] The Editors of Encyclopaedia Britannica. Hall effect. article. Online., 2021. 2021, <https://www.britannica.com/science/Hall-effect>.
- [25] Hallův snímač as5600-asom. store. Online., 2021. 2021, <https://sk.rsdelivers.com/product/ams/as5600-asom/halluv-snimac-as5600-asom-pocet-koliku-8-soic-typ/2006337>.
- [26] As5600. 2022, <https://ams.com/en/as5600>.
- [27] Tps56339. 2022, <https://www.ti.com/product/TPS56339>.

- [28] Prototyping circuit boards: Everything you need to know before you start. article. Online., 2020. 2020, <https://www.pcbnet.com/blog/prototyping-circuit-boards-everything-you-need-to-know-before-you-start/>.
- [29] What are the factors that will affect the pcb lifespan and how to extend it: 2021 newest.
- [30] Writing a library for arduino. article. Online., 2022. 2022, <https://docs.arduino.cc/hacking/software/LibraryTutorial>.
- [31] Arduino - data types. article. Online., -. 2022, https://www.tutorialspoint.com/arduino/arduino_data_types.htm.
- [32] Michael Dellnitz Martin Golubitsky. Linear mappings and bases. article. Online., -. 2022, <https://ximera.osu.edu/laode/linearAlgebra/linearMapsAndChangesOfCoordinates/linearMappingsAndBases>.
- [33] Nicholas Zambetti. A guide to arduino & the i2c protocol (two wire). article. Online., 2022. 2022, <https://docs.arduino.cc/learn/communication/wire>.
- [34] Bit shifting. article. Online., 2017. 2022, <https://www.techopedia.com/definition/26846/bit-shifting>.
- [35] Changbo Xu, Congcong Liu, Zhenfu Bi, and Chengjin Zhang. *2006 6th World Congress on Intelligent Control and Automation*.

Zdrojový kód súboru AeroShield.h

```
1      #ifndef AEROSHIELD_H
2      #define AEROSHIELD_H
3
4      #include "AutomationShield.h"
5      #include <Wire.h>
6      #include <Arduino.h>
7      #define AERO_RPIN A3
8      #define VOLTAGE_SENSOR_PIN A2
9      #define AERO_UPIN 5
10
11     class AeroShield{
12         public:
13             AeroShield();
14             float begin(bool isDetected);
15             void actuatorWrite(float PotPercent);
16             float calibration(word RawAngle);
17             float convertRawAngleToDegrees(word newAngle);
18             float referenceRead();
19             float currentMeasure();
20             int detectMagnet();
21             int getMagnetStrength();
22             word getRawAngle();
23
24         private:
25             int ang;
26             float startangle;
27             float referenceValue;
28             float referencePercent;
29             float correction1= 4.1220;
30             float correction2= 0.33;
31             int repeatTimes= 100;
32             float voltageReference= 5.0;
33             float ShuntRes= 0.1;
34             float current;
35             float voltageValue;
36             int _ams5600_Address = 0x36;
37             int _stat = 0x0b;
38             int _raw_ang_hi = 0x0c;
39             int _raw_ang_lo = 0x0d;
40             int readOneByte(int in_adr);
41             word readTwoBytes(int in_adr_hi, int in_adr_lo);
42     };
43 #endif
```

Zdrojový kód 4.1: Zdrojový kód súboru AeroShield.h.

Zdrojový kód súboru AeroShield.cpp

```

1 #include "AeroShield.h"
2
3 float AeroShield::begin(bool isDetected){
4     pinMode(AERO_UPIN,OUTPUT);
5     #ifdef ARDUINO_ARCH_AVR
6         Wire.begin();
7     #elif ARDUINO_ARCH_SAM
8         Wire1.begin();
9     #elif ARDUINO_ARCH_SAMD
10        Wire.begin();
11    #endif
12
13    if(isDetected == 0){
14        while(1){
15            if(isDetected == 1){
16                AutomationShield.serialPrint("Magnet detected \n");
17                break;
18            }
19            else{
20                AutomationShield.serialPrint("Can not detect
21                                magnet \n");
22            }
23        }
24    }
25
26    float AeroShield::convertRawAngleToDegrees(word newAngle) {
27        float retVal = newAngle * 0.087;
28        ang = retVal;
29        return ang;
30    }
31
32    float AeroShield::calibration(word RawAngle) {
33        AutomationShield.serialPrint("Calibration running... \n");
34        startangle=0;
35        analogWrite(AERO_UPIN,50);
36        delay(250);
37        analogWrite(AERO_UPIN,0);
38        delay(4000);
39
40        startangle = RawAngle;
41        analogWrite(AERO_UPIN,0);
42        for(int i=0;i<3;i++){
43            analogWrite(AERO_UPIN,1);
44            delay(200);
45            analogWrite(AERO_UPIN,0);
46            delay(200);
47        }
48        AutomationShield.serialPrint("Calibration done");
49        return startangle;
50    }
51
52    float AeroShield::referenceRead(void) {
53        referencePercent = AutomationShield.mapFloat(analogRead(AERO_RPIN), 0.0,
54                                         1024.0, 0.0, 100.0);
55        return referencePercent;
56    }
57
58    void AeroShield::actuatorWrite(float PotPercent) {
59        float mappedValue = AutomationShield.mapFloat(PotPercent, 0.0, 100.0,
60                                         0.0, 255.0);
61        mappedValue = AutomationShield.constrainFloat(mappedValue, 0.0, 255.0);
62        analogWrite(AERO_UPIN, (int)mappedValue);
63    }
64
65    float AeroShield::currentMeasure(void){
66        for(int i=0 ; i<repeatTimes ; i++){
67            voltageValue= analogRead(VOLTAGE_SENSOR_PIN);
68            voltageValue= (voltageValue * voltageReference) / 1024;
69            current= current + correction1-(voltageValue / (10 * ShuntRes));
70        }
71        float currentMean= current/repeatTimes;
72        currentMean= currentMean-correction2;
73        if(currentMean < 0.000){
74            currentMean= 0.000;
75        }
76        current= 0;
77        voltageValue= 0;
78        return currentMean;
79    }
80
81    word AeroShield::getRawAngle()
82    {
83        return readTwoBytes(_raw_ang_hi, _raw_ang_lo);
84    }

```

```

83     int AeroShield :: detectMagnet()
84     {
85         int magStatus;
86         int retVal = 0;
87         magStatus = readOneByte(_stat);
88         if (magStatus & 0x20)
89             retVal = 1;
90         return retVal;
91     }
92
93     int AeroShield :: getMagnetStrength()
94     {
95         int magStatus;
96         int retVal = 0;
97         magStatus = readOneByte(_stat);
98         if (detectMagnet() == 1)
99         {
100             retVal = 2;
101             if (magStatus & 0x10)
102                 retVal = 1;
103             else if (magStatus & 0x08)
104                 retVal = 3;
105         } return retVal;
106     }
107
108     int AeroShield :: readOneByte(int in_addr)
109     {
110         int retVal = -1;
111         Wire.beginTransmission(_ams5600_Address);
112         Wire.write(in_addr);
113         Wire.endTransmission();
114         Wire.requestFrom(_ams5600_Address, 1);
115         while (Wire.available() == 0);
116         retVal = Wire.read();
117         return retVal;
118     }
119
120     word AeroShield :: readTwoBytes(int in_addr_hi, int in_addr_lo)
121     {
122         word retVal = -1;
123         /* Read Low Byte */
124         Wire.beginTransmission(_ams5600_Address);
125         Wire.write(in_addr_lo);
126         Wire.endTransmission();
127         Wire.requestFrom(_ams5600_Address, 1);
128         while (Wire.available() == 0);
129         int low = Wire.read();
130
131         /* Read High Byte */
132         Wire.beginTransmission(_ams5600_Address);
133         Wire.write(in_addr_hi);
134         Wire.endTransmission();
135         Wire.requestFrom(_ams5600_Address, 1);
136         while (Wire.available() == 0);
137         word high = Wire.read();
138         high = high << 8;
139         retVal = high | low;
140     }

```

Zdrojový kód 4.2: Zdrojový kód súboru AeroShield.cpp.

Zdrojový kód súboru AeroShieldOpenLoop.ino

```
1 #include "AeroShield.h"
2
3     float startangle=0;
4     float lastangle=0;
5     float pendulumAngle;
6     float referencePercent;
7     float CurrentMean;
8
9     void setup() {
10
11         Serial.begin(115200);
12         AeroShield.begin(AeroShield.detectMagnet());
13         startangle = AeroShield.calibration(AeroShield.getRawAngle
14             ());
15         lastangle=startangle+1024;
16     }
17
18     void loop() {
19         if(pendulumAngle>120){
20             AeroShield.actuatorWrite(0);
21             while(1);
22         }
23
24         pendulumAngle= AutomationShield.mapFloat(AeroShield.
25            .getRawAngle(),startangle,lastangle,0.00,90.00);
26         Serial.print("pendulum angle is: ");
27         Serial.print(pendulumAngle);
28         Serial.print(" degree || ");
29
30         referencePercent= AeroShield.referenceRead();
31         Serial.print("pot value is: ");
32         Serial.print(referencePercent);
33         Serial.print(" percent || ");
34
35         AeroShield.actuatorWrite(referencePercent);
36
37         CurrentMean= AeroShield.currentMeasure();
38         Serial.print("current value is: ");
39         Serial.print(CurrentMean);
40         Serial.println(" A || ");
```

Zdrojový kód 4.3: Zdrojový kód súboru AeroShieldOpenLoop.ino.

Zdrojový kód súboru AeroShieldPID.ino

```
1 #include "AeroShield.h"
2 #include <Sampling.h>
3
4 #define MANUAL 0
5 #define KP 1.7
6 #define TI 3.8
7 #define TD 0.25
8
9 float startangle=0;
10 float lastangle=0;
11 float pendulumAngle;
12
13 unsigned long Ts = 3;
14 unsigned long k = 0;
15 bool nextStep = false;
16 bool realTimeViolation = false;
17
18 int i=i;
19 int T=1000;
20 float R
21 []={45.0,23.0,75.0,32.0,58.0,10.0,35.0,19.0,9.0,43.0,23.0,65.0,15.0,80.0}
22
23 float r=0.0;
24 float y = 0.0;
25 float u = 0.0;
26
27 void setup() {
28     Serial.begin(250000);
29     AeroShield.begin(AeroShield.detectMagnet());
30     startangle = AeroShield.calibration(AeroShield.getRawAngle());
31     lastangle=startangle+1024;
32     Sampling.period(Ts*1000);
33     PIDAbs.setKp(KP);
34     PIDAbs.setTi(TI);
35     PIDAbs.setTd(TD);
36     PIDAbs.setTs(Sampling.samplingPeriod);
37     Sampling.interrupt(stepEnable);
38 }
39
40 void loop() {
41     if(pendulumAngle>120){
42         AeroShield.actuatorWrite(0);
43         while(1);
44     }
45     if (nextStep) {
46         step();
47         nextStep = false;
48     }
49     void stepEnable() {
50         if(nextStep == true) {
51             realTimeViolation = true;
52             Serial.println("Real-time samples violated.");
53             analogWrite(5,0);
54             while(1);
55         }
56         nextStep = true;
57     }
58
59     void step() {
60         #if MANUAL
61         r = AeroShield.referenceRead();
62         #else
```

```

63     if (i > (sizeof(R) / sizeof(R[0]))) {
64         analogWrite(5, 0);
65         while (1);
66     } else if (k % (T*i) == 0) {
67         r = R[i];
68         i++;
69     }
70 #endif
71 y = AutomationShield.mapFloat(AeroShield.getRawAngle(),
72                               startangle, lastangle, 0.00, 100.00);
73 u = PIDAbs.compute(r-y, 0, 100, 0, 100);
74 AeroShield.actuatorWrite(u);
75 Serial.print(r);
76 Serial.print(" , ");
77 Serial.print(y);
78 Serial.print(" , ");
79 Serial.println(u);
80 k++;
81 }
```

Zdrojový kód 4.4: Zdrojový kód súboru AeroShieldPID.ino.

Zdrojový kód súboru AeroShield.m

```
1      classdef AeroShield < handle
2
3      properties
4          arduino;
5          as5600;
6      end
7      properties(Constant)
8          AERO_UPIN = 'D5';
9          AERO_RPIN = 'A3';
10         VOLTAGE_SENSOR_PIN = 'A2';
11         voltageReference = 5.0;
12         ShuntRes = 0.1;
13         correction1 = 4.1220;
14         correction2 = 0.33;
15         repeatTimes = 50;
16     end
17
18     methods
19         function begin(AeroShieldObject)
20             AeroShieldObject.arduino = arduino();
21             AeroShieldObject.as5600 = device(AeroShieldObject.arduino,
22                 'I2CAddress', 0x36);
23             configurePin(AeroShieldObject.arduino, AeroShieldObject.AERO_UPIN,
24                 'DigitalOutput')
25             disp('AeroShield initialized.')
26         end
27         function startangle = calibration(AeroShieldObject)
28             write(AeroShieldObject.as5600, 0x0c, 'uint8');
29             write(AeroShieldObject.as5600, 0xd, 'uint8');
30             startangle = read(AeroShieldObject.as5600, 1, 'uint16');
31         end
32         function PWM = referenceRead(AeroShieldObject)
33             PWM = readVoltage(AeroShieldObject.arduino, AeroShieldObject.
34                 AERO_RPIN);
35         end
36         function actuatorWrite(AeroShieldObject, PWM)
37             writePWMVoltage(AeroShieldObject.arduino, AeroShieldObject.
38                 AERO_UPIN, PWM);
39         end
40         function RAW = getRawAngle(AeroShieldObject)
41             write(AeroShieldObject.as5600, 0x0c, 'uint8');
42             write(AeroShieldObject.as5600, 0xd, 'uint8');
43             RAW = read(AeroShieldObject.as5600, 1, 'uint16');
44         end
45         function currentMean = getCurrent()
46             for r = 1:repeatTimes
47                 voltageValue = readVoltage(AeroShieldObject.arduino,
48                     AeroShieldObject.VOLTAGE_SENSOR_PIN);
49                 voltageValue = (voltageValue * voltageReference) / 1024;
50                 Current = Current + correction1 - (voltageValue / (10 * ShuntRes));
51             end
52             currentMean = Current / repeatTimes;
53             currentMean = currentMean - correction2;
54             if currentMean < 0.000
55                 currentMean = 0.000;
56             end
57             current = 0;
58             voltageValue = 0;
59         end
60     end
61 end
```

Zdrojový kód 4.5: Zdrojový kód súboru AeroShield.m.

Zdrojový kód súboru AeroShieldOpenLoop.m

```
1      clear all;
2      clear a;
3      clc
4
5      AeroShield=AeroShield;
6      AeroShield.begin();
7      startangle= AeroShield.calibration();
8      lastangle=startangle+2048;
9
10     time = 0;
11     count = 0;
12     angle = 0;
13     potentiometer = 0;
14
15     yyaxis right
16     plotGraph = plot(time,angle,'-r')
17     ylabel('Angle (degree)', 'FontSize',15);
18     xlabel('Time (s)', 'FontSize',15);
19     hold on
20
21     yyaxis left
22     plotGraph1 = plot(time,potentiometer,'-b')
23     title('Pendulum plot','FontSize',15);
24     ylabel('Percent','FontSize',15)
25     legend('Potentiometer value','Pendulum angle')
26     grid('on');
27
28     tic
29
30     while ishandle(plotGraph)
31     pwm = AeroShield.referenceRead();
32     AeroShield.actuatorWrite(pwm);
33
34     RAW= AeroShield.getRawAngle();
35     angle_ = mapped(RAW, startangle, lastangle, 0, 180);
36     count = count + 1;
37     time(count) = toc;
38     angle(count) = angle_(1);
39     percenta= mapped(pwm, 0.0, 5.0, 0.0, 100.0);
40     potentiometer(count) = percenta(1);
41     set(plotGraph,'XData',time,'YData',angle);
42     set(plotGraph1,'XData',time,'YData',potentiometer);
43     axis([time(count)-5 time(count) 0 100]);
44
45     if (angle_ > 110)
46     AeroShield.actuatorWrite(0.0);
47     disp('Angle of pendulum too high. AeroShield is turned off')
48     break
49   end
50 end
51
52 clear AeroShield.arduino;
```

Zdrojový kód 4.6: Zdrojový kód súboru AeroShieldOpenLoop.m.

Zdrojový kód súboru AeroShieldPID.m

```
1      clear all
2      clc
3      AeroShield=AeroShield;
4      PID = PID;
5      AeroShield.begin();
6      startangle= AeroShield.calibration();
7      lastangle=startangle+1024;
8      Ts = 0.0017;
9      Kp=0.015 ;
10     Ti=0.00020 ;
11     Td=0.0003 ;
12     PID.setParameters(Kp, Ti, Td, Ts);
13     MANUAL= 0;
14     R=[23 48 69 45 19 37 78];
15     secLength=30;
16     stepEnable = 0;
17     k=1;
18     j=1;
19     r=R(1);
20     y=0;
21
22     tic
23     while (1)
24         if (stepEnable)
25             RAW = AeroShield.getRawAngle();
26             y = map(RAW, startangle, lastangle, 0.0, 100.0);
27             if MANUAL
28                 PWMvalue = AeroShield.referenceRead();
29                 r=map(PWMvalue, 0, 5, 0, 100);
30             else
31                 if (mod(k, secLength*j)==0);
32                     j=j+1;
33                     if (j > length(R))
34                         AeroShield.actuatorWrite(0.0);
35                         break
36                     end
37                     r=R(j);
38                 end
39             end
40             u = PID.compute(r-y, 0, 90, 0, 90);
41             coercedInput = constrain(u, 0, 100);
42             PWM=map(coercedInput, 0, 100, 0, 5);
43             AeroShield.actuatorWrite(PWM);
44             PIDresponse(k,:)=[r y u];
45             plotLive(PIDresponse(k,:));
46             k=k+1;
47             stepEnable = 0;
48         end
49         if (toc>=Ts*k)
50             stepEnable = 1;
51         end
52         if (y > 110)
53             AeroShield.actuatorWrite(0.0);
54             disp('Angle of pendulum too high. AeroShield is turned off')
55             break
56         end
57     end
58
59     disp('Example finished. Captured data saved to "PIDresponse.mat"')
60     file.')
61     save PIDresponse PIDresponse
```

Zdrojový kód 4.7: Zdrojový kód súboru AeroShieldPID.m.