

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE**  
**Strojnícka fakulta**

Evidenčné číslo: SjF-13432-106440

**AeroShield: Miniatúrny experimentálny modul  
aerokyvadla**

**Bakalárska práca**

**2022**

**Peter Tibenský**



**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE**

**Strojnícka fakulta**

Evidenčné číslo: SjF-13432-106440

# **AeroShield: Miniatúrny experimentálny modul aerokyvadla**

**Bakalárska práca**

Študijný program: automatizácia a informatizácia strojov a procesov

Študijný odbor: kybernetika

Školiace pracovisko: Ústav automatizácie, merania a aplikovanej informatiky

Vedúci záverečnej práce: Ing. Mgr. Anna Vargová

Konzultant: Ing. Erik Mikuláš

**Bratislava 2022**

**Peter Tibenský**





## ZADANIE BAKALÁRSKEJ PRÁCE

Študent: **Peter Tibenský**  
ID študenta: 106440  
Študijný program: automatizácia a informatizácia strojov a procesov  
Študijný odbor: kybernetika  
Vedúca práce: Ing. Mgr. Anna Vargová  
Vedúci pracoviska: doc. Ing. Martin Halaj, PhD.  
Konzultant: Ing. Erik Mikuláš  
Miesto vypracovania: Ústav automatizácie, merania a aplikovanej informatiky (SjF)

Názov práce: **AeroShield: Miniatúrny experimentálny modul aerokyvadla**

Jazyk, v ktorom sa práca vypracuje: slovenský jazyk

Špecifikácia zadania:

Úlohou študenta je navrhnuť, 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,
- vytvorí inštruktážne príklady, ktoré ukazujú funkčnosť zariadenia,
- manaže verzie projektu v Git pre GitHub a piše úplnú dokumentáciu v MarkDown.

Rozsah práce: 30 – 50 strán

Termín odovzdania bakalárskej práce: 27. 05. 2022

Dátum schválenia zadania bakalárskej práce: 26. 05. 2022

Zadanie bakalárskej práce schválil: prof. Ing. Cyril Belavý, CSc. – garant študijného programu



## **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 autorské práva tretích osôb.

Bratislava, 27. máj 2022

.....  
Vlastnoručný podpis



V prvom rade by som sa chcel podakovať vedúcej mojej bakalárskej práce, Ing. Mgr. Anne Vargovej, za cenné rady pri písaní, nekonečné opravy tejto práce, za výborne kávičky v kabinete ako aj za najlepšie meme obrázky v skupine. Ďalej chcem podakovať 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 trpeživo počúvala moje nekonečné nadávky, stažnosti ale aj radosti pri tvorbe tejto práce. Zároveň sa chcem ospravedlniť spolužiakom za stres, ktorý mali z toho, keď som sa chvastal 40. stranou práce, zatiaľ čo podaktori ešte nepoznali tému ich bakalárskej práce.

Bratislava, 27. mája 2022

Peter Tibenský



**Názov práce:** AeroShield: Miniatúrny experimentálny modul aerokyvadla

**Klúčové slová:** Arduino, AutomationShield, PID, AeroShield, Aero pendulum, softvér, hardvér, Arduino IDE, MATLAB, Simulink

**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 hardvérová a softvérová časť. V rámci bakalárskej práce boli navrhnuté dva moduly, AeroShield R2 a R3, ktoré replikujú experiment známy pod názvom aeropendulum. V rámci hardvérovej časti bola zostavená doska plošných spojov, ako aj celkový model kyvadla. V softvérovej časti boli vytvorené didaktické príklady pre API Arduino IDE ako aj pre MATLAB a Simulink.

**Title:**AeroShield: Miniature experimental module of aeropendulum

**Keywords:** Arduino, AutomationShield, PID, AeroShield, Aero pendulum, software, hardware, Arduino IDE, MATLAB, Simulink

**Abstract:** The aim of this 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 engineering. Each module consists of hardware and a software part. As a part of this bachelor thesis, two modules were designed, the AeroShield R2 and R3. AeroShield was designed to replicate an experiment known as an aeropendulum. In the hardware part, the printed circuit board (PCB) was designed as well as the overall model of the pendulum. In the software part, didactic examples were created for the API Arduino IDE as well as for MATLAB and Simulink.



# Obsah

<b>Zoznam obrázkov</b>	i
<b>Zoznam zdrojového kódu</b>	iii
<b>Úvod</b>	1
<b>1 AeroShield</b>	5
1.1 Hardvér . . . . .	7
1.1.1 Popis súčiastok . . . . .	7
1.1.2 Schéma zapojenia . . . . .	11
1.1.3 Doska plošných spojov . . . . .	11
1.1.4 Model držiaku kyvadla . . . . .	13
1.1.5 Tretia verzia AeroShieldu . . . . .	15
1.2 Softvér . . . . .	17
1.2.1 Arduino API . . . . .	17
1.2.2 MATLAB . . . . .	25
1.2.3 Simulink . . . . .	27
<b>2 Didaktické príklady</b>	35
2.1 Programy v otvorenej slučke, bez spätej väzby . . . . .	35
2.1.1 Arduino IDE . . . . .	35
2.1.2 MATLAB . . . . .	37
2.1.3 Simulink . . . . .	38
<b>3 PID regulácia</b>	41
3.1 Programy v uzavorennej slučke, so spätnou väzbou . . . . .	43
3.1.1 Arduino IDE . . . . .	43
3.1.2 MATLAB . . . . .	49
3.1.3 Simulink . . . . .	51
<b>4 Záver</b>	53
<b>Literatúra</b>	55
<b>Arduino IDE</b>	v
Zdrojový kód AeroShield.h . . . . .	v
Zdrojový kód AeroShield.cpp . . . . .	vi

Zdrojový kód AeroShieldOpenLoop.ino . . . . .	viii
Zdrojový kód AeroShieldPID.ino . . . . .	ix
<b>MATLAB</b>	<b>xi</b>
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	Znižovací menič.	7
1.4	Zapojenie akčného člena a typ motorčeka.	8
1.5	Meranie prúdu.	9
1.6	Schematická reprezentácia Hallovho javu.	9
1.7	Enkóder slúžiaci na meranie uhla kyvadla.	10
1.8	Schéma zapojenia AeroShieldu.	11
1.9	Breakout board.	12
1.10	(a) Vrchná strana AeroShieldu. (b) Spodná strana AeroShieldu.	13
1.11	Dosky plošných spojov AeroShieldu.	14
1.12	AeroShield.	14
1.13	Tretia verzia AeroShieldu.	16
1.14	Schéma zapojenia tretej verzie AeroShieldu.	16
1.15	Knižnica AeroLibrary.	27
1.16	Reference read- Simulink.	28
1.17	Automatická trajektória- Simulink.	29
1.18	Angle read- Simulink.	30
1.19	Mapovanie uhlu kyvadla- Simulink.	31
1.20	Kalibrácia- Simulink.	31
1.21	Podsystém na kontrolu uhlu kyvadla.	32
1.22	Nastavenia parametrov masky bloku Reference read.	34
1.23	Reference read- Simulink.	34
2.1	Výstup z programu AeroShieldOpenLoop.ino.	36
2.2	Výstup z programu AeroShieldOpenLoop.m.	38
2.3	AeroShield_OpenLoop.	39
2.4	Výstup z programu AeroShieldOpenLoop.sim.	39
3.1	Schéma riadenia PID regulátorm.	41
3.2	Reakcia systému na skokovú zmenu referenčnej hodnoty- Arduino IDE.	47
3.3	Automatická trajektória.	47
3.4	Manuálna trajektória.	48

3.5	Reakcia systému na skokovú zmenu referenčnej hodnoty- MATLAB.	49
3.6	Automatická trajektória.	50
3.7	Manuálna trajektória.	51
3.8	Ukážka riadenia systému pomocou PID regulátora v API Simulink.	52
3.9	Reakcia systému na skokovú zmenu referenčnej hodnoty- Simulink.	52
3.10	Manuálna trajektória.	52

# Zoznam zdrojového kódu

1.1	Ukážka zdrojového kódu headeru.	17
1.2	Zdrojový kód funkcie mapFloat.	18
1.3	Zdrojový kód funkcie readOneByte.	19
1.4	Zdrojový kód funkcie readTwoBytes.	20
1.5	Zdrojový kód funkcie detectMagnet.	20
1.6	Zdrojový kód funkcie getMagnetStrength.	21
1.7	Zdrojový kód funkcie getRawAngle.	21
1.8	Zdrojový kód funkcie begin.	22
1.9	Zdrojový kód funkcie calibration.	23
1.10	Zdrojový kód funkcie convertRawAngleToDegrees.	23
1.11	Zdrojový kód funkcie referenceRead.	24
1.12	Zdrojový kód funkcie actuatorWrite.	24
1.13	Zdrojový kód funkcie currentMeasure.	24
1.14	Knižnica AeroShield.m properties.	25
1.15	Knižnica AeroShield.m properties.	26
1.16	MATLAB function blok autiomatická trajektoria.	28
1.17	Mapovacia funkcia vo fcn bloku.	30
1.18	Callback funkcia.	32
1.19	Initialization- maska Reference read.	33
2.1	AeroShield open loop dekleracia.	35
2.2	AeroShield open loop setup().	35
2.3	AeroShield open loop loop().	36
2.4	AeroShield open loop inicializacia.	37
2.5	AeroShield open loop, while cyklus.	37
3.1	Načítanie knižníc a premenných do programu.	43
3.2	Organizačná funkcia setup.	44
3.3	Funkcia stepEnable().	45
3.4	Organizačná funkcia loop.	45
3.5	Funkcia step().	46
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

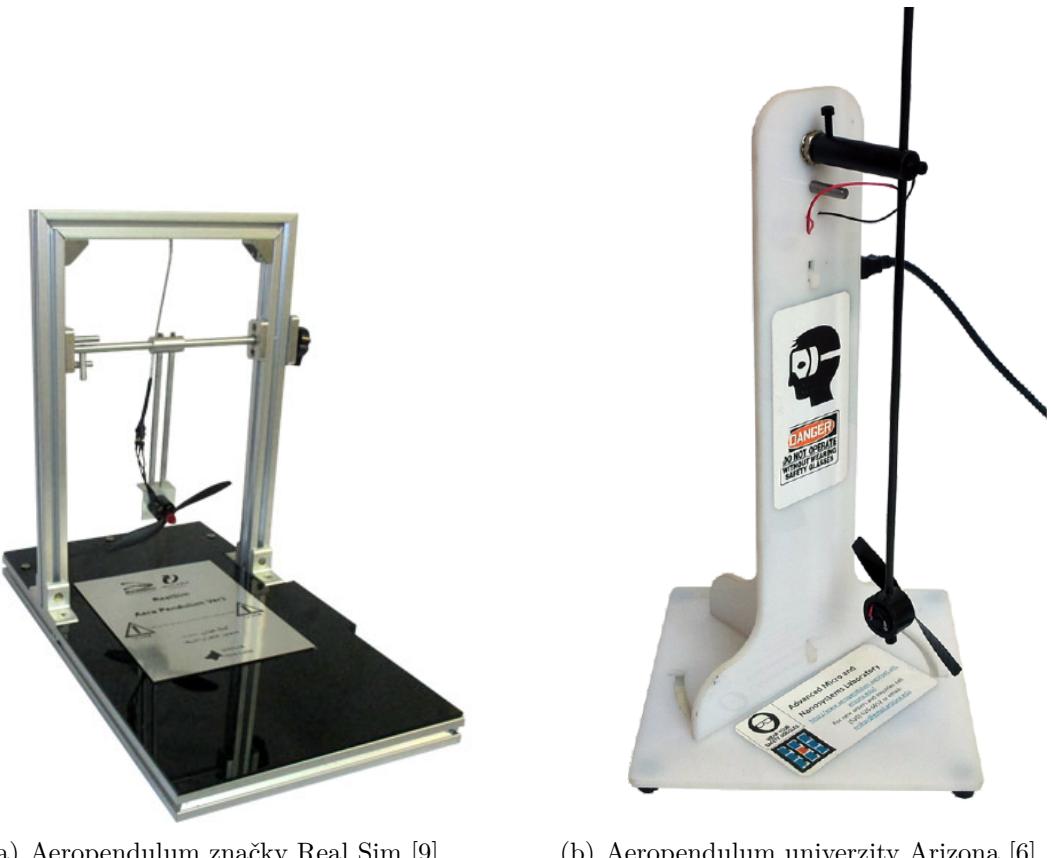
Cieľom tejto práce je návrh, výroba a naprogramovanie modernej učebnej pomôcky AeroShieldu (ďalej len „shield“), ktorá slúži na výučbu základov teórie riadenia a elektrotechniky. Učebné pomôcky sú nevyhnutnou, no často zanedbávanou súčasťou výučby. Študenti si vďaka nim môžu lepšie predstaviť a pochopiť 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 privelične drahé [3]. Z týchto dôvodov je ich použitie pri výučbe 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 tah. Motorček je zvyčajne upevnený na koniec ľahkej tyčky, ktorá je v mieste otáčania pripevnená k zariadeniu na meranie uhlu pootočenia. Takýmto zariadením môže byť potenciometer, senzor hallovho javu(efektu), alebo iné [4]. Zariadenie na meranie uhlu je upevnené na podstavec, ktorý kyvadlo stabilizuje a umožňuje jeho voľný pohyb.

Tvorba AeroShieldu bola inšpirovaná experimentom známym pod názvom Aerpendulum, čo v doslovnom preklade znamená vzdušné kyvadlo. Na túto tému existuje niekoľko publikácií, ktoré sa zaobrajú zostavením, ovládaním, alebo simuláciami takého 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, 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 zaobrali vylepšením Arizonského kyvadla, lepším pohonom, presnejším ovládaním, rôznymi meraniami polohy a zrýchlenia ako aj zmenou ovládacieho modulu za mikrokontrolér Arduino.

Medzi komerčne dostupné riešenia tohto experimentu patrí napríklad kyvadlo značky Real Sim Obr. 1.a, ktorá ponúka hotový, zostavený modul. Ďalším takýmto modulom je kyvadlo od univerzity Arizona [6] Obr. 1.b, ktoré je predávané ako nezostavený model.



(a) Aeropendulum značky Real Sim [9].

(b) Aeropendulum univerzity Arizona [6].

Obr. 1: Experimentálne moduly vzdušného kyvadla.

Open-source<sup>1</sup> projekt AutomationShield je zameraný na vývoj hardvérových a softvérových nástrojov, určených na vzdelávanie a doplnenie vzdelávacieho procesu. Jadrom celého projektu je tvorba rozširujúcich dosiek (shieldov) vyvájaných pre populárny typ prototypizačných dosiek s mikrokontrolérmi Arduino. Tieto pomerne lacné učebné pomôcky majú za cieľ skvalitniť výučbu strojného inžinierstva, mechatroniky a základov automatického riadenia [10].

Všetky informácie ohľadom projektu AutomationShield, sú dostupné na open-source 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.

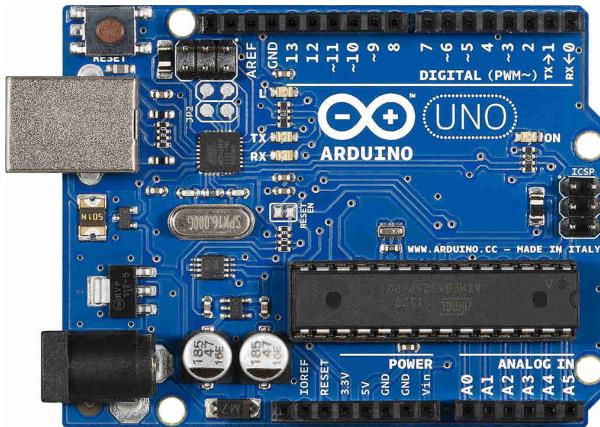
Hlavnou motiváciou tohto projektu je nízka dostupnosť a vysoká cena podobných učebných pomôcok. Z môjho pohľadu je výučba častokrát až príliš zameraná na memorovanie faktov a teórie, namiesto praktických experimentov a skúseností typu pokus-omyl. Študenti pochopia vyučovanú teóriu jednoduchšie, pokial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

V roku 2005 prišla na trh prototypizačná doska Arduino. Projekt vznikal v Talian-

---

<sup>1</sup>Open-source je zo všeobecného pohľadu akákolvek informácia, ktorá je dostupná verejnosti bez poplatku(s voľným prístupom), s ohľadom na fakt, že jej volné šírenie zostane zachované.

sku ako kolaborácia medzi viacerými nadšencami elekrotechniky a programovania, na ktorej čele bol Massimo Banzi. Veľkou výhodou dosiek Arduino a ich nadstavbových shieldov je fakt, že sú pomerne lacné a majú malé rozmery (Arduino UNO:  $68.6 \times 53.4$  mm [13]). Tieto skutočnosti umožňujú študentom pracovať na experimentoch nielen na pôde školy, ale experimenty si môžu zobrať aj domov.



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 komponentov a fakt, že mikročip Arduina je v prípade poruchy jednoducho vymeniteľný<sup>2</sup>, 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 najmä dve verzie Arduina. Prvou z nich je doska Arduino UNO R3 Obr. 2. Na doske sa nachádza 14 digitálnych a 6 analógových pinov.

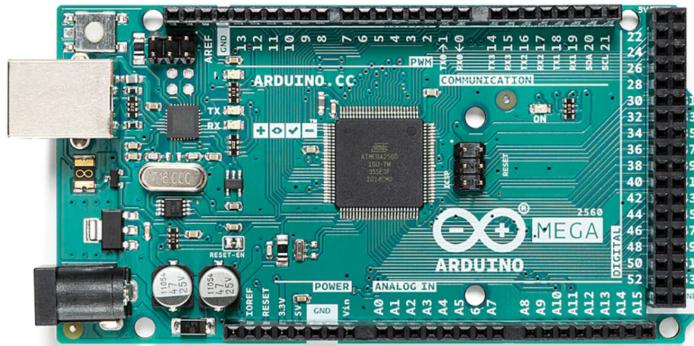
Na prácu v MATLABE a Simulinku využívame najmä Arduino Mega 2560 R3 Obr. 3. Na tejto doske sa nachádza 54 digitálnych a 16 analógových pinov. AeroShield je kompatibilný so všetkými doskami s označením R3 alebo s doskami, ktoré majú rozloženie pinov rovnaké ako Arduino UNO R3.

Niektoré piny sú označené špeciálnym symbolom „~“. Tieto piny sú schopné generovať PWM<sup>3</sup> signál, ktorý využívame na ovládanie motora kyvadla.

---

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

<sup>3</sup>Šírková 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].

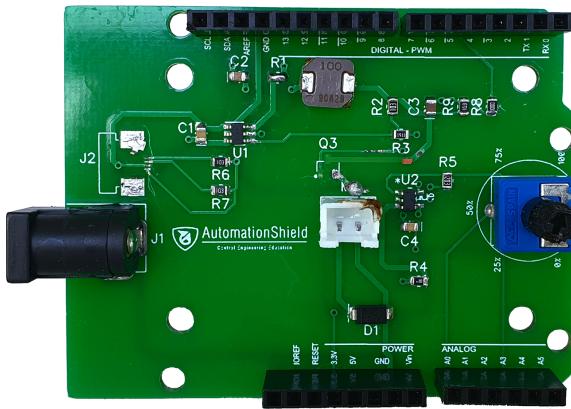
Práca je rozdelená na štyri logické celky. Na začiatku v časti hardvér je opísaný základný princíp fungovania shieldu a jeho jednotlivé súčiastky. V tejto časti sa taktiež opisuje tvorba schémy zapojenia a dosky plošných spojov AeroShieldu.

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

Predposlednú časť práce tvoria samotné didaktické príklady, za ktorými nasleduje záverečná časť, ktorou je finálne zhodnotenie práce.

# 1 AeroShield

Práca je založená na už započatom projekte vzdušného kyvadla. Na jeho tvorbe sa najviac podieľali študenti: Ján Boldocký, Denis Skokan, Dávid Vereš a Tadeáš Vojtko. 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.



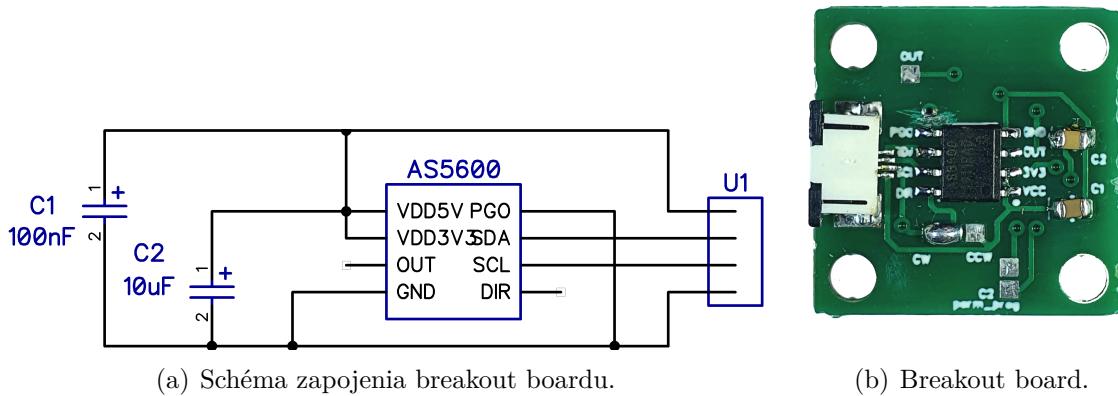
Obr. 1.1: Prvá verzia AeroShieldu.

V novej verzii dosky bolo odstránených niekoľko nedostatkov predchádzajúcej verzie:

- neprepojenie pinov komunikácie I<sub>2</sub>C tj. piny SDA a SCL senzoru hall efektu, ktorý slúži na meranie uhlu 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, ich úprava do funkčnej podoby a následne zostavenie modelu. 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 prepojenie senzoru hall efektu s hlavnou doskou shieldu. Doska breakout boardu fungovala bezproblémovo, a teda nebolo potrebné nijakým spôsobom meniť jej schému zapojenia Obr. 1.2.a. Breakout boardu sa budeme bližšie venovať v Kap. 1.1.3.



Obr. 1.2: Meranie uhla kyvadla.

## 1.1 Hardvér

### 1.1.1 Popis súčiastok

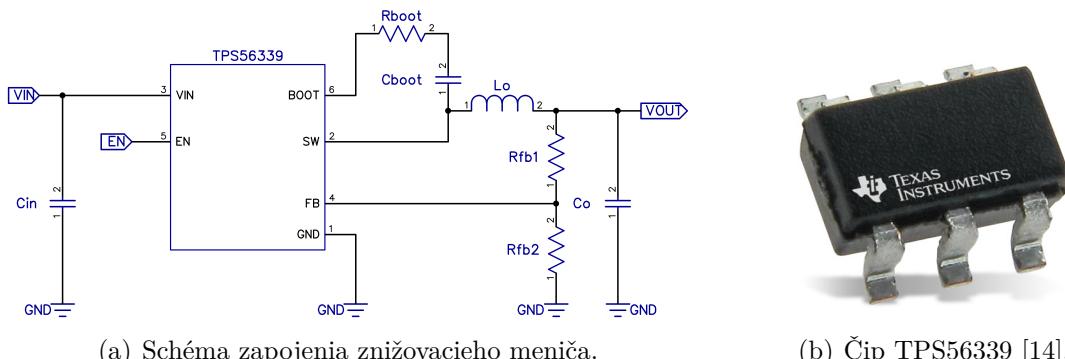
V tejto časti sa bližšie pozrieme na menšie celky zapojenia AeroShieldu:

- napájanie,
- meranie prúdu,
- ovládanie akčného člena,
- meranie uhlu kyvadla.

#### Znižovací menič

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

Hlavnou časťou konvertora je čip TPS56339 od výrobcu Texas Instruments Obr. 1.3.b. Znižovanie napäťia funguje za pomoci dvoch integrovaných N-kanálových  $70\text{ m}\Omega$  a  $35\text{ m}\Omega$  high-side mosfetov<sup>4</sup>, v spolupráci s ďalšími komponentami. Celkový prevádzkový prúd zariadenia je pribлизne  $98\text{ }\mu\text{A}$ , keď funguje bez spínania a bez zátaže. Keď je zariadenie vypnuté, napájací prúd je pribлизne  $3\text{ }\mu\text{A}$  a zariadenie umožňuje nepretržitý výstupný prúd do 3 A [14].



Obr. 1.3: Znižovací menič.

Na čip je privádzané napäťie 12 V, ktoré sa pomocou zapojenia viditeľného na schéme Obr. 1.3.a, znižuje na napäťie 3,7 V. 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 sice nachádza aj na doske Arduino UNO a pomocou VIN pinu sa z neho dajú napájať napäťím 6-12 V aj iné zariadenia, avšak tento pin je napojený na diódu, obmedzujúcu prúd na 1 A [15][16].

Pri testovaní prototypov AeroShieldu, sa merané hodnoty prúdu pohybovali nad hodnotu 1 A, čo by zničilo spomínanú internú diódu na doske Arduino UNO. Po

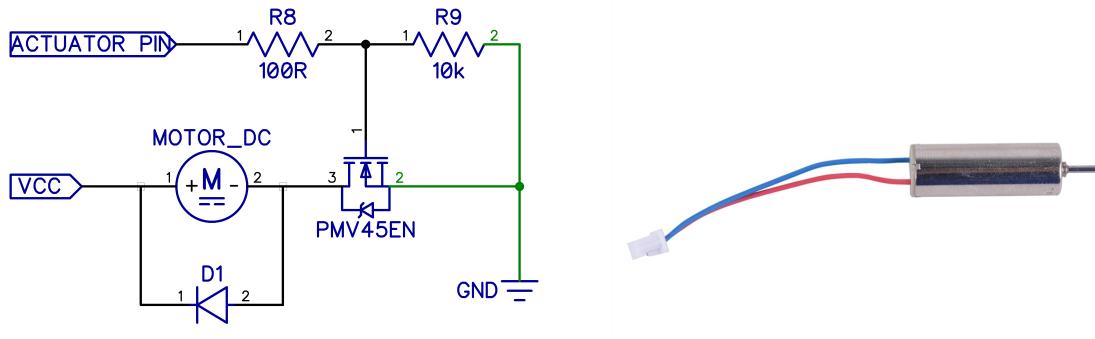
<sup>4</sup>N-kanálový mosfet je typ tranzistora, v ktorom sa ovláda kolektorový prúd pomocou napäťia medzi riadiacou elektródou a emitorom.

zostavení druhej verzie AeroShieldu a opäťovnom meraní odoberaného prúdu, bola maximálna dosahovaná hodnota menšia ako 0,5 A. Z tohto dôvodu sme sa rozhodli pre zmenu v schéme kyvadla, z ktorej sme odobrali externý napájací konektor a následne bola vyrobená nová doska plošných spojov. Tretej verzii AeroShieldu sa bližšie venujeme v Kap. 1.1.5.

### Akčný člen

Ako akčný člen AeroShieldu je použitý 7 mm, 3,7 V motorček na jednosmerný prúd bez jadra, používaný hlavne pre pohon dronov. Motor bez jadra (coreless motor), je motor s cievkou navinutou samou na sebe [17]. Stator je vyrobený z magnetov na báze vzácných zemín, ako je neodým alebo SmCo(samárium-kobalt).

Takýto motor ponúka niekoľko výhod. Výrazne sa znižuje hmotnosť a tým aj zotrvačnosť rotora, čo je dôležité pre naše použitie, kedy potrebujeme dosahovať vysokú akceleráciu motora. Ďalšou výhodou je fakt, že nedochádza k stratám na železe a tým pádom sa účinnosť takýchto motorov blíži až ku 90% [18]. Motor, resp. otáčky motora sú riadené pomocou PWM signálu a ten do motoru prechádza cez N-kanálový mosfet PMV45EN2 od výrobcu Nexperia [19].



(a) Schéma zapojenia motorčeka.

(b) Akčný člen [20].

Obr. 1.4: Zapojenie akčného člena a typ motorčeka.

### Meranie prúdu

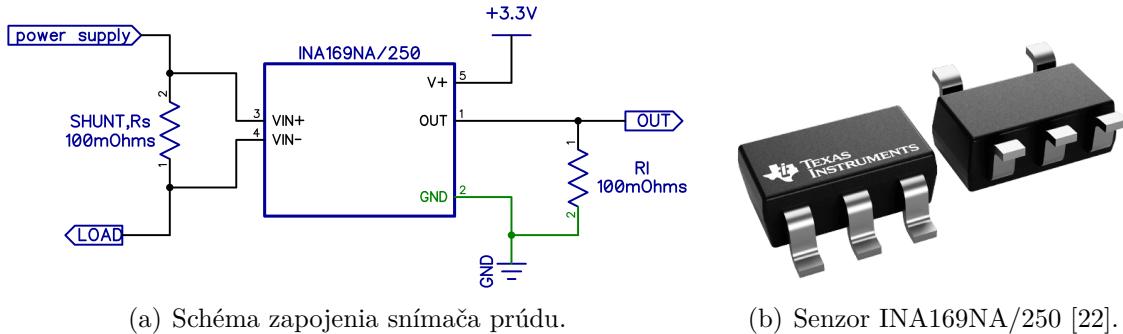
Z dôvodu merania prúdu odoberaného motorom bol do schémy pridaný monitor prúdu, takzvaný „current shunt monitor”. Nameraný prúd môžeme využiť na implementáciu riadenia motora na základe prúdu, ktorý odoberá. AeroShield používa snímač INA169NA/250 od výrobcu Texas Instruments Obr. 1.5.b.

INA169 funguje na základe zaznamenávania zmien napäcia na stranách shunt rezistora Obr. 1.5.a. Na základe nameraného úbytku napäcia vysiela senzor podľa nami zvoleného stupňa zosilnenia, napätie s maximálnou hodnotou  $V_{OUTMAX} = V_{IN} - 0.5 \text{ V}$ .

Prúd  $I_s$  odoberaný motorom, vypočítame pomocou vzorca 1.1

$$I_s = \frac{(V_{OUT} \times 1 \text{ k}\Omega)}{(R_s \times R_l)} \quad (1.1)$$

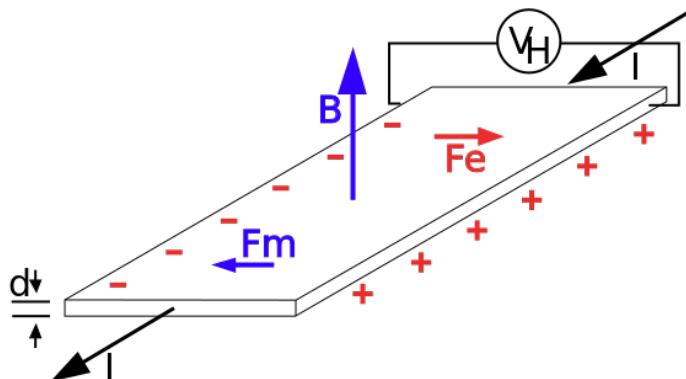
kde  $V_{OUT}$  je napäťie namerané na výstupe,  $1 \text{ k}\Omega$  je konštantou vnútorných odporov senzoru,  $R_s$  je hodnota bočníka v  $\Omega$  a  $R_l$  je hodnota rezistora na výstupe, taktiež v  $\Omega$  [21].



Obr. 1.5: Meranie prúdu.

### Meranie uhla kyvadla

Na fungovanie AeroShieldu je dôležité vedieť s vysokou presnosťou merať uhol náklonenia kyvadla. Na tento účel sme si zvolili meranie uhlu bezkontaktnou formou, pomocou snímača Hallovho javu. Hallov jav vieme opísť ako vznik priečného elektrického poľa v pevnom materiáli, keď ním preteká elektrický prúd a tento materiál je umiestnený v magnetickom poli, ktoré je kolmé na prúd [23]. 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  $V_H(V)$  je Hallovo napätie,  $B(T)$  je magnetické pole,  $F_m(N)$  magnetická sila pôsobiaca na negatívne prenášače náboja,  $F_e(C)$  elektrická sila z nahromadeného náboja,  $I(A)$  je dohodnutý smer elektrického prúdu.

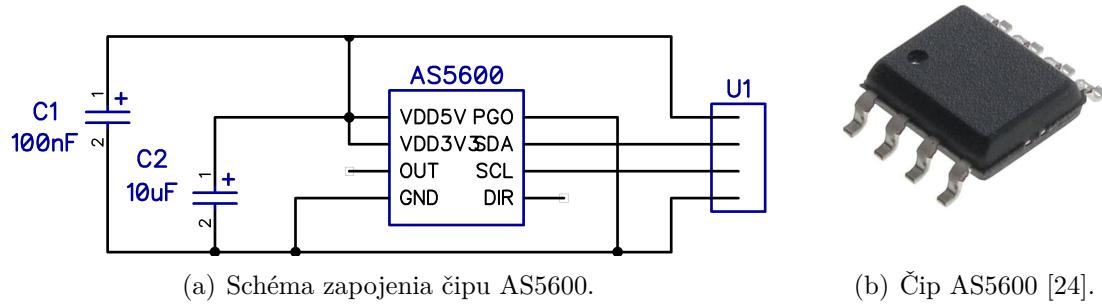


Obr. 1.6: Schematická reprezentácia Hallovho javu.

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 zachytené senzorom sa v čipe najskôr 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, 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 člena sústavy. Schéma zapojenia enkóderu na meranie uhla môžeme vidieť na Obr. 1.7.a.



Obr. 1.7: Enkóder slúžiaci na meranie uhla kyvadla.

### 1.1.2 Schéma zapojenia

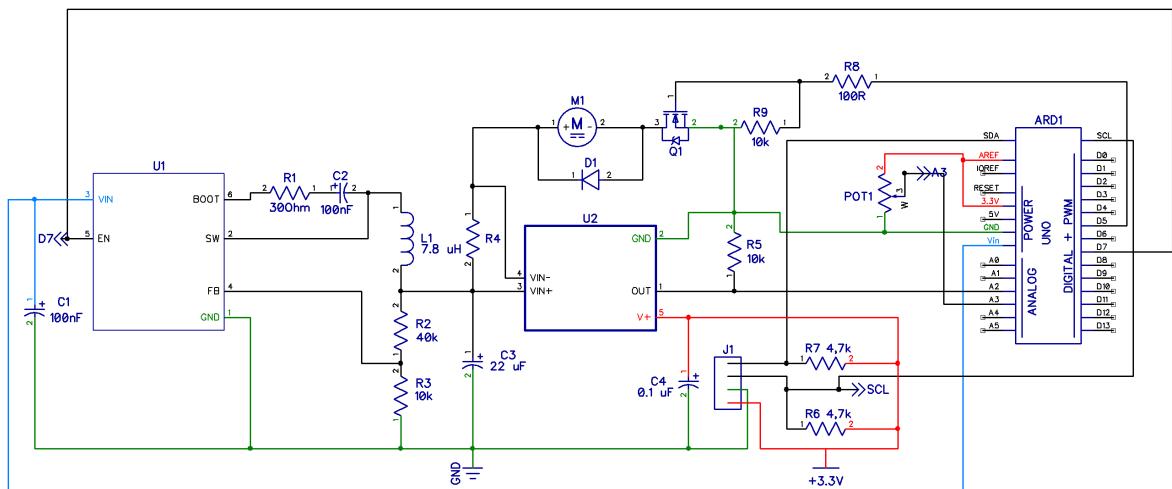
Všetky schémy zapojenia boli tvorené v bezplatnej verzii programu DipTrace, ktorý slúži ako prostredie na tvorbu elektrotechnických schém a dosiek plošných spojov.

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 [25][26][22]. Do programu bola taktiež vložená knižnica AutomationShieldu, ktorá obsahuje najčastejšie používané komponenty. Jednotlivé komponenty majú podobu elektrotechnických značiek s danými vlastnosťami.

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

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 zemniace spojenie- zelená, fázové spojenia 3,3 V- červená. Na schéme zapojenia sú použité nasledujúce komponenty:

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



Obr. 1.8: Schéma zapojenia AeroShieldu.

### 1.1.3 Doska plošných spojov

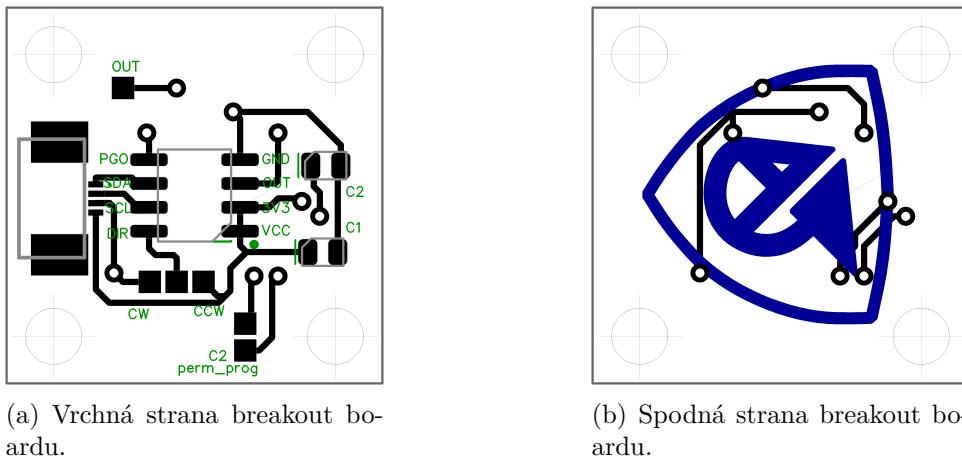
Po návrhu a kontrole schém zapojenia sa schémy ďalej spracovávajú do podoby dosky plošných spojov. Schémy exportujeme do programu DipTrace PCB v ktorom máme

následne niekoľko možností postupu. Jednotlivé komponenty sa nám už zobrazujú v reálnej podobe, takže vidíme ich veľkosť a rozmiestnenie konektorov. Program ponúka možnosť automatického alebo manuálneho rozmiestnenia komponentov.

Dosky plošných spojov majú niekoľko výhod, ale aj negatív oproti ponúkaným alternatívam [27].

Výhodou je fakt, že vodivé spojenia medzi jednotlivými súčiastkami 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 [28]. Tým že vodivé cesty môžu mať veľmi malé rozmery, ovplyvňujúcim faktorom veľkosti dosky plošných spojov je samotná veľkosť použitých komponentov.

Nevýhodami sú napríklad vyššia cena, ako pri spojení pomocou káblov, zložitý proces výroby, ktorý predlžuje čakaciu dobu na hotový produkt. Ďalšou z nevýhod je zložitá, alebo nemožná oprava chyby na hotovej doske, ktorá nastala v procese tvorby schémy zapojenia .



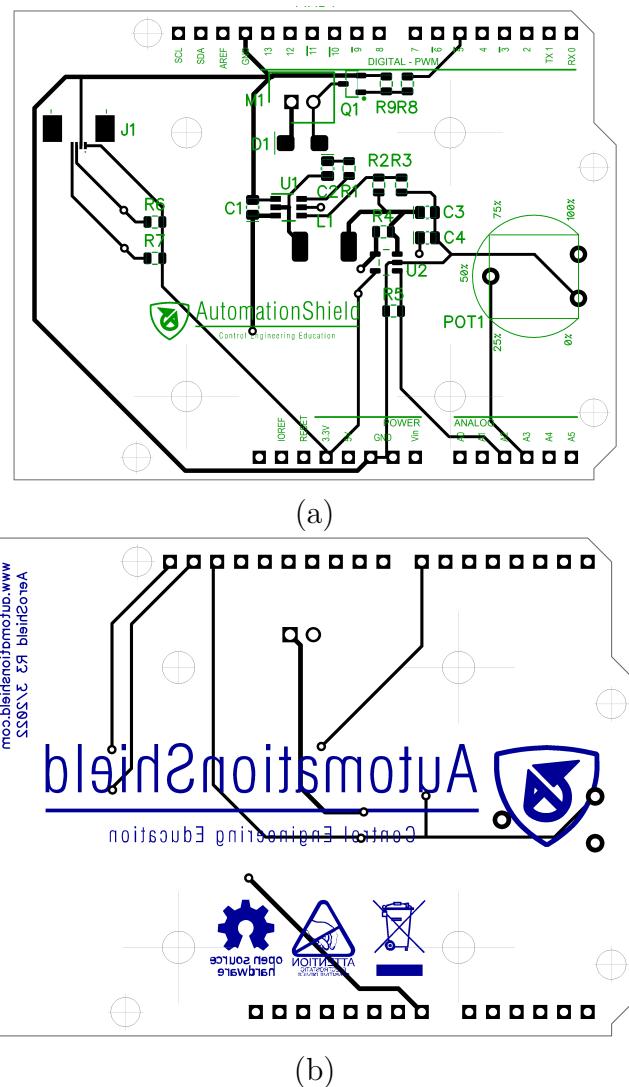
Obr. 1.9: Breakout board.

Po rozmiestnení komponentov treba jednotlivé piny poprepájať vodivými cestami, ktoré nahradzajú funkciu kálov. Máme možnosť zvoliť automatické alebo manuálne rozmiestnenie ciest. Ako je vidieť na Obr. 1.10.a, nie všetky cesty majú rovnakú šírku. Dôvodom je fakt, že niektorými cestami prúdi väčší prúd. V zásade sa používa pravidlo, čím vyšší prúd preteká vodičom, tým väčšiu plochu prierezu by mal mať. Prúd pretekajúci vodičom tento vodič zahrieva. Pokial 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. Pokial by k takému križeniu 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 konektory „via”, spájajúce obe strany dosky.

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 zvolať parametre dosky ako jej farbu,

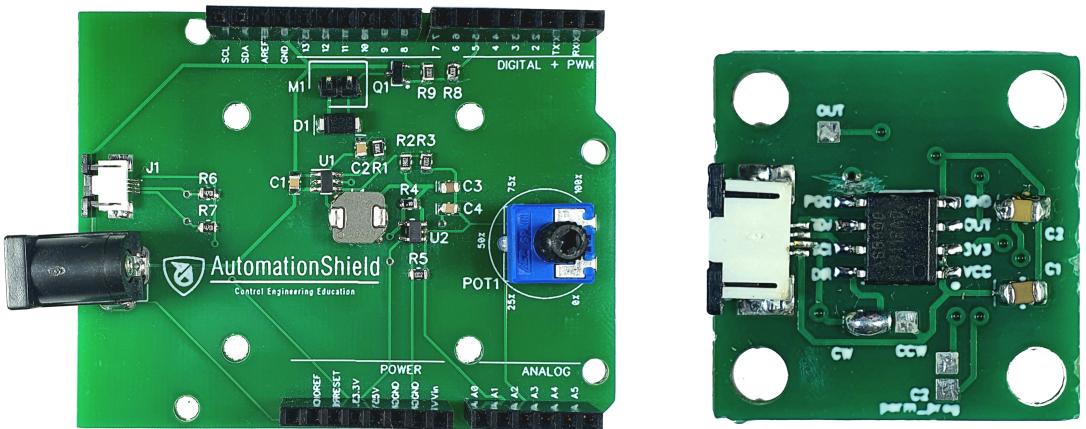
typy spájkovacích doštičiek a iné. Podobu finálnej dosky tretej verzie AeroShieldu môžeme vidieť na Obr. 1.11.a a dosky breakout boardu na Obr. 1.11.b.



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

#### 1.1.4 Model držiaku kyvadla

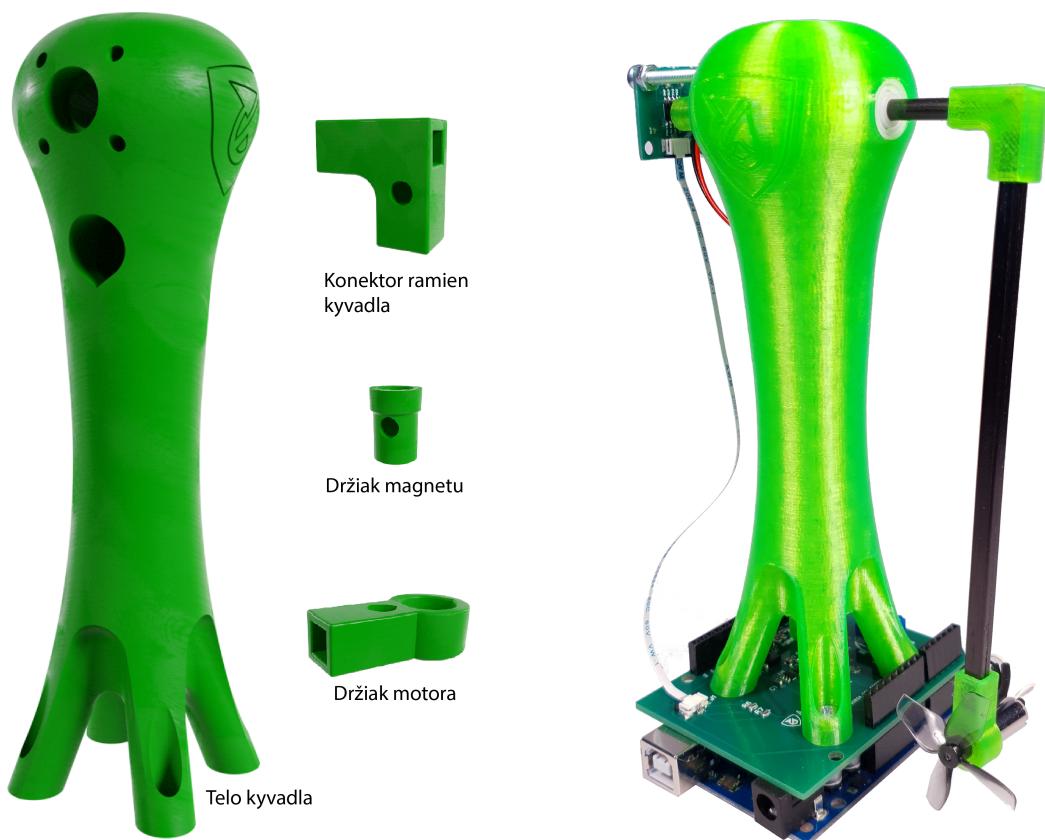
Telo kyvadla ako aj všetky konektory spájajúce jeho jednotlivé časti, boli vytvorené v 3D modelovacom softvéri CATIA Obr. 1.12. Zámerom bolo vytvoriť pevný a zároveň estetický držiak, ktorý sa dá následne vytlačiť na 3D tlačiarni. Telo kyvadla stojí na štyroch nožičkách, ktoré sú priskrutkované k hlavnej 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 (druhá verzia).

(b) Vedľajšia doska AeroShieldu.

Obr. 1.11: Dosky plošných spojov AeroShieldu.



(a) Model tvorený v programe CATIA.

(b) Zostavený model AeroShieldu.

Obr. 1.12: AeroShield.

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 7x20 mm 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
<b>Spolu</b>				<b>25,95€</b>

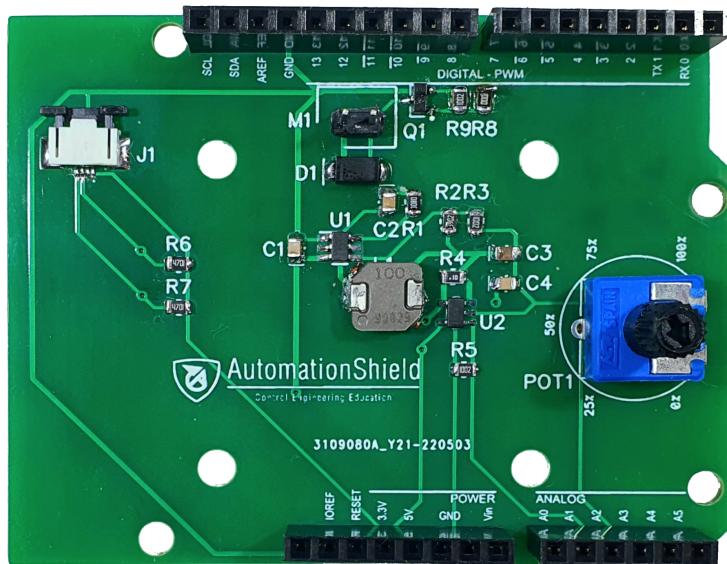
Tabuľka 1.1: Cenová kalkulácia AeroShieldu.

### Cenová kalkulácia AeroShieldu

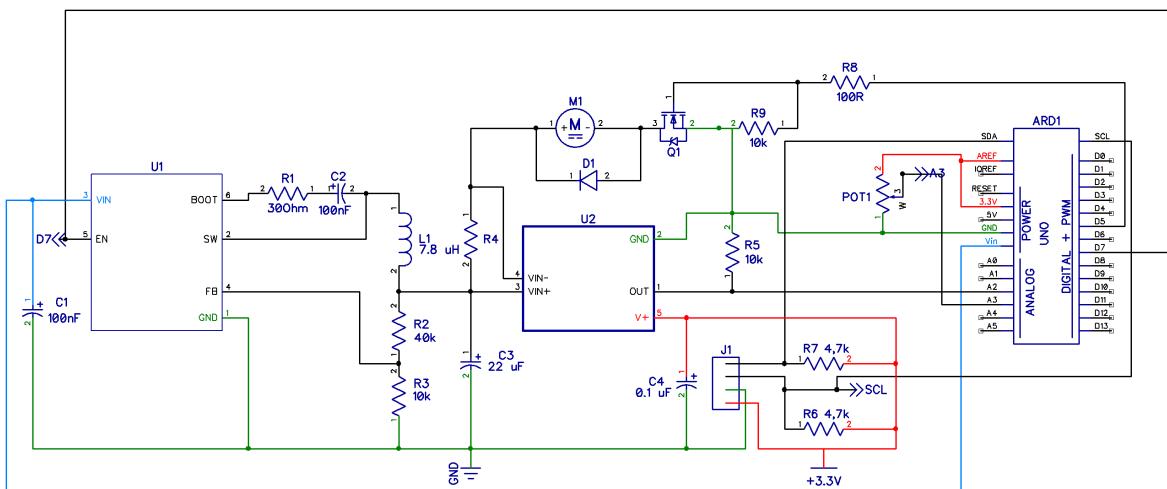
Hlavnou podmienku pri tvorbe AeroShieldu bola jeho funkciu, dostupnosť použitých komponentov, jednoduchosť vyhotovenia, ako aj nízka cena zostaveného modelu. Za účelom predstavy ceny jedného kusu AeroShieldu, bola zostavená Tab. 1.1 s použitými komponentami, ich počtom a reálnou kúpnou cenou v eurách(spolu s DPH). Pri komponentoch ako sú rezistory a kapacity, 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 viacerých kusov.

#### 1.1.5 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, R9 a mosfet Q1. Ako posledná úprava bola otočená polarita potenciometra, ktorý bol na druhej verzii 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

Prvé vývojové prostredie, ktoré sme zvolili na programovanie arduina sa nazýva Arduino IDE<sup>5</sup> a využíva programovací jazyk C/C++ resp. jeho mutáciu, s pridanými špecializovanými príkazmi a funkciami. V rámci projektu AutomationShield existuje niekoľko rôznych Shieldov, ktoré však často využívajú rovnaké funkcie a príkazy. Z tohto dôvodu bola vytvorená knižnica „AutomationShield”, ktorá v sebe zahŕňa niekoľko ďalších, často využívaných knižníc a súborov. Ide o funkcie potrebné pri riadení pomocou PID regulátora, vzorkovanie programu alebo mapovanie premenných. O jednotlivých funkciách, v rámci knižnice, si povieme viacej pri opise didaktických príkladov v Kap. 2.

V rámci programovania knižníc využívame objektovo orientované programovanie (OOP) [29]. Knižnice väčšinou rozdeľujeme na dva súbory. 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 [30].

#### 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 definoval niekolkokrát, spôsobovalo by to problém pri komplikácii kódu. Z toho dôvodu využívame príkazy **#ifndef** spolu s **#define**, ktoré zamedzujú násobnému definovaniu rovnakých súborov.

Hneď za definovaním knižnice AeroShield.h môžeme do súboru vkladať ďalšie potrebné knižnice a to pomocou príkazu **#include**. Následne určujeme premenné, ktoré môžu mať priradené fyzické čísla pinov na Arduine.

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

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

#define AERO_RPIN A3             // Vstup z potenciometra
#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.

---

<sup>5</sup>Arduino Integrated Development Environment.

V časti **Zdrojovy kod**, vytvárame **triedu** (class), ktorá v sebe zahŕňa funkcie a premenné, ktoré sa nazývajú **objekty** (objects). Trieda 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 nachádzat jedna, alebo viacero tried, záleží to od logicky deliteľných úsekov kódu, alebo od preferencie programátoru.

Rozdelenie na public a privat má zmysel 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

Súbory **header** a **source** prepojíme príkazom `#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 [31], najviac však v AeroShielde využívame typy **void**, **float** a **bool**.

## Popis použitých funkcií z knižnice AutomationShield

Pri rôznych velkostiah 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 docieliли vyššiu presnosť, potrebujeme mapovať dátový typ **float**. Na tento účel nám slúži funkcia **mapFloat**.

```
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.

## Popis použitých funkcií z knižnice AeroShield

Kedže na AeroShielde využívame senzor hall efektu, musíme s ním v prvom rade nadviazať komunikáciu a to pomocou sériovej komunikácie I<sup>2</sup>C.

Táto komunikácia funguje na princípe master-slave. Master môže naraz komunikať s viacerými zariadeniami a to na základe jedinečných adres zariadení, ktoré sa medzi sebou striedajú v komunikácii.

Protokol I<sup>2</sup>C 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 impulsy.

Hodinový pin udáva tempo komunikácie a je ovládaný mestrom. 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 najsú obsahujú adresu zariadenia slave, s ktorým chce mestor komunikovať, následne sa odosielajú bity príkazov. Keď sa táto informácia celá odošle, slave vykoná požiadavku a ak je to vyžadované, môže späť mestrovi poslať údaje. Všetky tieto bity informácií sa posielajú na linke SDA [32].

### 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_adr )
{
    int retVal = -1;           // Zadefinovanie pomocnej premennej
    Wire.beginTransmission(_ams5600_Address); // Zaciatoč komunikacie
    Wire.write(in_adr);       // 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 1.3: Zdrojový kód funkcie `readOneByte()`.

Ako môžeme vidieť v kóde funkcie, mestor najskôr 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 odpoveď 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 posun<sup>6</sup>.

---

<sup>6</sup>Bitový posun je operácia vykonávaná so všetkými bitmi binárnej hodnoty, pri ktorej sa posúvajú o určený počet miest doľava alebo doprava [33].

```

word AeroShield :: readTwoBytes( int in_adr_hi , int in_adr_lo )
{
    word retVal = -1;                      // Zadefinovanie pomocnej
    premennej
    /* citanie "Low" bajtu */
    Wire.beginTransmission(_ams5600_Address); // Zaciatak komunikacie
    Wire.write(in_adr);           // Poziadavka na zaznamenianie 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

    int low = Wire.read();                // Ulozenie prveho bajtu
    /* citanie "High" bajtu */
    Wire.beginTransmission(_ams5600_Address); // Zaciatak komunikacie
    Wire.write(in_adr);           // Poziadavka na zaznamenianie 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

    word high = Wire.read();              // Ulozenie druheho bajtu

    high = high << 8;                  // Posun bitov
    retVal = high | low;

    return retVal;                      // Zaslanie odpovede
}

```

Zdrojový kód 1.4: Zdrojový kód funkcie readTwoBytes.

### Funkcia detectMagnet()

Ďalšou dôležitou funkciou, je zistíť prítomnosť magnetu na kyvadle. Túto úlohu vykonáva funkcia `bool AeroShield :: detectMagnet()`. Využívame ju vždy pri inicializácii AeroShieldu na to, aby sme zistili či nenastali problémy s magnetom, alebo so senzorom samotným. Na základe výstupu z funkcie vieme určiť či bol magnet detegovaný. Funkcia vráti na výstupe 1- pokial sa magnet nachádza pri senzore a 0- pokial magnet nie je zachytený.

```

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

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

    return retVal;                  // Zaslanie odpovede
}

```

Zdrojový kód 1.5: Zdrojový kód funkcie detectMagnet.

## Funkcia getMagnetStrength()

Pre správnosť fungovania hall senzoru je dôležité dodržať predpísanú vzdialenosť magnetu od senzoru. Výrobca udáva že najideálnejšia vzdialenosť je 0.5-3 mm, 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 1.6: Zdrojový kód funkcie `getMagnetStrength ()`.

## Funkcia getRawAngle()

Táto funkcia slúži na čítanie uhlu kyvadla. Výstupom tejto funkcie, je číslo s rozsahom 12 bitov, 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, ktory rovno vrati vysledok pomocou  
                                                    // prikazu return  
}
```

Zdrojový kód 1.7: Zdrojový kód funkcie `getRawAngle ()`.

## Funkcia begin()

Prvou z funkcií mimo komunikácie s rotačným enkóderom je `void AeroShield :: begin (void)`, v ktorej sa ako prvé zapíše výstup 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 I<sup>2</sup>C. Príkaz na započatie komunikácie I<sup>2</sup>C 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 I<sup>2</sup>C. 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<sup>7</sup> cyklus, sa pomocou príkazu `break` ukončí. Pokiaľ magnet detegovaný neboli, vypíše „Can not detect magnet”, no while cyklus pokračuje.

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

    #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
        detegovaný
        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 detegovaný
            AutomationShield.serialPrint("Can not detect magnet \n");
        }
    }
}
```

Zdrojový kód 1.8: Zdrojový kód funkcie `begin`.

## Funkcia `calibration()`

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

Do funkcie vstupuje hodnota aktuálneho uhla 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

---

<sup>7</sup>Cyklus while sa bude opakovat nepretržite, pokiaľ sa výraz vnútri zátvoriek () nestane nepravdivým.

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 1.9: Zdrojový kód funkcie `calibration`.

### Funkcia `convertRawAngleToDegrees()`

Ako sme už spomínali v Kap. 1.1.1, zaznamenávaná hodnota uhlu kyvadla je v rozmedzí od 0 do 4096 a tieto hodnoty sú priamo úmerné so stupňami od  $0^\circ$  do  $360^\circ$ .  $1^\circ$  predstavuje hodnotu približne 11.77 vo formáte raw. Funkcia teda prenásobí raw hodnotu číslom  $0.087^\circ$ , ktoré sme získali z Rov. 1.2 a predstavuje velkosť rozlíšenia rotačného enkóderu. Výstupom funkcie je uhol kyvadla v stupňoch.

$$\frac{360^\circ}{4096} = 0.087^\circ \quad (1.2)$$

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

Zdrojový kód 1.10: 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 rozsahu 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 1.11: Zdrojový kód funkcie referenceRead.

### Funkcia actuatorWrite()

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

```

void AeroShield :: actuatorWrite(float motorPercent) {
    float mappedValue = AutomationShield.mapFloat(
        motorPercent, 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 1.12: Zdrojový kód funkcie actuatorWrite.

### Funkcia currentMeasure()

Poslednou funkciou AeroShieldu je currentMeasure(), ktorá slúži na zaznamenávanie prúdu, ktorý odoberá motor kyvadla. 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 prerušované vypínaný a zapínaný, kolísá 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 1.13: 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 ide 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** [34].

Každá trieda môže následne obsahovať jeden alebo viacero triednych blokov. V knižnici **AeroShield** 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 využívat. 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 1.14: Knižnica AeroShield.m properties.

Nasleduje triedy blok `methods`, v ktorom sú definované všetky funkcie. Keďže logika funkcií je rovnaká ako v Kap. 1.2.1, nebudeme si funkcie vysvetľovať po jednom. Na ukážku si opíšeme časť kódu `function begin(AeroShieldObject)`, v ktorej prebieha tvorba objektov pre arduino a hall senzor. Príkaz `arduino()` slúži na vyhľadanie dosky arduino, pripojenej do počítača a nastavenie parametrov potrebných na komunikáciu s doskou. Rovnako sa odvolávame aj na objekt `as5600`, ktorý vytvoríme pomocou príkazu `device()`, v ktorom zadávame objekt dosky arduino, a adresu zariadenia I<sup>2</sup>C. 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 1.15: Knižnica AeroShield.m properties.

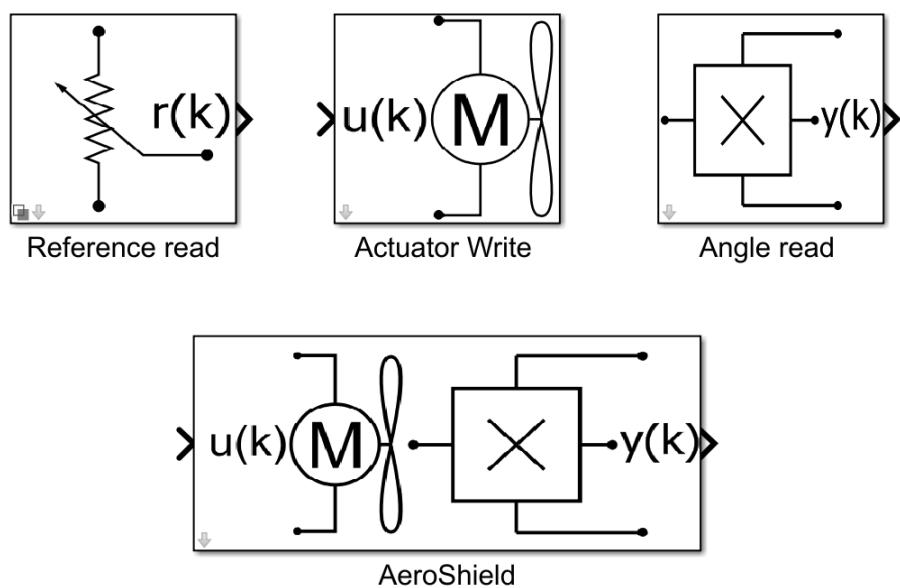
Zostávajúce funkcie, ako aj celý zdrojový kód knižnice `AeroShield.m` sa nachádza v prílohe 4.

### 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 vytvorená 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 poviem v Kap. 1.2.3.



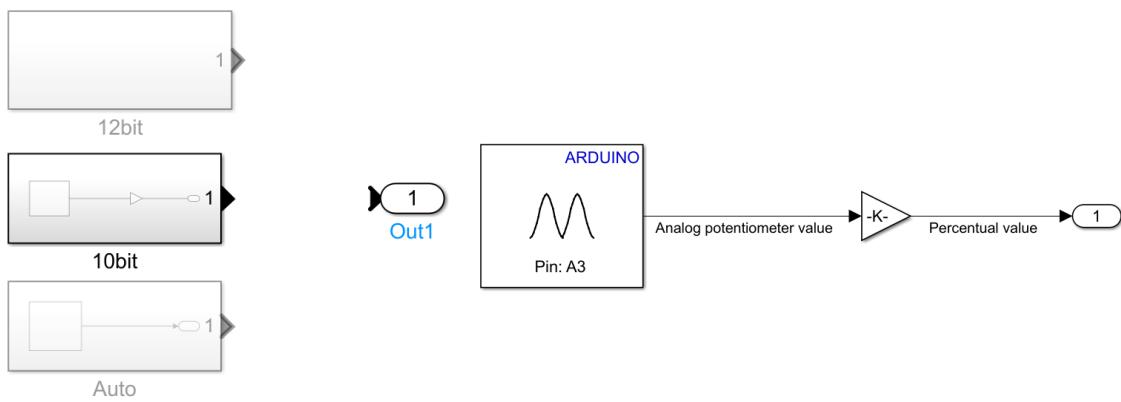
Obr. 1.15: Knižnica AeroLibrary.

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 Arduina<sup>8</sup> je pri použití bloku **Analog input**, výstup v tvare 12-bitového čísla, narozdiel 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 Kap. 1.2.3 na strane 32.



Obr. 1.16: Reference read- Simulink.

V MATLAB **function** bloku, ktorý sa nachádza v podsystéme pre automatickú trajektóriu, nájdeme kód 1.16. 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. Pokial je úsek trajektórie spustený dlhšie ako zadávame v maske bloku, funkcia automaticky prejde na ďalší úsek. Pokial 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
```

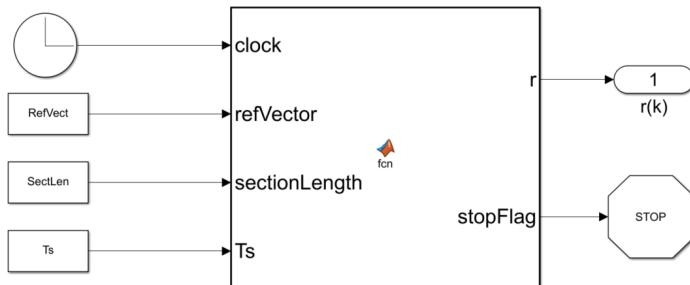
<sup>8</sup>Due, MKR1000, MKR1010, MKRZero, Nano 33IoT.

```

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 1.16: MATLAB function blok automatická trajektoria.

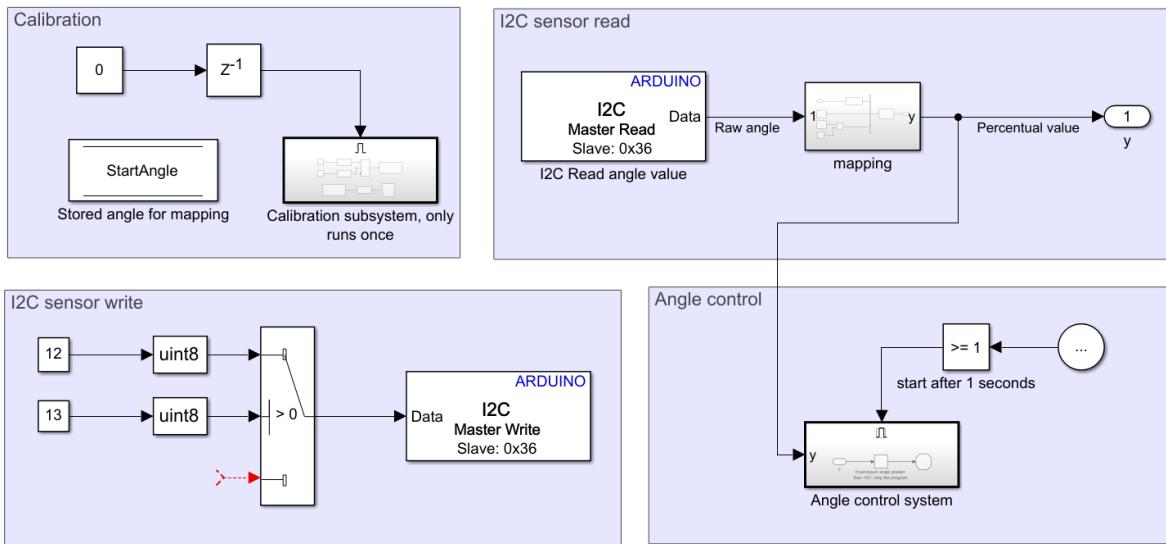


Obr. 1.17: Automatická trajektória- Simulink.

### Angle read

Blok `angle read` Obr. 1.18 slúži na čítanie uhlu kyvadla, ako aj pre jeho bezpečnostnú kontrolu. Vnútorná štruktúra tohto bloku 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  $360^\circ$ . Tento prevod sa vykonáva v mapovacom podsystéme Obr. 1.19. Mapovanie funguje na rovnakom princípe ako v príklade 2.1.2. Jednotlivé premenné vchádzajú do bloku MUX, ktorý slúži ako multiplexer<sup>9</sup>. Signál z multiplexoru vstupuje do fcn bloku, ktorý prepája Simulink s MATLABOM. Zdrojový kód 1.17 zobrazuje funkciu vo vnútri tohto bloku.

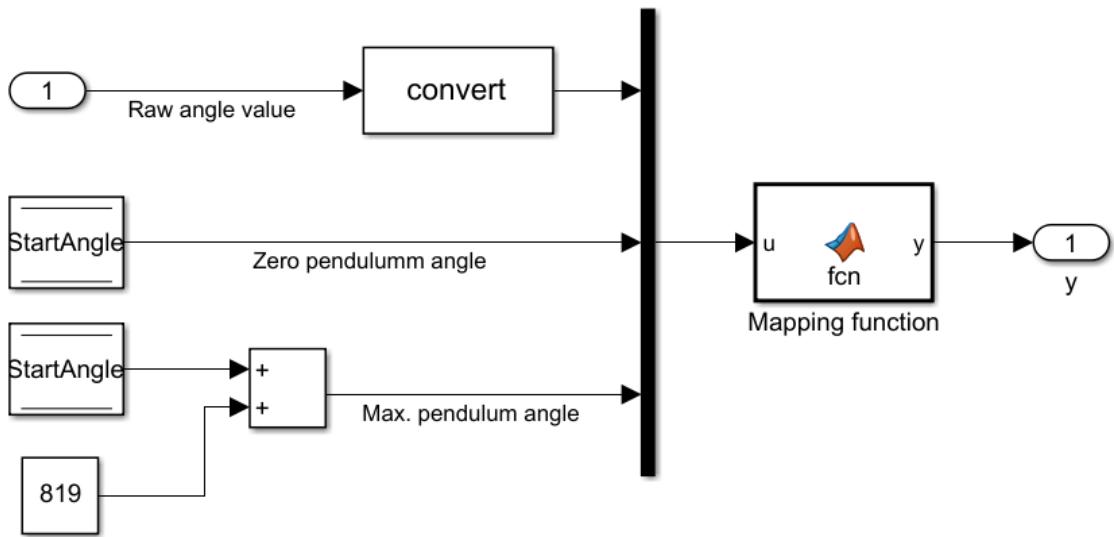
```
function y = fcn(u)
angMin=0;
angMax=72;
y = ((u(1) - u(2)) * (angMax - angMin)) /(u(3)- u(2)) + angMin ;
```

Zdrojový kód 1.17: Mapovacia funkcia vo fcn bloku.

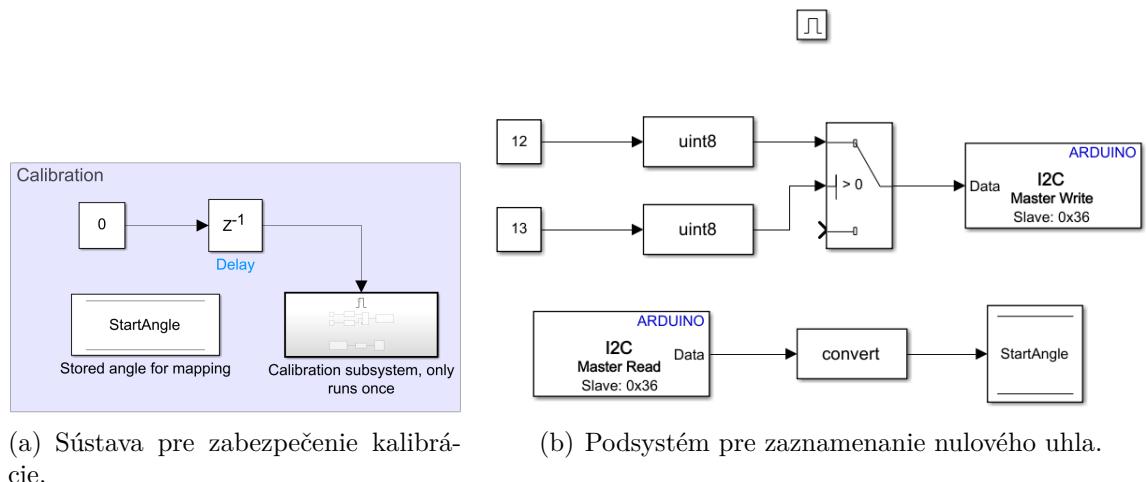
Proces získania premennej **StartAngle**, ktorú využívame pri mapovaní, je opísaný v nasledujúcom odstavci.

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

<sup>9</sup>Zariadenie, ktoré vyberá medzi viacerými analógovými alebo digitálnymi vstupnými signálmi a tieto preposielá na jednu výstupnú linku.

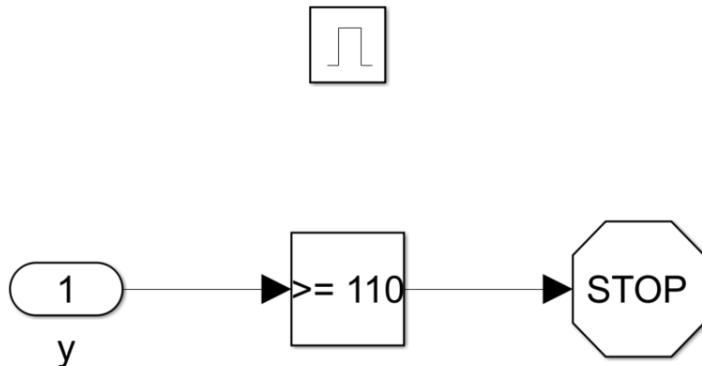


Obr. 1.19: Mapovanie uhlu kyvadla- Simulink.



Obr. 1.20: Kalibrácia- Simulink.

**Kontrola uhlu** Kontrola uhlu kyvadla prebieha v podsystéme, ktorý je spustený jednu sekundu po spustení Obr. 1.18 (pravý dolný roh). Časový posun je použitý z dôvodu občasného šumu premenných pri spustení príkladu. 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  $110^\circ$ . Pokial 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 zapúzdríme blokovú schému tak, aby mala podľa potreby, vlastné dialógové okná s nastaviteľnými parametrami, popisy bloku, výzvy na zadanie parametrov alebo texty s návodami. Vďaka maske je používanie a editácia blokov intuitívnejšia a vďaka textovým popisom funkcií jednoduchšia. 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 tou je model dosky Arduino. Ide o roletové, alebo „popup“ menu, v ktorom si vyberieme typ dosky s ktorou používame AeroShield. Dôvod tohto výberu je opísaný v Kap. 1.2.3, na strane 28. 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 1.18, 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 1.18: 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 písmeňo z premennej `ref`, s písmeňom 'M' (Manual). Pokiaľ sa tieto zhodujú, príkaz `set_param`, spolu s príkazom `MaskVisibilities`, nastaví polia tlačidiel ako viditeľné, teda s hodnotou 'on'. Pokiaľ sa písmeňa nezhodujú a teda bola vybraná trajektória automatická, možnosť výberu dosky Arduino, je skrytá príkazom 'off' a namiesto nej, sa zobrazia ďalšie štyri editovateľné polia.

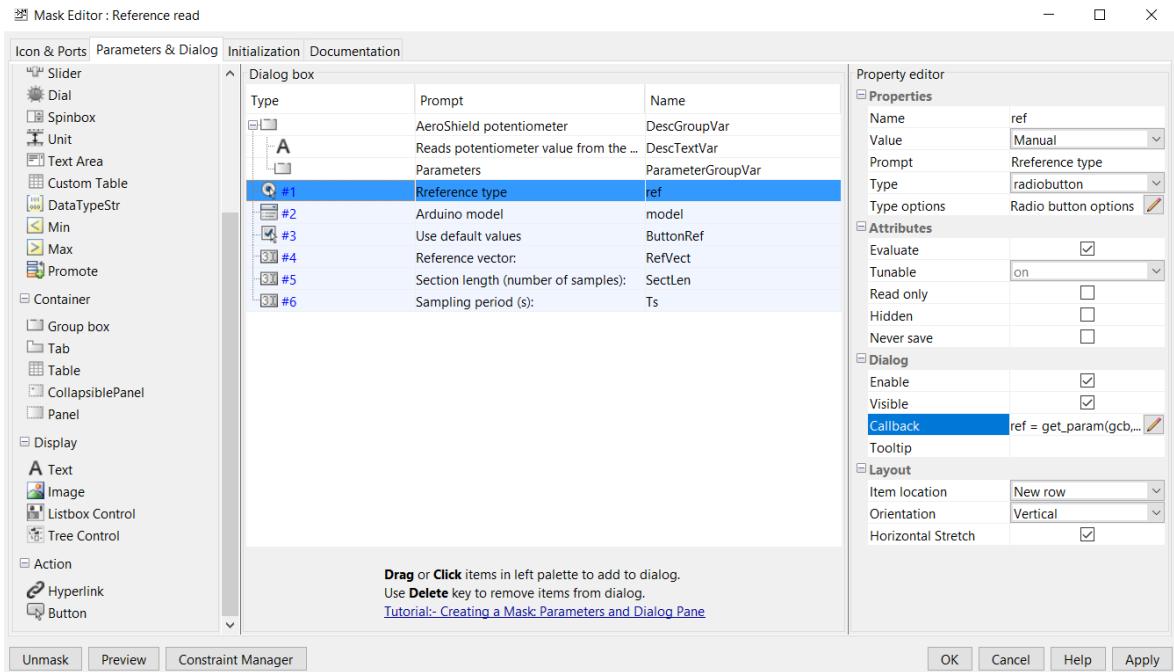
`Initialization` tvorí tretiu záložku v maske. V tejto záložke môžeme pridať MATLAB kód, ktorý ovláda a formátuje podsystémy. V maske bloku `Reference read`, máme v tejto záložke vložený kód 1.19. Podľa pravdivosti `if` podmienky, príkaz `set_param(gcb, 'LabelModeActiveChoice')` aktivuje podsystém bloku, ktorého názov je uvedený ako tretí parameter v zátvorke príkazu.

```

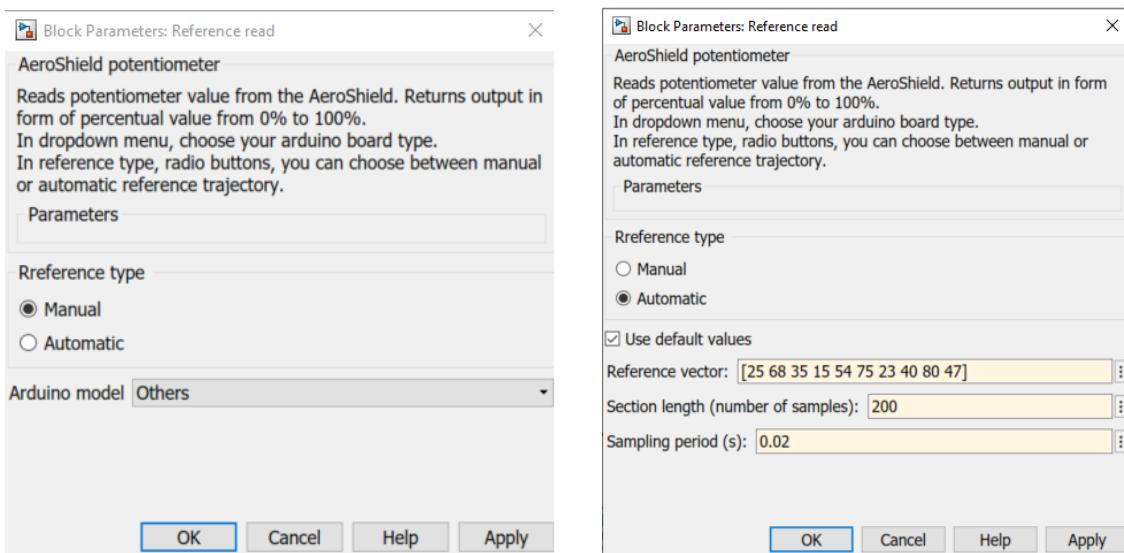
ref = get_param(gcb, 'ref');
model = get_param(gcb, 'model');
if strcmp(ref, 'Manual')
    if strcmp(model, 'Due')
        set_param(gcb, 'LabelModeActiveChoice', '12bit');
    elseif strcmp(model, 'MKR1000')
        set_param(gcb, 'LabelModeActiveChoice', '12bit');
    elseif strcmp(model, 'MKR1010')
        set_param(gcb, 'LabelModeActiveChoice', '12bit');
    elseif strcmp(model, 'MKRZero')
        set_param(gcb, 'LabelModeActiveChoice', '12bit');
    elseif strcmp(model, 'Nano 33IoT')
        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 1.19: Initialization- maska Reference read.



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.

## 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 spätej väzby a programy v uzavretej slučke so spätnou väzbou.

Ich rozdiel spočíva v tom, že pri riadení bez spätej väzby, hovoríme o ovládaní systému, kedy sa snažíme dosiahnuť žiadané hodnoty výstupov bez spätej 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 spätej 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.

V časti `setup()` 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 rýchlosťi nazývame **baud rate**. Nasleduje funkcia `AeroShield.begin()`, ktorá sleduje prítomnosť magnetu, a prednastaví 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 uhlu kyvadla
                                pre map function
}

```

Zdrojový kód 2.2: AeroShield open loop setup().

V cykle `loop()` prebehne ako prvé mapovanie uhlu kyvadla pomocou funkcie `AutomationShield.mapFloat()`. 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.

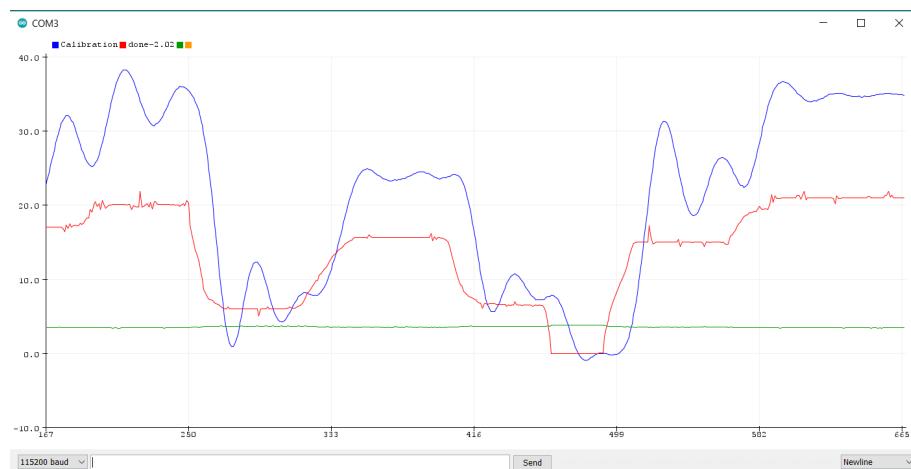
```

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.

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.

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()`. Pokračujeme čítaním uhlu kyvadla `AeroShield.getRawAngle()`, za ktorým prebehne mapovanie premennej z hodnoty raw na stupne. Premenná `count` slúži na počítanie počtu prejdených cyklov, na tvorbu usporiadanej 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  $110^\circ$ , proces automaticky ukončí a vypíše upozornenie. Posledný príkaz `clear AeroShield.arduino` vymaze objekt `arduino` a pripraví MATLAB na spustenie ďalšieho programu.

```
while ishandle(plotGraph)          % slucka bezi pokial sa
    nezatvorí graf

    pwm = AeroShield.referenceRead();    % citanie hodnoty
                                         % potenciometra
    AeroShield.actuatorWrite(pwm);       % zapis na aktuator
    RAW= AeroShield.getRawAngle();        % citanie raw uhlu
    angle_ = mapped(RAW, startangle, lastangle, 0, 180); % mapovanie
                                         % raw uhol na stupne
```

```

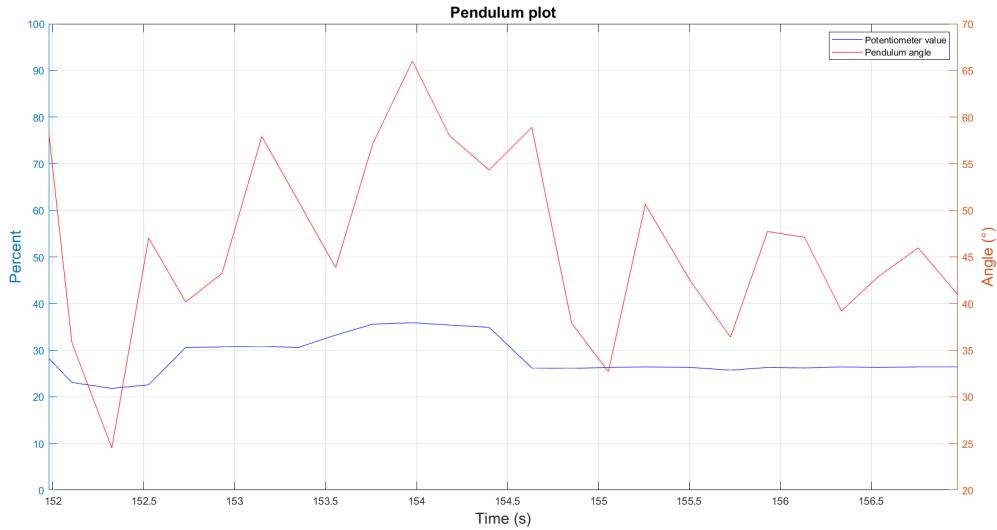
count = count + 1;                                % zaznamenavanie
    poctu cyklov
time(count) = toc;                               % zaznamenavanie
    casu
angle(count) = angle_(1);                         % hodnota uholu v
    case
percenta= mapped(pwm, 0.0, 5.0, 0.0, 100.0);    % mapovanie pwm
    na percenta
potentiometer(count) = percenta(1);              % hodnota
    potenciometra v case
set(plotGraph , 'XData' , time , 'YData' , angle);    % vykresli prve
    data
set(plotGraph1 , 'XData' , time , 'YData' , potentiometer); % vykresli
    druhe data
axis([time(count)-5 time(count) 0 100]);          % beziaca x-ova
    osa

if (angle_ > 110)                                 % ak uhol kyvadla
    vacsi ako 110 stupnov
AeroShield.actuatorWrite(0.0);                   % zastav motor
disp('Angle of pendulum too high. AeroShield is turned off') % zastav program
break
end
end

clear AeroShield.arduino;

```

Zdrojový kód 2.5: AeroShield open loop, while cyklus.

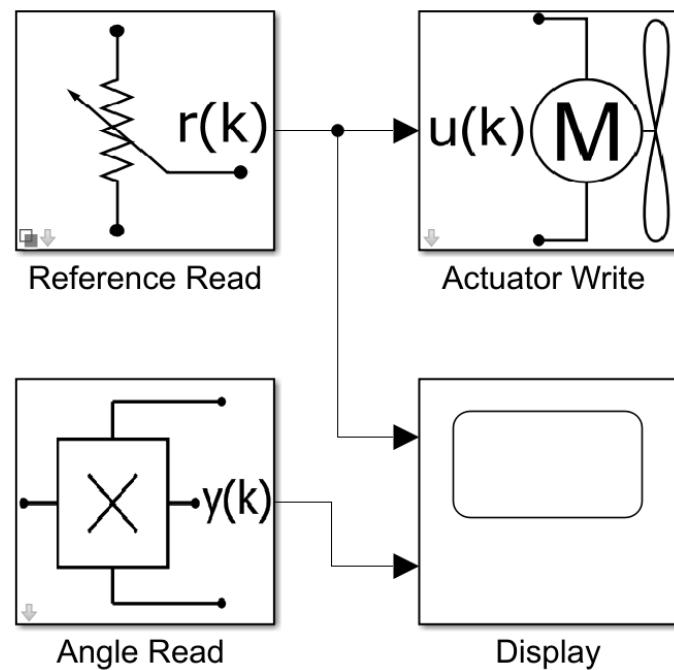


Obr. 2.2: Výstup z programu AeroShieldOpenLoop.m.

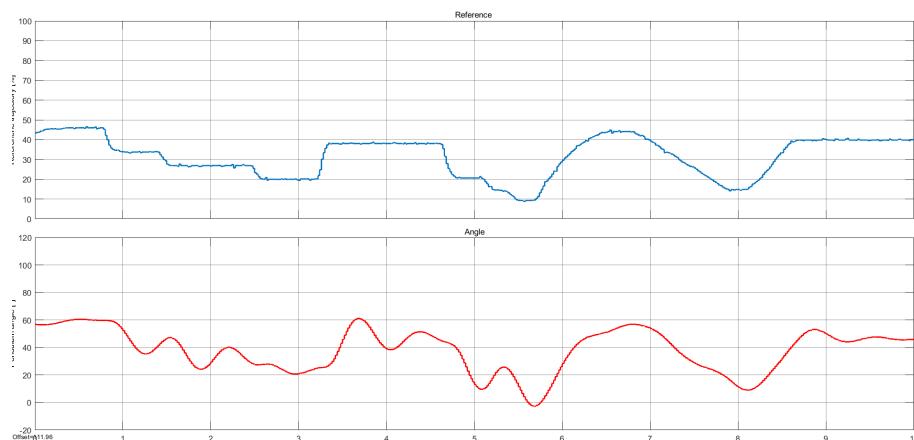
### 2.1.3 Simulink

Na ukážku funkcií jednotlivých blokov AeroLibrary, bol v API Simulink zostavený inštruktážny príklad AeroShieldOpenLoop Obr. 2.3. V tomto príklade sa pomocou

hodnoty potenciometra ovláda akčný člen. Zároveň je meraný uhol, v ktorom sa kyvadlo nachádza a obe tieto hodnoty sú priebežne zobrazované na grafe pomocou bloku Scope Obr. 2.4.



Obr. 2.3: AeroShield\_OpenLoop.



Obr. 2.4: Výstup z programu AeroShieldOpenLoop.sim.

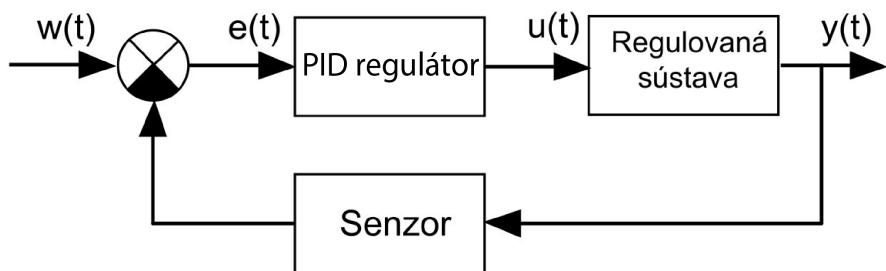


### 3 PID regulácia

PID reguláciu sme si vybrali z dôvodu jednoduchosti použitia, ako aj z dôvodu že práve PID regulácia je vyučovaná medzi prvými v rámci teórie riadenia. P, PI alebo PID regulátory taktiež patria medzi veľmi populárne typy riadiacich algoritmov.

Skôr ako si ukážeme príklady s využitím PID regulátora, musíme si vysvetliť ako takýto 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 ovplyvňuje 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: Lineárne riadenie s premenlivým parametrom (LPV), Lineárno-kvadratické riadenie (LQ), Modelové prediktívne riadenie (MPC), 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 je zložená zo začiatočných písmen zložiek, z ktorých sa regulátor skladá.

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

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

Konšanta  $K_p$  je veľmi dôležitou pri nastavovaní parametrov PID regulátora. Zvyšovaním hodnoty proporcionálnej zložky znižujeme regulačnú odchýlku, avšak nikdy nedosiahneme úplne odstránenie trvalej regulačnej odchýlky. Zmenou proporcionálnej zložky vieme taktiež urýchliť, alebo spomalíť nábeh na požadovanú hodnotu. Pre malé hodnoty  $K_p$  je nábeh pomalý, a regulačná odchýlka veľká. Pri zvyšovaní hodnoty  $K_p$  sa zrýchluje 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 chyby, ako aj dobe jej trvania. Pokiaľ má regulovaná veličina menšiu hodnotu ako je požadovaná, integrálna časť PID sa zväčšuje. Naopak pokiaľ je hodnota regulovanej veličiny väčšia ako požadovaná, integrálna časť PID klesá. Pri použití integračnej zložky PID regulátora, bude trvalá regulačná odchýlka nulová [35]. Čím nižšia bude hodnota  $T_i$  Rov. 3.3, tým rýchlejšie sa bude výstup približovať žiadanej hodnote, no zároveň však bude narastať kmitanie sústavy.

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

$$T_i = \frac{K_p}{K_i} \quad (3.3)$$

- **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 počítaný podľa Rov. 3.4. Zvyšovanie derivačnej zložky PID regulátora tlmí kmitanie sústavy. Ak však zvolíme derivačnú konštantu priveľkú, kmitanie začne znova narastať [35]. V praxi sa derivačná zložka veľmi nepoužíva a využívaný je P alebo PI regulátor [36].

$$u(t) = K_d \frac{de(t)}{dt} \quad (3.4)$$

Pre lepšie chápanie je v Tab. 3.1 [37] zhrnutý vplyv jednotlivých zložiek PID, na odozvu systému v uzavorenenej slučke a to pri zvyšovaní hodnoty  $K_p$ ,  $K_i$  a  $K_d$ <sup>10</sup>.

	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. 3.5.

$$u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \quad (3.5)$$

<sup>10</sup>Tabuľka slúži len ako pomôcka pre hrubé ladenia. Jednotlivé zložky PID regulátora, ako aj ich vplyv na sústavu, sú v realite previazané.

$$u(t) = K_p \left( e(t) + \frac{1}{T_i} \int_0^t e(\tau) d\tau + T_d \frac{de(t)}{dt} \right) \quad (3.6)$$

Rovnica 3.5 predstavuje jeden z možných zápisov vzťahu pre PID regulátor. V praxi sa často využíva tvar Rov. 3.6, z dôvodu lepšej interpretácie parametrov, ktoré rovnicu 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) [37].

Predchádzajúce rovnice PID regulátorov fungujú pri spojitéh procesoch. Avšak pri implementácii PID pomocou číslicových regulátorov, musíme rovnicu transformovať do jej diskrétnej podoby Rov. 3.7.

$$u(kT) = K_p \left( e(kT) + \frac{T}{T_i} \sum_{i=0}^k e(iT) + \frac{T_d}{T} [e(kT) - e[(k-1)T]] \right) \quad (3.7)$$

Diskrétna forma PID regulátora je využívaná z toho dôvodu že Arduino resp. počítače nie sú schopné nepretržito zaznamenávať merané hodnoty. Dáta sú preto spracované v diskrétnych časových intervaloch  $kT$ , ktorým hovoríme vzorky. Proces získavania takýchto vzoriek v istej pravidelnom frekvencii, nazývame vzorkovanie.

Vzorkovanie môže mať rýchlosť niekolko minút, pri pomaly sa meniacich procesoch, až po mikrosekundy pri procesoch dynamických. Rovnicu v tvare 3.7 využíva na implementáciu PID regulátora knižnica AutomationShield.

## 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. Možnosti zadávania 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 `Ts` 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 (Uhlo ktory chceme
                         dosiahnut)
float y = 0.0;            // Vystup (Realny uhol kyvadla)
float u = 0.0;            // Vstup (Vykon 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 periódna. Slúži na kontrolu postupnosti vzoriek a povolenie spustenia nasledujúcej vzorky. Táto funkcia je volaná vždy len na začiatku jednotlivých vzoriek. Pokial 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.

Pokial 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`. Pokial 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;         // Vynuluje 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 velkostí premennej `i`, ktorá zaznamenáva počet vykonaných sekcií trajektórie, oproti velkosti 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. Pokial 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 premená `y`. Veľkosť regulačnej odchýlky, obmedzenie integrač-

ného nasýtenia<sup>11</sup>(angl. anti-windup), ako aj hodnotu saturácie systému<sup>12</sup>, zadávame do algoritmu 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++; // Priprav sa na nasledujucu 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 zaznamenané programom CoolTerm a následne vykreslené do grafov v prostredí MATLAB.

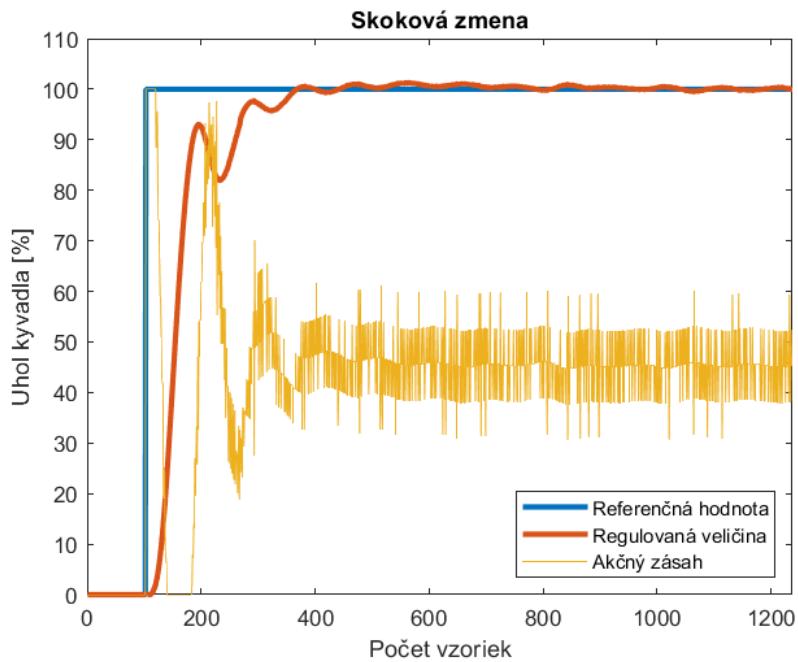
Na Obr. 3.2, vidíme reakciu systému na skokovú zmenu z nulovej referenčnej hodnoty na hodnotu maximálnu, teda 100%. Pomocou sledovania odozvy systému vieme lepšie nastaviť parametre PID regulátora. Systém nadobudne 1% regulačnú odchýlku v priebehu 500 vzoriek, čo znamená čas približne jeden a pol sekundy.

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

---

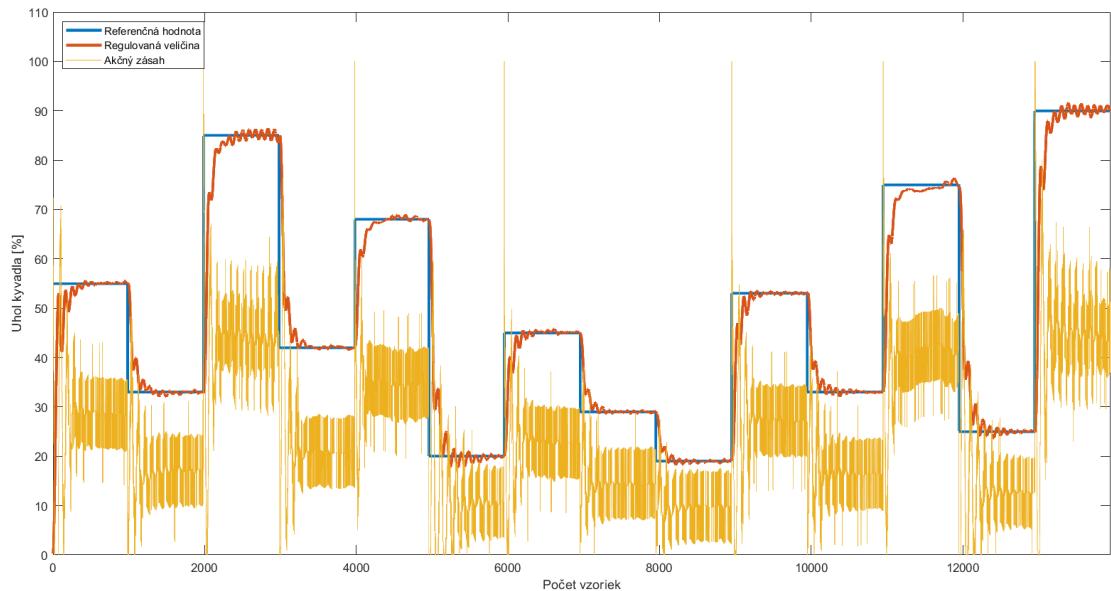
<sup>11</sup>K nasýteniu integračnej zložky dochádza, v prípade že akčný člen nie je schopný dosiahnuť požadovanú referenčnú hodnotu. V takom prípade začne hodnota integračná zložka nekontrolovatelné stúpat.

<sup>12</sup>Ak sa ktorákolvek zo zložiek PID regulátora dostane do oblasti nasýtenia, ďalšia zmena tejto zložky nevyvolá žiadnu odozvu.

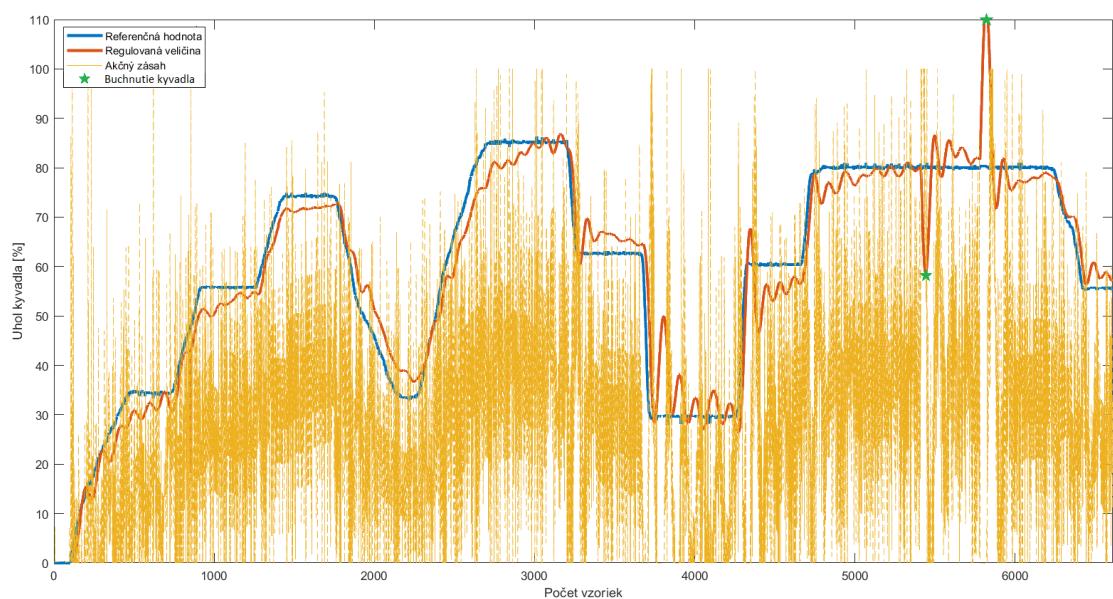


Obr. 3.2: Reakcia systému na skokovú zmenu referenčnej hodnoty- Arduino IDE.

môžeme všimnúť časť medzi vzorkami 5000-6000. Ide o manuálne zavedenie chyby, pomocou buchnutia do kyvadla. Systém na takúto zmenu reaguje zmenou akčného zásahu a regulačnú odchýlku menšiu ako 2%, nadobúda v priebehu jednej sekundy.



Obr. 3.3: Automatická trajektória.



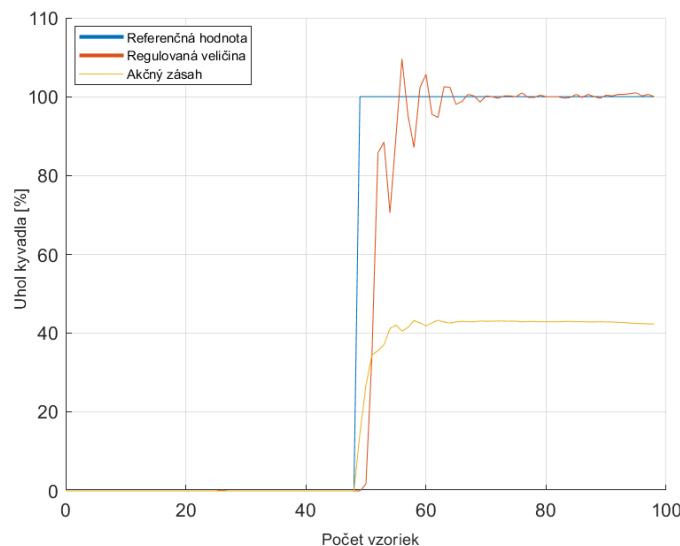
Obr. 3.4: Manuálna trajektória.

### 3.1.2 MATLAB

Príklad na ukážku fungovania PID regulátora bol taktiež vytvorený v prostredí MATLAB a simulink. 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 `PIDresponse(k,:)= [r y u]`, a toto pole je vykreslované pomocou funkcie `plotLive(PIDresponse(k,:))`. Na konci sú všetky dáta uložené, a teda sú prístupné aj po ukončení programu. Kompletný zdrojový kód sa nachádza v prílohe 4.

Regulácia PID v prostredí MATLAB potrebuje pre svoje fungovanie pomerne vysoký výpočtový výkon počítača, pretože výpočty prebiehajú na zariadení, s ktorým je Arduino prepojené. Na zariadení [38] v ktorom boli písané všetky didaktické príklady dosahujeme rýchlosť maximálne piatich vzoriek za sekundu tj.  $T_s=0.2\text{s}$ . Pri pomalšom vzorkovaní musíme spomaliť aj rýchlosť reakcie PID regulátora (napr. znížením proporcionalnej zložky), inak regulácia prestane byť možná. Výpočtová náročnosť sa dá znížiť, zamedzením vykreslovania grafu, alebo odstránením možnosti ukladania zaznamenaných dát.



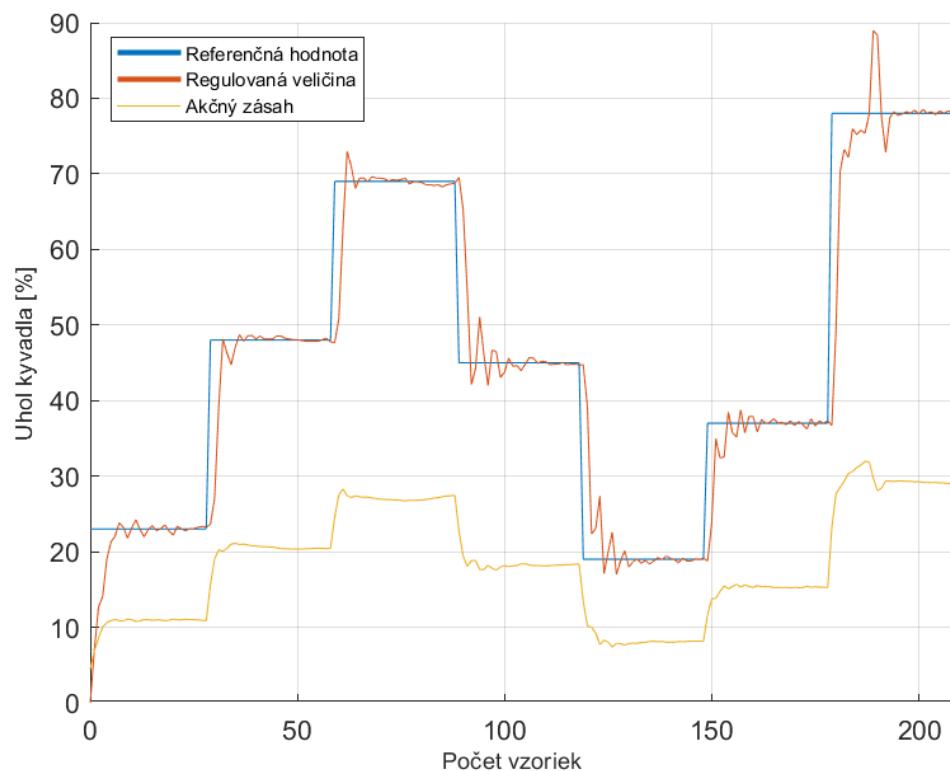
Obr. 3.5: Reakcia systému na skokovú zmenu referenčnej hodnoty- MATLAB.

### Výstupy

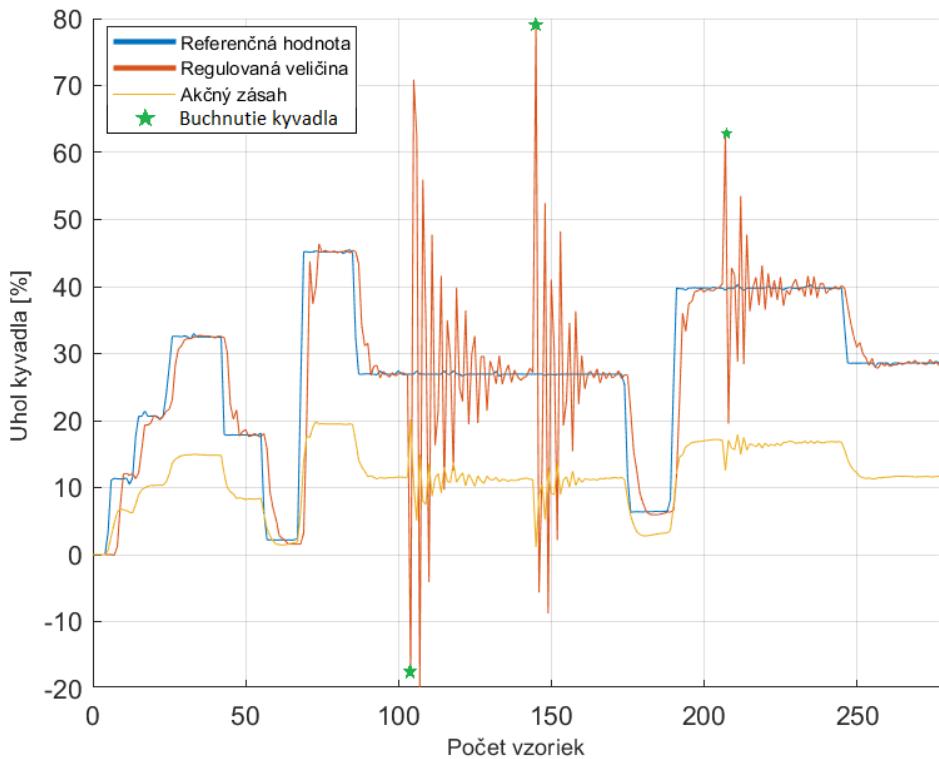
Všetky výstupy z príkladu 3.1.2, majú priamu funkciu vykreslovania grafov spolu s legendou výstupov. Tieto grafy majú taktiež definovaný rozsah zobrazovaných hodnôt na oboch osiach, ako aj pomenovania týchto osí. Na Obr. 3.5 vidíme reakciu systému na

jednotkový skok. Obrázok 3.6 zobrazuje automatickú trajektóriu referenčnej hodnoty a Obr. 3.7 trajektóriu manuálnu.

Pri manuálnej trajektórii bola trikrát vnesená velká chyba a to pomocou úderu do kyvadla. Ako je vidieť z grafu 3.7, 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 predovšetkým pomalším vzorkovaním.



Obr. 3.6: Automatická trajektória.



Obr. 3.7: Manuálna trajektória.

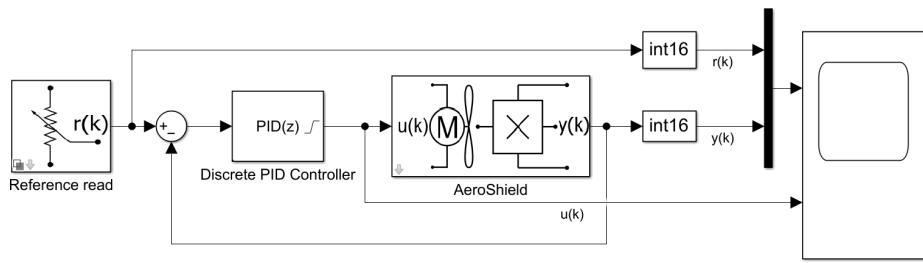
### 3.1.3 Simulink

Vzhľadovo pôsobí príklad PID riadenia v API Simulink jednoducho a elegantne. Predpripravené bloky z knižnice AeroLibrary stačí v príklade pospájať podľa potreby a následne zvoliť vhodné parametre v maskách blokov. Prepájanie blokov slúži na spájanie vstupov s výstupmi, alebo na matematické operácie s premennými. Regulovanú sústavu v tomto príklade predstavuje blok **AeroShield**, do ktorého vstupuje z bloku **Reference read** percentuálna hodnota referenčnej trajektórie. Z bloku **AeroShield** získavame ako výstup uhol kyvadla, ktorý využívame na výpočet regulačnej odchýlky.

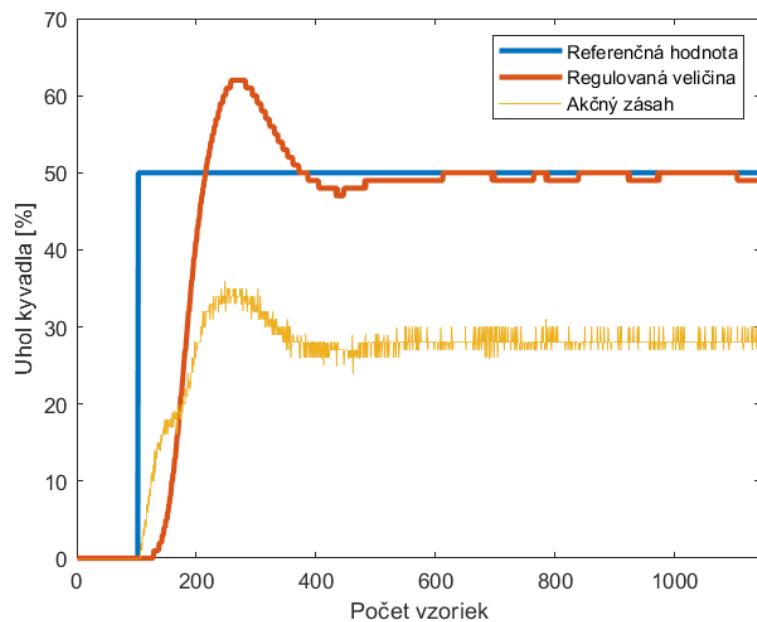
Blok **Discrete PID Controller** predstavuje riadiaci systém PID regulátora. Hodnoty jednotlivých zložiek regulátora sú: P=0.011, I=200, D=8. Matematická reprezentácia výpočtového algoritmu ideálneho PID regulátora, je v tvare Rov. 3.8:

$$P \left( 1 + I * T_s \frac{1}{z - 1} + D * \frac{1}{T_s} \frac{z - 1}{z} \right) \quad (3.8)$$

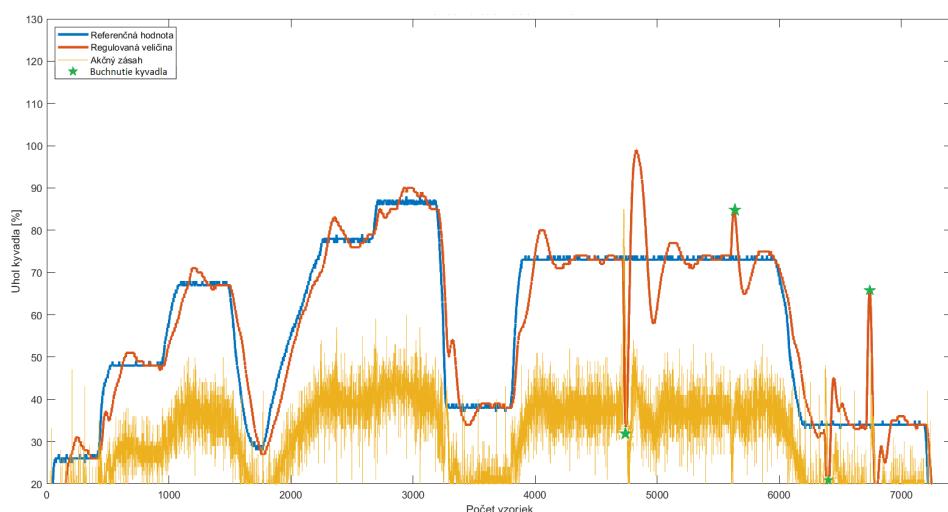
## Výstupy



Obr. 3.8: Ukážka riadenia systému pomocou PID regulátora v API Simulink.



Obr. 3.9: Reakcia systému na skokovú zmenu referenčnej hodnoty- Simulink.



Obr. 3.10: Manuálna trajektória.

## 4 Záver

V rámci bakalárskej práce boli vytvorené 2 verzie AeroShieldu R2 1.1.3 a R3 1.1.5, ako aj jeho spojovacie prvky a telo kyvadla 1.1.4. Zároveň boli vytvorené a otestované 3 knižnice AeroShieldu a to pre API Arduino IDE 1.2.1, MATLAB 1.2.2 a Simulink 1.2.3. Na ukážku funkcií z knižníc boli vytvorené príklady v otvorenej slučke bez spätnej väzby 2.1.1, ako aj príklady v uzavorennej slučke so spätnou väzbou 3.1.1. Týmito príkladmi 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. Momentálne sa pracuje na vylepšení výstupu z merania prúdu pomocou metódy filtrácie (low pass filter).

Ďalším problémom pri PID regulácii AeroShieldu je jeho zložité a dlhotrvajúce nastavenie parametrov v API Arduino IDE a MATLABE. Problémom bolo taktiež pomalšie vzorkovanie v príklade 1.2.2, ako aj nelineárne správanie sústavy.

Priame prepojenie motora so Shieldom taktiež spôsobuje isté nepríjemnosti. 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 meríme jeho uhol. Použitý konektor by preto musel mať stredovú časť s možnosťou pripojenia magnetu, alebo by sa uhol kyvadla musel merat 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.

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), „fuzzy” PID a mnoho ďalších. 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. Jedná sa skôr o 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. 2022, <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. 2022, <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] Andrej Polák. Spracovanie meraných údajov, modelovanie a identifikácia mechatronického systému aerokyvadlo. 2022, <https://opac.crzp.sk/?fn=detailBiblioForm&sid=85D30AFD358849054A9BB34E20A9&seo=CRZP-detail-kniha>.
- [8] Jakub Ondera. Návrh konštrukcie, riadiacich prvkov a algoritmov mechatronického systému aerokyvadlo. 2022, <https://opac.crzp.sk/?fn=detailBiblioForm&sid=85D30AFD358849054A9BBC4E20A9&seo=CRZP-detail-kniha>.
- [9] Fanavararan Sharif. Aero pendulum control system (pr22). Store. Online., 2021. 2021, <https://www.zodel.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] Arduino uno rev3. Info. Online., 2021. 2021, <https://store.arduino.cc/products/arduino-uno-rev3>.
- [14] Texas instruments tps56339 buck converters. store. Online., 2021. 2021, <https://www.mouser.ee/new/texas-instruments/ti-tps56339-buck-converters/>.
- [15] Arduino. Overview of the arduino uno components. article. Online., 2021. 2021, <https://docs.arduino.cc/tutorials/uno-rev3/intro-to-board>.
- [16] Arduino uno r3 - schematic with ch340. article. Online., 2021. 2021, [http://electronoobs.com/eng\\_arduino\\_tut31\\_sch3.php](http://electronoobs.com/eng_arduino_tut31_sch3.php).
- [17] DANIELLE COLLINS. What are coreless dc motors? article. Online., 2018. 09.10.2021, <https://www.motioncontroltips.com/what-are-coreless-dc-motors/>.
- [18] 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.
- [19] Pmv45en2. shop. Online., 2021. 2021, <https://www.nexperia.com/products/mosfets/small-signal-mosfets/PMV45EN2.html>.
- [20] 7mm diameter 720 coreless motor for quadcopter. shop. Online., 2021. 2021, <https://www.elecrow.com/7mm-diameter-720-coreless-motor-for-quadcopter.html>.
- [21] SHAWN HYMEL. Ina169 breakout board hookup guide. article. Online., 2021. 2021, <https://learn.sparkfun.com/tutorials/ina169-breakout-board-hookup-guide/all>.
- [22] Ina169na/250. store. Online., 2021. 2021, <https://www.ti.com/store/ti/en/p/product/?p=INA169NA/250>.
- [23] The Editors of Encyclopaedia Britannica. Hall effect. article. Online., 2021. 2021, <https://www.britannica.com/science/Hall-effect>.
- [24] Hallův snímač as5600-asom. store. Online., 2021. 2021, <https://sk.rsdelivers.com/product/ams/as5600-asom/halluv-snimap-as5600-asom-pocet-koliku-8-soic-typ/2006337>.
- [25] As5600 position sensor 12-bit on-axis magnetic rotary position sensor with analog or pwm output. 2022, <https://ams.com/en/as5600>.

- [26] Tps56339 data sheet. 4.5-v to 24-v input, 3-a output synchronous buck converter. 2022, <https://www.ti.com/product/TPS56339>.
- [27] Prototyping circuit boards: Everything you need to know before you start. article. Online., 2020. 2020, <https://wwwpcbnet.com/blog/prototyping-circuit-boards-everything-you-need-to-know-before-you-start/>.
- [28] What are the factors that will affect the pcb lifespan and how to extend it: 2021 ne-west. 2022, [https://www.pcbonline.com/blog/How\\_Is\\_The\\_Lifespan\\_of\\_PCB\\_Products\\_Determined\\_117.html](https://www.pcbonline.com/blog/How_Is_The_Lifespan_of_PCB_Products_Determined_117.html).
- [29] Writing a library for arduino. article. Online., 2022. 2022, <https://docs.arduino.cc/hacking/software/LibraryTutorial>.
- [30] Programming for engineers multiple source files. 2022, [https://www.albany.edu/faculty/dsaha/teach/2018Spring\\_CEN200/slides/19\\_mulSrc.pdf](https://www.albany.edu/faculty/dsaha/teach/2018Spring_CEN200/slides/19_mulSrc.pdf).
- [31] Arduino - data types. article. Online., -. 2022, [https://www.tutorialspoint.com/arduino/arduino\\_data\\_types.htm](https://www.tutorialspoint.com/arduino/arduino_data_types.htm).
- [32] Nicholas Zambetti. A guide to arduino & the i2c protocol (two wire). article. Online., 2022. 2022, <https://docs.arduino.cc/learn/communication/wire>.
- [33] Bit shifting. article. Online., 2017. 2022, <https://www.techopedia.com/definition/26846/bit-shifting>.
- [34] handle class. 2022, <https://www.mathworks.com/help/matlab/ref/handle-class.html>.
- [35] K. J. Astrom and T. Hagglund. *PID Controllers, Theory, Design and Tuning*. 2022, <https://aiecp.files.wordpress.com/2012/07/1-0-1-k-j-astrom-pid-controllers-theory-design-and-tuning-2ed.pdf>.
- [36] Kiam Heong Ang, G. Chong, and Yun Li. Pid control system analysis, design, and technology. *IEEE Transactions on Control Systems Technology*, 13(4):559–576, 2005.
- [37] Changbo Xu, Congcong Liu, Zhenfu Bi, and Chengjin Zhang. *2006 6th World Congress on Intelligent Control and Automation*.
- [38] Acer swift 3. 2021, <https://store.acer.com/en-in/acer-swift-3-thin-and-light-laptop-amd-ryzen-5-5500u-hexa-core-processor-radeon-graphics-512gb-ssd-office-2021-sf314-43>.





# 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             void begin();
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_addr);
41             word readTwoBytes(int in_addr_hi, int in_addr_lo);
42     };
43 #endif
```

Zdrojový kód 4.1: Zdrojový kód súboru AeroShield.h.

# Zdrojový kód súboru AeroShield.cpp

```
1 #include "AeroShield.h"
2
3 void AeroClass::begin( void ){
4     bool isDetected = AeroShield.detectMagnet();
5     pinMode(AERO_UPIN,OUTPUT);
6
7     #ifdef ARDUINO_ARCH_AVR
8         Wire.begin();
9     #elif ARDUINO_ARCH_SAM
10        Wire1.begin();
11    #elif ARDUINO_ARCH_SAMD
12        Wire.begin();
13    #endif
14    if(isDetected == 0){
15        while(1){
16            if(isDetected == 1){
17                AutomationShield.serialPrint("Magnet detected \n");
18                break;
19            }
20            else{
21                AutomationShield.serialPrint("Can not detect
22                                magnet \n");
23            }
24        }
25    }
26
27    float AeroShield::convertRawAngleToDegrees(word newAngle) {
28        float retVal = newAngle * 0.087;
29        ang = retVal;
30        return ang;
31    }
32
33    float AeroShield::calibration(word RawAngle) {
34        AutomationShield.serialPrint("Calibration running ... \n");
35        startangle=0;
36        analogWrite(AERO_UPIN,50);
37        delay(250);
38        analogWrite(AERO_UPIN,0);
39        delay(4000);
40
41        startangle = RawAngle;
42        analogWrite(AERO_UPIN,0);
43        for( int i=0; i<3; i++){
44            analogWrite(AERO_UPIN,1);
45            delay(200);
46            analogWrite(AERO_UPIN,0);
47            delay(200);
48        }
49        AutomationShield.serialPrint("Calibration done");
50        return startangle;
51    }
52
53    float AeroShield::referenceRead( void ) {
54        referencePercent = AutomationShield.mapFloat(analogRead(AERO_RPIN), 0.0,
55                                         1024.0, 0.0, 100.0);
56        return referencePercent;
57    }
58
59    void AeroShield::actuatorWrite( float PotPercent ) {
60        float mappedValue = AutomationShield.mapFloat(PotPercent, 0.0, 100.0,
61                                         0.0, 255.0);
62        mappedValue = AutomationShield.constrainFloat(mappedValue, 0.0, 255.0);
63        analogWrite(AERO_UPIN, (int)mappedValue);
64    }
65
66    float AeroShield::currentMeasure( void ) {
67        for( int i=0 ; i<repeatTimes ; i++){
68            voltageValue= analogRead(VOLTAGE_SENSOR_PIN);
69            voltageValue= (voltageValue * voltageReference) / 1024;
70            current= current + correction1-(voltageValue / (10 * ShuntRes));
71        }
72        currentMean= current/repeatTimes;
73        currentMean= currentMean-correction2;
74        if(currentMean < 0.000){
75            currentMean= 0.000;
76        }
77        current= 0;
78        voltageValue= 0;
79        return currentMean;
80    }
81
82    word AeroShield::getRawAngle()
83    {
84        return readTwoBytes(_raw_ang_hi, _raw_ang_lo);
85    }
86
```

```

85     int AeroShield::detectMagnet()
86     {
87         int magStatus;
88         int retVal = 0;
89         magStatus = readOneByte(_stat);
90         if ((magStatus & 0x20)
91             retVal = 1;
92             return retVal;
93     }
94
95     int AeroShield::getMagnetStrength()
96     {
97         int magStatus;
98         int retVal = 0;
99         magStatus = readOneByte(_stat);
100        if (detectMagnet() == 1)
101        {
102            retVal = 2;
103            if (magStatus & 0x10)
104                retVal = 1;
105            else if (magStatus & 0x08)
106                retVal = 3;
107            } return retVal;
108    }
109
110    int AeroShield::readOneByte( int in_adr )
111    {
112        int retVal = -1;
113        Wire.beginTransmission(_ams5600_Address);
114        Wire.write(in_adr);
115        Wire.endTransmission();
116        Wire.requestFrom(_ams5600_Address, 1);
117        while (Wire.available() == 0);
118        retVal = Wire.read();
119        return retVal;
120    }
121
122    word AeroShield::readTwoBytes( int in_adr_hi, int in_adr_lo )
123    {
124        word retVal = -1;
125        /* Read Low Byte */
126        Wire.beginTransmission(_ams5600_Address);
127        Wire.write(in_adr_lo);
128        Wire.endTransmission();
129        Wire.requestFrom(_ams5600_Address, 1);
130        while (Wire.available() == 0);
131        int low = Wire.read();
132
133        /* Read High Byte */
134        Wire.beginTransmission(_ams5600_Address);
135        Wire.write(in_adr_hi);
136        Wire.endTransmission();
137        Wire.requestFrom(_ams5600_Address, 1);
138        while (Wire.available() == 0);
139        word high = Wire.read();
140        high = high << 8;
141        retVal = high | low;
142        return retVal;
143    }

```

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();
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     pendulumAngle= AutomationShield.mapFloat(AeroShield.getRawAngle(), 
24         startAngle, lastAngle, 0.00, 90.00);
25     referencePercent= AeroShield.referenceRead();
26     AeroShield.actuatorWrite(referencePercent);
27     CurrentMean= AeroShield.currentMeasure();
28
29     Serial.print(pendulumAngle);
30     Serial.print(" ");
31     Serial.print(referencePercent);
32     Serial.print(" ");
33     Serial.println(CurrentMean);
34 }
```

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[]={45.0,23.0,75.0,32.0,58.0,
21             10.0,35.0,19.0,9.0,43.0,23.0,65.0,15.0,80.0};
22 float r=0.0;
23 float y = 0.0;
24 float u = 0.0;
25
26 void setup() {
27     Serial.begin(250000);
28     AeroShield.begin();
29     startAngle = AeroShield.calibration(AeroShield.getRawAngle
29         ());
30     lastAngle=startAngle+1024;
31     Sampling.period(Ts*1000);
32     PIDAbs.setKp(KP);
33     PIDAbs.setTi(TI);
34     PIDAbs.setTd(TD);
35     PIDAbs.setTs(Sampling.samplingPeriod);
36     Sampling.interrupt(stepEnable);
37 }
38
39 void loop() {
40     if(pendulumAngle>120){
41         AeroShield.actuatorWrite(0);
42         while(1);
43     }
44     if (nextStep) {
45         step();
46         nextStep = false;
47     }
48 }
49
50 void stepEnable() {
51     if(nextStep == true) {
52         realTimeViolation = true;
53         Serial.println("Real-time samples violated.");
54         analogWrite(5,0);
55         while(1);
56     }
57     nextStep = true;
58 }
59
60 void step() {
61     #if MANUAL
62         r = AeroShield.referenceRead();
63     #else
64         if(i>(sizeof(R)/sizeof(R[0]))) {
```

```

65          analogWrite(5,0);
66          while(1);
67      } else if (k % (T*i) == 0) {
68          r = R[i];
69          i++;
70      }
71 #endif
72 y= AutomationShield.mapFloat(AeroShield.getRawAngle(),
73 startAngle, lastAngle, 0.00, 100.00);
74 u = PIDAbs.compute(r-y, 0, 100, 0, 100);
75 AeroShield.actuatorWrite(u);
76
77 Serial.print(r);
78 Serial.print(" , ");
79 Serial.print(y);
80 Serial.print(" , ");
81 Serial.println(u);
82 k++;
83 }
```

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

Zdrojový kód 4.7: Zdrojový kód súboru AeroShieldPID.m.