

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE  
STROJNÍCKA FAKULTA**

**AEROSHIELD: MINIATÚRNY EXPERIMENTÁLNY MODUL  
AEROKYVADLA**

**Bakalárska práca**

**SjF-číslo b. práce**

**2022**

**Peter Tibenský**

**SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE  
STROJNÍCKA FAKULTA**

**AEROSHIELD: MINIATÚRNY EXPERIMENTÁLNY MODUL  
AEROKYVADLA**

**Bakalárska práca**

SjF-12345-67890

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

**Bratislava, 2022**

**Peter Tibenský**

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

## Čestné prehlásenie

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

Bratislava, 23. máj 2022

.....  
Vlastnoručný podpis

V prvom rade by som rád podľakoval vedúcej mojej bakalárskej práce, Ing. Mgr. Anne Vargovej, za odbornú pomoc, ľudský prístup a cenné rady pri vypracovávaní práce. Ďalej chcem podľakovať aj konzultantom bakalárskej práce, Ing. Erikovi Mikulášovi, za pomoc a pripomienky pri tvorbe dosky plošných spojov a návrhu 3D modelov.

Bratislava, 20. mája 2018

Peter Tibenský

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

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

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

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

**Keywords:** Arduino, AutomationShield, PID, AeroShield, AeroPendulum

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

# Obsah

<b>Zoznam obrázkov</b>	i
<b>Zoznam zdrojových kódov</b>	iii
<b>Zoznam tabuliek</b>	v
<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 . . . . .	12
1.1.4 Model držiaku kyvadla . . . . .	14
1.1.5 Cenová kalkulácia AeroShieldu . . . . .	14
1.1.6 Tretia verzia AeroShieldu . . . . .	14
1.2 Softvér . . . . .	17
1.2.1 Arduino API . . . . .	17
1.2.2 MATLAB . . . . .	27
1.2.3 Simulink . . . . .	29
<b>2 Didaktické príklady</b>	36
2.1 Programy v otvorenej slučke, bez späťnej väzby . . . . .	36
2.1.1 Arduino IDE . . . . .	36
2.1.2 MATLAB . . . . .	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 . . . . .	48
3.1.3 Simulink . . . . .	51
<b>4 Záver</b>	53
<b>Literatúra</b>	54

<b>Arduino IDE</b>	<b>iv</b>
Zdrojový kód AeroShield.h . . . . .	iv
Zdrojový kód AeroShield.cpp . . . . .	v
Zdrojový kód AeroShieldOpenLoop.ino . . . . .	vii
Zdrojový kód AeroShieldPID.ino . . . . .	viii
<b>MATLAB</b>	<b>x</b>
Zdrojový kód AeroShield.m . . . . .	x
Zdrojový kód AeroShieldOpenLoop.m . . . . .	xi
Zdrojový kód AeroShielPID.m . . . . .	xii

# Zoznam obrázkov

1	Experimentálne moduly vzdušného kyvadla.	2
2	Arduino UNO R3.[1]	3
3	Arduino Mega 2560 R3.[2]	4
1.1	Prvá verzia AeroShieldu.	5
1.2	Meranie uhla kyvadla	6
1.3	buck converter	7
1.4	Zapojenie akčného člena a typ motorčeka	8
1.5	meranie prúdu	9
1.6	Schematická reprezentácia hallovho javu.	9
1.7	meranie uhla kyvadla	10
1.8	Schéma zapojenia AeroShieldu	11
1.9	Vedľajšia doska AeroShieldu- breakout board	12
1.10	(a) Vrchná strana AeroShieldu (b) Spodná strana AeroShieldu	13
1.11	Dosky plošných spojov AeroShieldu	14
1.12	Model kyvadla, tvorený v programe CATIA.	16
1.13	Knižnica AeroLibrary.	29
1.14	Reference read- Simulink.	30
1.15	Angle read- Simulink.	31
1.16	Mapovanie uhlu kyvadla- Simulink.	31
1.17	Kalibrácia- Simulink.	32
1.18	Podsystém na kontrolu uhlu kyvadla.	33
1.19	Reference read- Simulink.	34
1.20	AeroShield_OpenLoop.	35
2.1	Výstup z programu AeroShieldOpenLoop.ino.	37
2.2	Výstup z programu AeroShieldOpenLoop.m.	40
3.1	Schéma riadenia PID regulátorom.	41
3.2	Reakcia systému na jednotkový skok.	47
3.3	Automatická trajektória.	47
3.4	Manuálna trajektória.	48
3.5	Manuálne zavedenie výchylky.	48
3.6	Reakcia systému na jednotkový skok.	49
3.7	Automatická trajektória.	50
3.8	Manuálna trajektória.	50
3.9	Ukážka riadenia systému pomocou PID regulátora v API Simulink.	51

3.10 Reakcia systému na skokovú zmenu. . . . .	52
3.11 Manuálna trajektória. . . . .	52

# Zoznam zdrojových kódov

1.1	Ukážka zdrojového kódu headeru.	18
1.2	Triedy a objekty.	18
1.3	Source volanie funkcie.	19
1.4	Zdrojový kód funkcie mapFloat.	20
1.5	Zdrojový kód funkcie serialPrint.	20
1.6	Zdrojový kód funkcie readOneByte.	21
1.7	Zdrojový kód funkcie readTwoBytes.	21
1.8	Zdrojový kód funkcie detectMagnet.	22
1.9	Zdrojový kód funkcie getMagnetStrength.	22
1.10	Zdrojový kód funkcie getRawAngle.	23
1.11	Zdrojový kód funkcie begin.	23
1.12	Zdrojový kód funkcie calibration.	24
1.13	Zdrojový kód funkcie convertRawAngleToDegrees.	25
1.14	Zdrojový kód funkcie referenceRead.	25
1.15	Zdrojový kód funkcie actuatorWrite.	26
1.16	Zdrojový kód funkcie currentMeasure.	26
1.17	Knižnica AeroShield.m properties.	27
1.18	Knižnica AeroShield.m properties.	28
1.19	Mapovacia funkcia vo fcn bloku.	32
1.20	Callback funkcia.	33
1.21	Initialization- maska Reference read.	34
2.1	AeroShield open loop dekleracia.	36
2.2	AeroShield open loop setup().	36
2.3	AeroShield open loop loop().	37
2.4	AeroShield open loop inicializacia.	38
2.5	AeroShield open loop grafy.	38
2.6	AeroShield open loop, while cyklus.	39
3.1	Načítanie knižníc a premenných do programu.	43
3.2	Organizačná funkcia setup.	44
3.3	Funkcia stepEnable().	44
3.4	Organizačná funkcia loop.	45
3.5	Funkcia step().	45
4.1	Zdrojový kód súboru AeroShield.h.	iv
4.2	Zdrojový kód súboru AeroShield.cpp.	v
4.3	Zdrojový kód súboru AeroShieldOpenLoop.ino.	vii
4.4	Zdrojový kód súboru AeroShieldPID.ino.	viii

4.5	Zdrojový kód súboru AeroShield.m.	x
4.6	Zdrojový kód súboru AeroShieldOpenLoop.m.	xi
4.7	Zdrojový kód súboru AeroShieldPID.m.	xii

# Zoznam tabuliek

1.1	Cenová kalkulácia AeroShieldu.	15
1.2	Dátové typy	19
3.1	Odozva systému na zmenu konštánt.	42

# Úvod

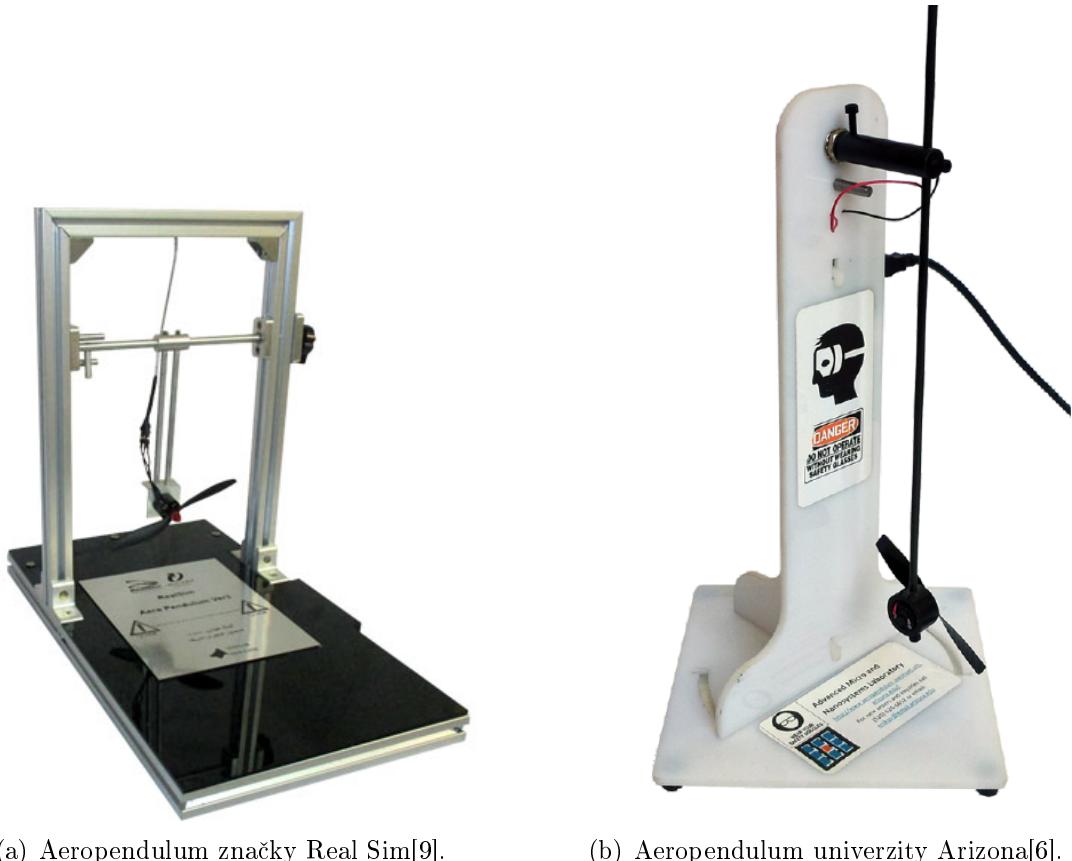
Cieľom tejto práce je návrh, výroba a naprogramovanie modernej učebnej pomôcky AeroShieldu (ďalej len „shield“), ktorá slúži na výučbu základov teórie riadenia a elektrotechniky. Učebné pomôcky sú nevyhnutnou, no často zanedbávanou súčasťou výučby. Študenti si vďaka nim môžu lepšie predstaviť a pochopiť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

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

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

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

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



(a) Aeropendulum značky Real Sim[9].

(b) Aeropendulum univerzity Arizona[6].

Obr. 1: Experimentálne moduly vzdušného kyvadla.

Open-source<sup>1</sup> projekt AutomationShield vyvíjaný na Ústave automatizácie, merania a aplikovanej informatiky SJF STU je zameraný na vývoj hardwarových a softwarových nástrojov určených na vzdelené a doplnenie vzdelenacieho procesu. Vedúcimi tohto projektu sú: Doc. Ing. Gergely Takács, PhD.[10] spolu s doc. Ing. Martin Gulan, PhD.[11]. Jadrom celého projektu je tvorba rozširujúcich dosiek (shieldov) vyvíjaných pre populárny typ prototypizačných dosiek s mikrokontrolérmi Arduino. Tieto pomerne lacné učebné pomôcky majú za cieľ lepšiu výučbu strojného inžinierstva, mechatroniky a základov automatického riadenia[12].

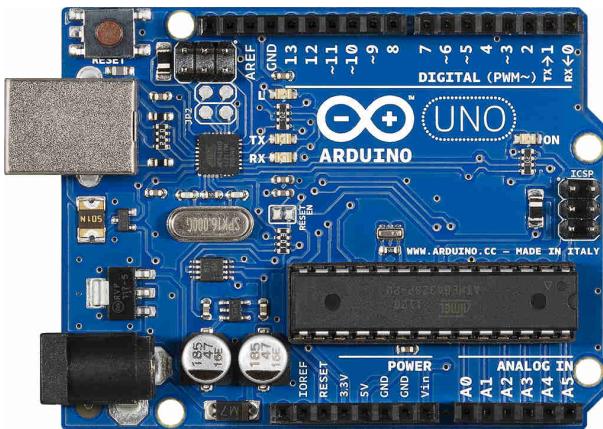
Všetky informácie ohľadom projektu AutomationShield, sú dostupné na platforme GitHub[13], ktorá slúži ako knižnica kódov, návodov a postupov, ktoré sú voľne dostupné na čítanie a úpravu. Na samostatnej stránke AutomationShieldu nájdeme zoznam jednotlivých shieldov a to, v akom procese výroby a fungovania sa nachádzajú. Ku každému shieldu nájdeme jeho podrobnejšiu dokumentáciu, knižnice, zdrojové kódy, ako aj predprogramované ukážky jeho fungovania. GitHub je open-source platforma. Dokumenty zdieľané na tejto stránke teda môže ktokoľvek upravovať, kontrolovať alebo vylepšovať, čo tvorí ideálny priestor pre rozvoj nových myšlienok a podporu tvorivého procesu.

---

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

Hlavnou motiváciou tohto projektu je nízka dostupnosť a vysoká cena podobných učebných pomôcok. Z môjho pohľadu je výučba častokrát až príliš zameraná na memorovanie faktov a teórie, namiesto praktických experimentov a skúseností typu pokus-omyl. Študenti pochopia vyučovanú teóriu jednoduchšie, pokiaľ majú možnosť experimenty sami tvoriť, skúmať a testovať[14].

S úmyslom priniesť širokej verejnosti lacnejšiu a výkonnejšiu alternatívu vtedajším mnohonásobne drahším a menej výkonným prototypizačným doskám[15] prišla na trh v roku 2005 prototypizačná doska Arduino. Projekt vznikol v Taliansku ako kolaborácia medzi viacerými nadšencami elektrotechniky a programovania, na ktorého čele bol Massimo Banzi. Veľkou výhodou dosiek Arduino a ich nadstavbových shieldov je fakt, že sú pomerne lacné a majú malé rozmer (Arduino UNO: 68.6\*53.4mm[16]). Tieto skutočnosti umožňujú študentom pracovať na experimentoch nielen na pôde školy, ale experimenty si môžu zobrať domov a pracovať na nich aj mimo vyučovacieho procesu.



Obr. 2: Arduino UNO R3.[1]

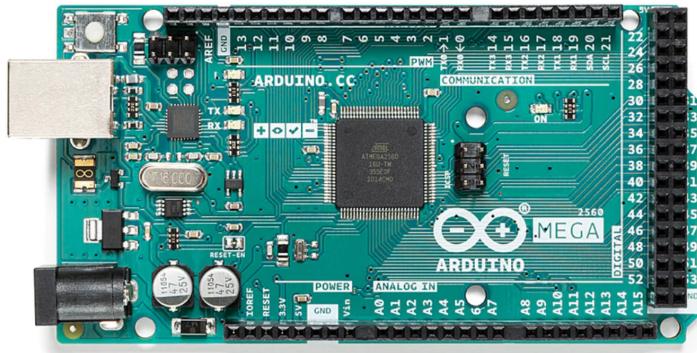
Na fungovanie a programovanie dosky postačuje len USB kábel, programovací softvér a samotná doska. Vzhľadom na nízky počet potrebných súčiastok a fakt, že mikročip Arduina je v prípade poruchy jednoducho vymeniteľný<sup>2</sup>, je jeho používanie na školách príjemné a jednoduché. Mikrokontroléri Arduino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

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

Niektoré piny sú označené špeciálnym symbolom "~~". Tieto piny sú schopné generovať 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]

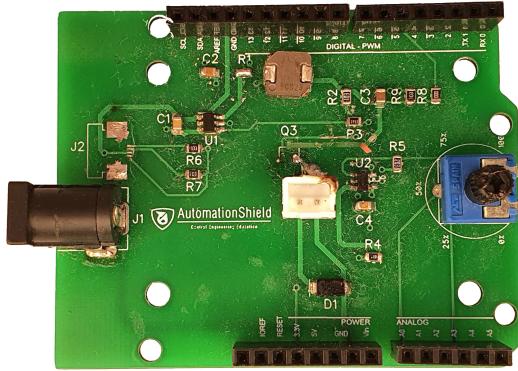
Práca je rozdelená do štyroch logických celkov. Na začiatku v časti Hardvér je opísaný základný princíp fungovania shieldu a následne jeho jednotlivé súčiastky. Nasleduje časť tvorby schémy zapojenia shieldu a dosky plošných spojov v programe DipTrace. Na konci časti Hardvér je spomenutá tvorba modelu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

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

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

# 1 AeroShield

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



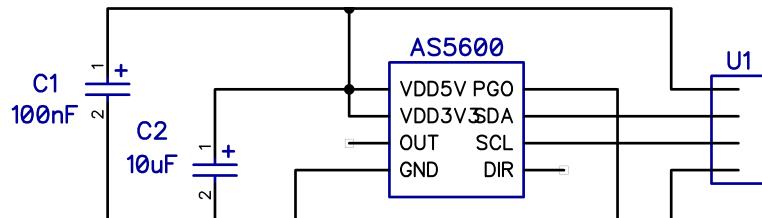
Obr. 1.1: Prvá verzia AeroShieldu.

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

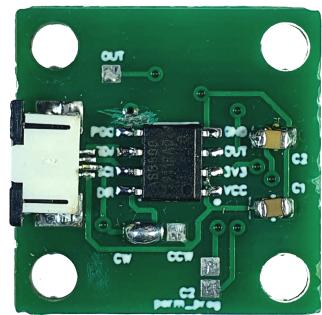
- neprepojenie pinov komunikácie I2C tj. piny SDA a SCL senzoru hall efektu, ktorý slúži na meranie uhlu natočenia kyvadla,
- nesprávne zapojenie mosfetu PMW45EN, ktorý ovláda PWM signál idúci do akčného člena,
- nesprávne umiestnená ochranná dióda na konektoroch akčného člena,
- nesprávne zapojený obvod 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, analyzovať chyby a ich následná oprava. V rámci projektu bola vytvorená hlavná doska, na ktorej sa nachádza väčšina elektroniky a menšia doska tzv. breakout board obr.1.2.b, ktorý

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



(a) Schéma zapojenia breakout boardu.



(b) Breakout board.

Obr. 1.2: Meranie uhla kyvadla

# 1.1 Hardvér

## 1.1.1 Popis súčiastok

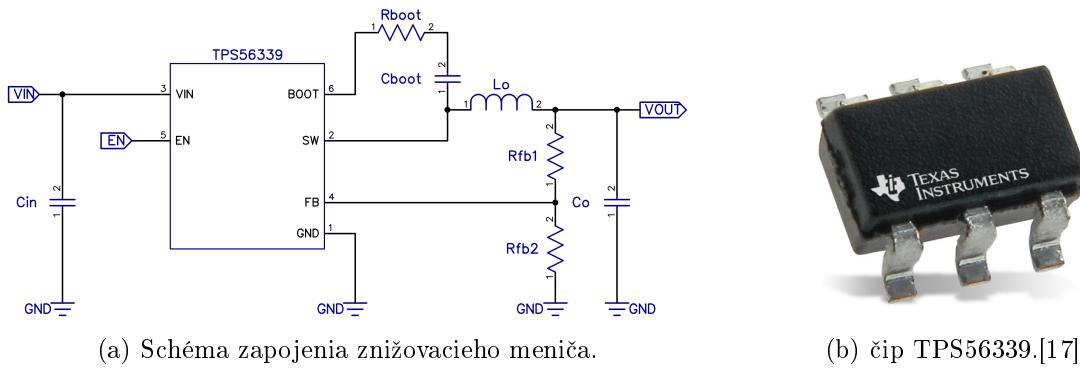
V tejto časti sa bližšie pozrieme na jednotlivé súčasti zapojenia AeroShieldu. Konkrétnie sa jedná o tieto prvky:

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

### Znižovací menič

Na napájanie akčného člena, motorčeka, potrebujeme napäťie v rozmedzí 0-3,7V. Na shield je však privádzané, pomocou koaxiálneho napájacieho konektora, napäťie 12V, 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-mΩ a 35-mΩ high-side mosfetov<sup>4</sup>, v spolupráci s ďalšími komponentami. Celkový prevádzkový prúd zariadenia je približne 98μA, keď funguje bez spínania a bez záťaže. Keď je zariadenie vypnuté, napájací prúd je približne 3μA a zariadenie umožňuje nepretržitý výstupný prúd do 3 A[17].



Obr. 1.3: buck converter

Na čip je privádzané napäťie 12V ktoré sa pomocou zapojenia viditeľného na schéme obr.1.3.a., znižuje na napäťie 3,7V. Domnievali sme sa že napájanie motora musí byť realizované externe, pomocou koaxiálneho napájacieho konektora z dôvodu vysokého prúdu odoberaného motorom počas silného zaťaženia. Rovnaký konektor sa sice nachádza aj na doske Arduino UNO a pomocou VIN pinu sa z neho dajú napájať napäťim 6-12V aj iné zariadenia, avšak tento pin je napojený na diódu, obmedzujúcu prúd na 1A[18][19].

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

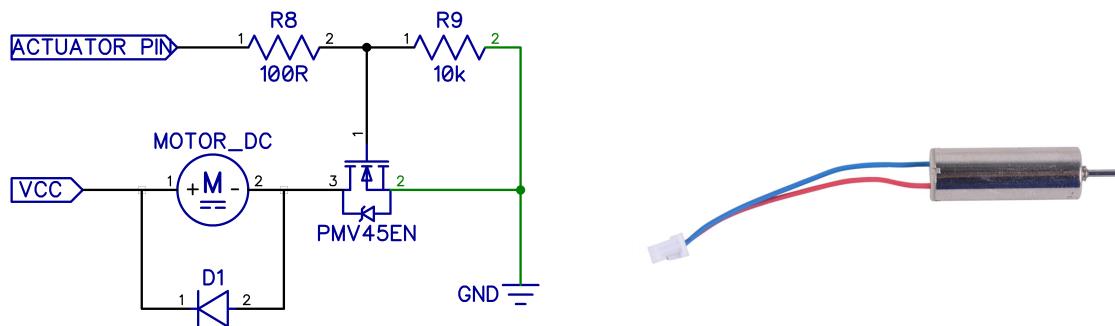
<sup>4</sup>N-kanálový mosfet je typ mosfetu, v ktorom tok prúdu nastáva kvôli pohybujúcim sa, záporne nabitém elektrónom. "High – side" znamená, že prúd prechádza z napájania, cez mosfet, do záťaže a potom do zeme.

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

### Akčný člen

Ako akčný člen AeroShieldu je použitý 7mm, 3,7V motorček na jednosmerný prúd bez jadra, používaný hlavne pre pohon dronov. "Coreless motor", alebo motor bez jadra, je motor s cievkou navinutou samou na sebe a nie na železe[20]. Takéto jadro ale nie je veľmi pevné a nedrží dobre tvar, preto sa častokrát zalieva epoxidom, ktoré jadro spevňuje. Stator je vyrobený z magnetov na báze vzácných zemín, ako je neodým alebo SmCo(samárium-kobalt), ktoré sa nachádzajú vo vnútri bezjadrového rotora.

Takýto motor ponúka mnoho výhod oproti motoru so železným jadrom. Tým že jadro v sebe nemá železo, výrazne sa znížuje hmotnosť a tým aj zotrvačnosť rotora, čo je dôležité pre naše použitie, kedy potrebujeme dosahovať vysokú akceleráciu a rýchle spomalenie rotora. Ďalšou výhodou je fakt, že nedochádza k stratám na železe a tým pádom sa účinnosť takýchto motorov blíži až ku 90%[21]. Motor, resp. otáčky motora sú riadené pomocou PWM signálu a ten do motoru prechádza cez N-kanálový mosfet PMV45EN2 od výrobcu Nexperia[22].



(a) Schéma zapojenia motorčeka.

(b) Akčný člen sústavy.[23]

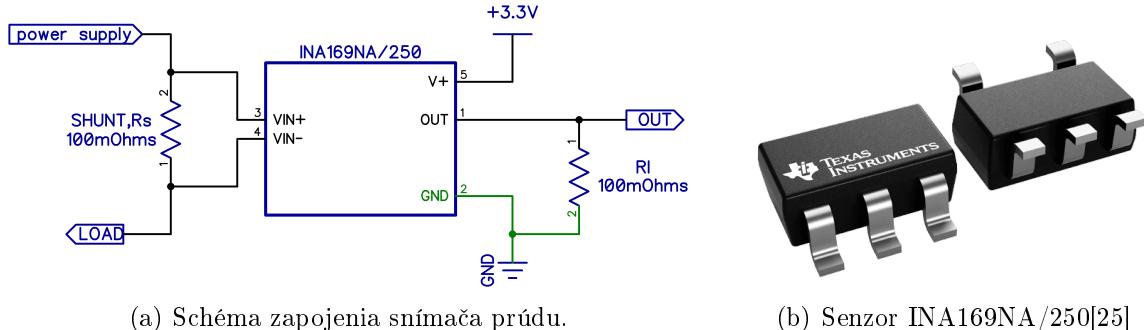
Obr. 1.4: Zapojenie akčného člena a typ motorčeka

### Meranie prúdu

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

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

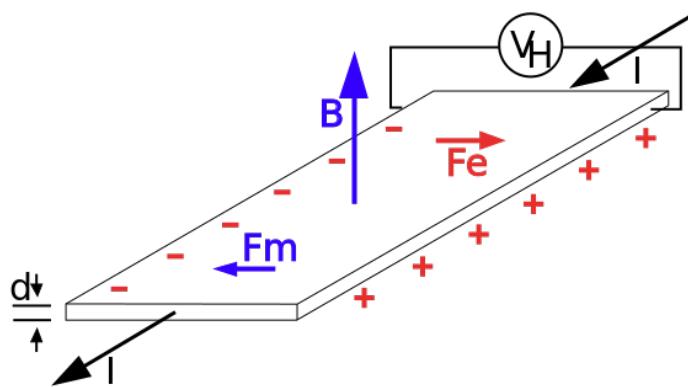
Prúd  $I_s$  odoberaný motorom, vypočítame pomocou vzorca  $I_s = (V_{OUT} \times 1k\Omega) / (R_s \times R_l)$  kde  $V_{OUT}$  je napätie namerané na výstupe,  $1k\Omega$  je konštanta vnútorných odporov senzoru,  $R_s$  je hodnota shunt rezistora v  $\Omega$  a  $R_l$  je hodnota rezistora na výstupe, taktiež v  $\Omega$ [24].



Obr. 1.5: meranie prúdu

### Meranie uhla kyvadla

Na správne fungovanie AeroShieldu je dôležité vedieť s vysokou presnosťou merať uhol naklonenia kyvadla. Na tento účel sme si zvolili meranie uhlia bez kontaktnou formou, pomocou snímača hall efektu. Hall efekt vieme opísť ako vznik priečneho elektrického poľa v pevnom materiáli, keď ním preteká elektrický prúd a tento materiál je umiestnený v magnetickom poli, ktoré je kolmé na prúd[26]. Toto elektrické pole resp. vznik elektrického potenciálu vieme detegovať ako Hallovho napätie a na základe jeho zmeny, vieme určiť rotáciu kyvadla. Fyzikálna podstata tohto javu je na obr.1.6, kde  $V_H$  je Hallovovo napätie,  $B$  je magnetické pole,  $F_m$  magnetická sila pôsobiaca na negatívne prenášače náboja,  $F_e$  elektrická sila z nahromadeného náboja,  $I$  je dohodnutý smer elektrického prúdu.

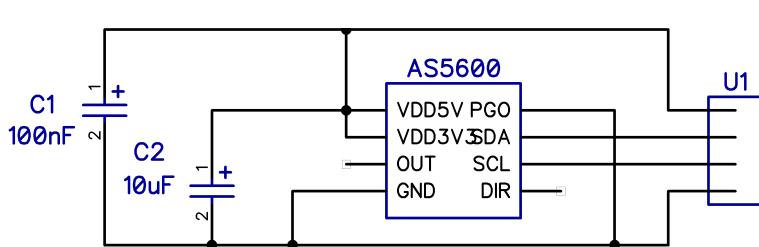


Obr. 1.6: Schematická reprezentácia hallovho javu.

V kyvadle je na konci horizontálneho ramena umiestnený špeciálny magnet kruhového tvaru, ktorý je polarizovaný naprieč prierezom magnetu. Ako senzor na meranie hall efektu je použitý AS5600 od výrobcu OSRAM obr.1.7.b. Signály prichádzajúce zo snímača

sa najprv zosilnia, následne sú filtrované a prechádzajú konverziou pomocou analógovo-digitálneho prevodníka(ADC). Snímaná je aj intenzita magnetického poľa, ktorou senzor pomocou automatického riadenia zosilnenia(AGC), kompenzuje zmeny teploty priestoru a taktiež zmeny sily magnetického poľa.

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



(a) Schéma zapojenia čipu na meranie uhlu.



(b) čip AS5600[27]

Obr. 1.7: meranie uhla kyvadla

### 1.1.2 Schéma zapojenia

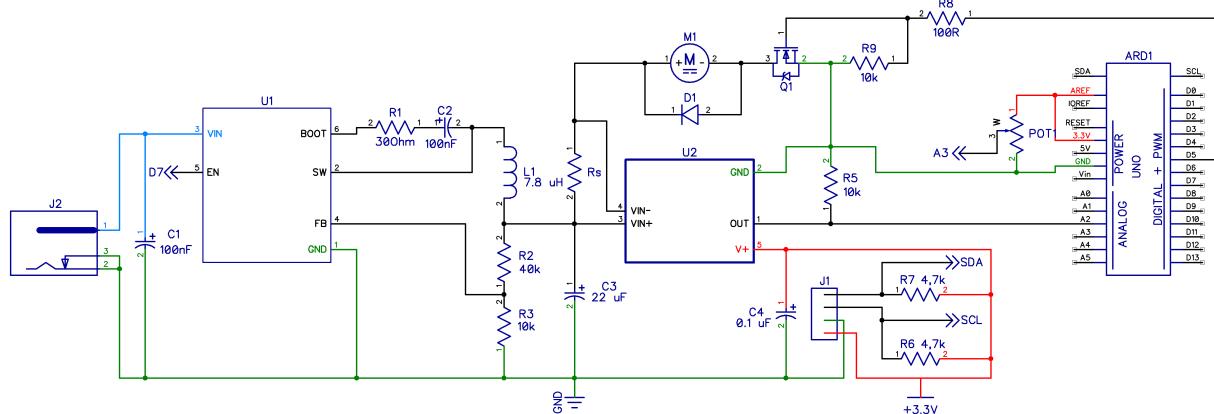
Všetky schémy zapojenia boli tvorené v bezplatnej verzii programu DipTrace. DipTrace slúži ako prostredie na tvorbu elektrotechnických schém a dosiek plošných spojov. Program v sebe zahŕňa aj časť pre tvorbu jednotlivých komponentov, pokiaľ sa tieto už nenachádzajú v niektornej z knižníc programu.

Nie všetky komponenty potrebné na tvorbu schémy zapojenia boli zahrnuté v knižniciach DipTracu, avšak tieto komponenty sú dostupné na stránkach výrobcov, odkiaľ sa dajú stiahnuť a následne použiť v schéme[28][29][25]. Do programu bola taktiež vložená knižnica AutomationShieldu ktorá má v sebe najčastejšie používané komponenty. Pri tvorbe schémy zapojenia sa najskôr všetky potrebné komponenty umiestnia na štvorcovú plochu a približne sa určí ich poloha. Jednotlivé komponenty majú podobu elektrotechnických značiek a každý komponent má ku sebe priradené reálne vlastnosti daného dielu(veľkosť, zapojenie, dĺžka pinov a iné).

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

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

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



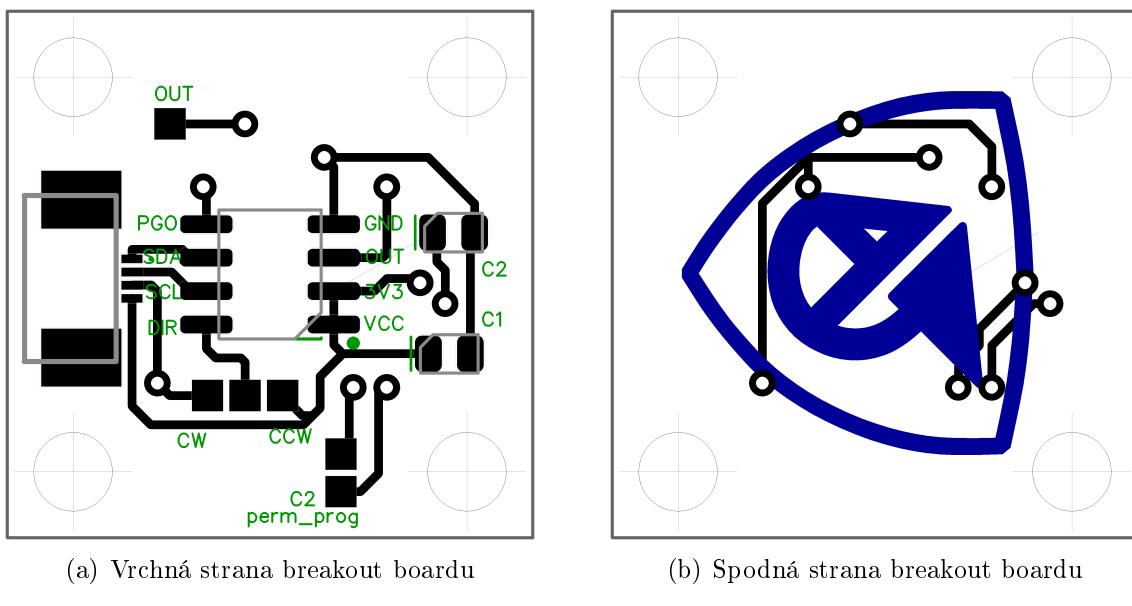
Obr. 1.8: Schéma zapojenia AeroShieldu

### 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 na spájkovanie. Dosky plošných spojov majú niekoľko výhod, ale aj negatív oproti ponúkaným alternatívam[30].

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[31]. Tým že vodivé cesty môžu mať veľmi malé rozmery, ovplyvňujúcim faktorom veľkosti dosky plošných spojov je samotná veľkosť použitých komponentov.

Po prenesení schém do DipTrace PCB, sú jednotlivé komponenty rozhádzané a nemajú žiadne logické rozloženie. Program ponúka možnosť automatického alebo manuálneho rozmiestnenia komponentov. Pri pohybovaní jednotlivými komponentami môžeme vidieť čiary, ktoré symbolizujú prepojenia s ostatnými súčiastkami a vďaka tomu vieme komponenty logicky poukladať.

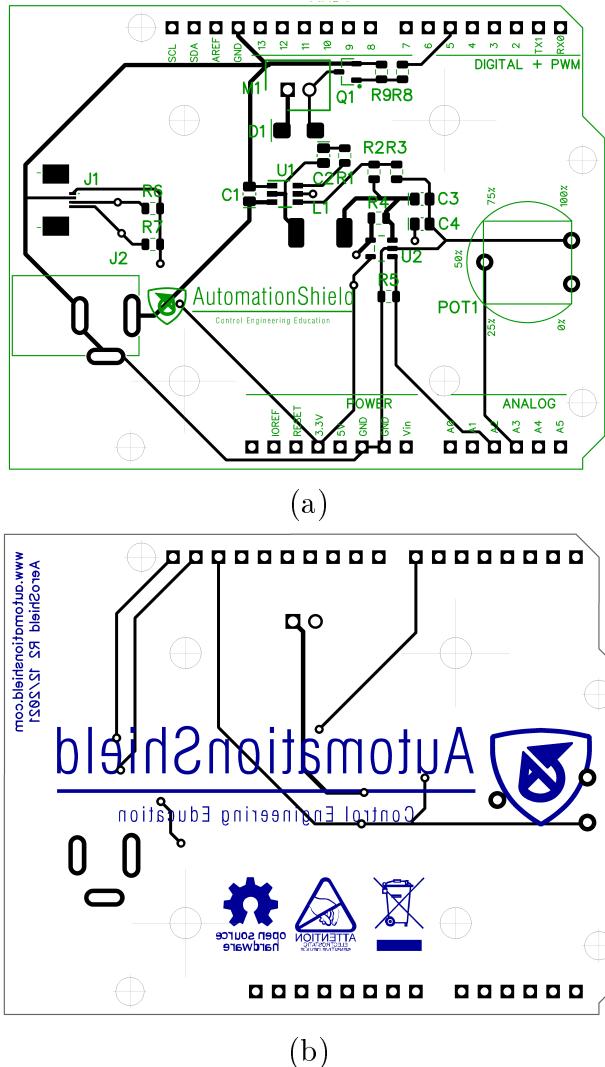


Obr. 1.9: Vedľajšia doska AeroShieldu- breakout board

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

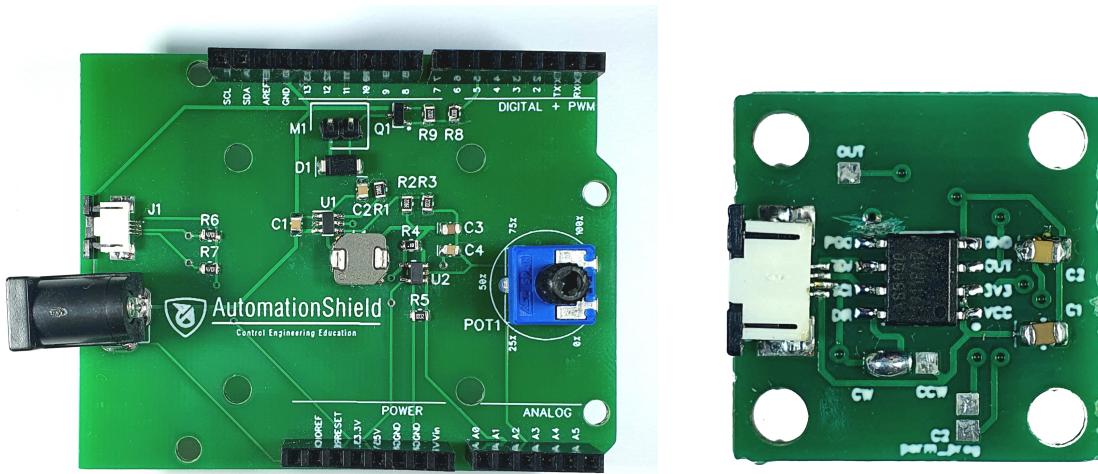
Tvorba elektrických ciest má niekoľko pravidiel. Najdôležitejšie z nich je, že cesty spájajúce rozdielne vodiče sa nemôžu križovať. Pokiaľ by k takému kríženiu došlo, jednotlivé cesty sa vzájomne vyskratujú. Z toho dôvodu treba niekedy cestu priviesť na druhú stranu dosky plošných spojov kde v jej pokračovaní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

používajú vodivé diery "via", spájajúce obe strany dosky. Druhá verzia dosky AeroShieldu je na obr.1.10.



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

Po finálnej kontrole zapojenia komponentov na doske plošných spojov, môžeme tieto dosky uložiť do formátu gerber. Súbory typu gerber v sebe ukladajú presné zloženie finálnej dosky plošných spojov a to po jej jednotlivých vrstvách. Nachádza sa tu teda vrstva zobrazujúca vodivé cesty, vrstva pre konektory via, vrstva pre farebné popisy a mnoho ďalších. Pri tvorbe súboru máme veľa možností aké parametre jednotlivých vrstiev chceme zvoliť. Môžeme meniť hrúbky jednotlivých vrstiev, veľkosť dier a priestoru okolo dier, veľkosť konektorov a ďalšie. Gerber súbor zasielame výrobcovi PCB dosiek kde si môžeme navoliť ďalšie parametre dosky, ako jej farbu, typy spájkovacích doštičiek, dokonca nám môže výrobca poslať už naspájkovanú dosku, ktorá je tak hned pripravená na použitie. Podobu finálnej dosky AeroShieldu môžeme vidieť na obr.1.11.a a dosky breakout boardu na obr.1.11.b.



(a) Hlavná doska AeroShieldu

(b) Vedľajšia doska AeroShieldu

Obr. 1.11: Dosky plošných spojov 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.

#### 1.1.5 Cenová kalkulácia AeroShieldu

Hlavnou podmienku pri tvorbe AeroShieldu, bola jeho funkcionálita, no zároveň nízka cena. Za účelom predstavy cenovej relácie jedného kusu AeroShieldu, bola zostavená tabuľka 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, keďže pri kúpe zopár kusov (1-20) je ich cena rádovo vyššia, ako cena pri nákupoch viacero kusov.

#### 1.1.6 Tretia verzia AeroShieldu

Názov	Popis	Ks.	Cena v €	Spolu
Kapacitor	SMD, sot23	6	0,6	3,6
Dióda	1N400IG	1	0,1	0,1
FFC konektor	FFC 4pin	2	0,2	0,4
Cievka	IND1210	1	0,2	0,2
Konektor DC motora	JST-XH 2,54	1	0,4	0,4
Motor	Howellp 7x20mm Motor	1	2,1	2,1
Potenciometer	CA14	1	0,45	0,45
Mosfet	pmv45en2	1	0,04	0,04
Rezistor	SMD, sot23	9	0,4	3,6
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 JLCPBCB	1	0,35	0,35
PCB brakout	Výroba JLCPBCB	1	0,35	0,35
Matice	M4	4	0,2	0,8
<b>Spolu</b>				<b>31,95€</b>

Tabuľka 1.1: Cenová kalkulácia AeroShieldu.



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

## 1.2 Softvér

### 1.2.1 Arduino API

Vývojové prostredie pre platformy arduino sa nazýva Arduino IDE<sup>5</sup> a využíva programovací jazyk C++ resp. jeho nadstavbu, s pridanými špecializovanými príkazmi a funkciami. C++ patrí medzi jeden z najviac používaných programovacích jazykov na svete. Je vhodný ako pre začiatočníkov, tak aj pre profesionálnych programátorov. V rámci projektu AutomationShield existuje niekoľko rôznych shildov, ktoré však často využívajú rovnaké funkcie a príkazy. Z tohto dôvodu bola vytvorená knižnica "AutomationShield", ktorá v sebe zahŕňa niekoľko ďalších, často využívaných knižníc a súborov. Jedná sa napríklad o funkcie potrebné pri riadení pomocou PID regulátora, vzorkovanie programu alebo mapovanie premenných. O jednotlivých funkciách v rámci knižnice "AutomationShield" si povieme viacej pri opise didaktických príkladov v časti:2.

V rámci programovania knižníc využívame objektovo orientované programovanie (OOP)[32]. OOP je výhodné z hľadiska prehľadnosti, úpravy funkcií, redukcie nepotrebného alebo duplicitného kódu a mnoho ďalšieho. Pri OOP vytvárame dátové štruktúry nazývané objekty. Objekty majú svoje vlastnosti, metódy a udalosti a pomocou kombinácií týchto vlastností vykonávajú určité naprogramované činnosti.

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

#### Header

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

Hneď za definovaním knižnice AeroShild.h môžeme vkladať ďalšie knižnice, ktoré sú potrebné pre funkcie danej knižnice, a to pomocou príkazu **#include**. Môžeme si všimnúť že za príkazom **#include** sa nachádzajú dva typy zátvoriek resp. znakov. Konvencia je taká že na preddefinované knižnice sa používajú hranaté zátvorky **<nazovKnižnice.h>** a na knižnice tvorené programátormi sa používajú úvodzovky **"nazovDalsejKnižnice.h"**.

Za knižnicami následne určujeme premenné, ktoré majú priradené fyzické čísla pinov na arduine. Tieto premenné potom využívame buď na posielanie, alebo na prijímanie signálov z daných pinov. Názvy týchto premenných sa snažíme voliť tak, aby bola na prvý pohľad jasná ich funkcia, alebo podľa všeobecne zaužívaných pravidiel. V teórii riadenia sa na označenie vstupov používa písmeno R a na označenie výstupov U,Y. Príkaz **#endif** vkladáme až na úplný záver header súboru.

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

```

#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.

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

```

class AeroShield{                      // Deklaracia triedy
    public :                           // Verejna cast
    void FirstObject();                 // Deklaracia funkcie

    private :                          // Sukromna cast
    float FirstVariable;              // Deklaracia premennej
};                                         // Koniec triedy

```

Zdrojový kód 1.2: Triedy a objekty.

V tomto prípade sa trieda nazýva **AeroShield** a má v sebe jednu funkciu s názvom **FirstObject()** v časti **public** a jednu premennú **FirstVariable**, typu **float**, v časti **private**. Rozdelenie na **public** a **privat** má zmysel hlavne v prípade ak chceme mať zadefinované isté premenné, pri ktorých nechceme aby sa dala externe zmeniť ich hodnota alebo typ. V prípade **privat**, takáto zmena nie je možná, jediná možnosť ako premennú zmeniť, je jej ručné prepísanie v súbore. V časti **private** deklarujeme funkcie ktoré následne využívame v rámci triedy a 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

Kedžže všetky potrebné knižnice sa už definovali a načítavali v súbore header, stačí nám už len časti source a header prepojiť. Urobíme tak pomocou príkazu `#include "AeroShield.h"`, ktorý vložíme na začiatok súboru. Ďalej v súbore deklarujeme jednotlivé funkcie, ktoré zapisujeme pomocou už spomínaného classu, dátového typu a názvu funkcie v podobe:

```
typFunkcie AeroShield :: nazovFunkcie()
```

Zdrojový kód 1.3: Source volanie funkcie.

V tomto prípade je AeroShield názov classu, nazovFunkcie hovorí sám za seba. Dátové typy funkcií poznáme rôzne. Vyberáme si ich na základe potreby ako chceme aby funkcia reagovala resp. aké hodnoty by mala prenášať. Dátové typy poznáme nasledovné[33] (všetky hodnoty sú platné pre arduino UNO, pre iné typy arduina sa hodnoty môžu lísiť):

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

Tabuľka 1.2: Dátové typy

## Popis použitých funkcií z knižnice AutomationShield

Ako už bolo spomenuté, knižnica AutomationShield ponúka najviac používané funkcie, ktoré sa využívajú takmer na každom shielde. Pri rôznych veľkostach a rozsahoch

číselných stupníc, je dobré vyjadrovať hodnoty v percentách, namiesto ich absolútnej hodnoty. Arduino ponúka funkciu `map()`, ktorá však pracuje len s dátovým typom `integer`. Aby sme docielili vyššiu presnosť, potrebujeme mapovať dátový typ `float`. Na tento účel nám slúži funkcia `mapFloat` do ktorej vstupuje veličina `x`, ktorej priradíme požadované hodnoty. Funkcie funguje na základe princípu lineárneho mapovania[34].

```
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 1.4: Zdrojový kód funkcie `mapFloat`.

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

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

Zdrojový kód 1.5: Zdrojový kód funkcie `serialPrint`.

## Popis použitých funkcií z knižnice `AeroShield`

Ked'že na `AeroShielde` využívame senzor hall efektu, musíme s ním v prvom rade nadviazať komunikáciu pomocou sériovej komunikácie  $I^2C$ . Protokol  $I^2C$  využíva na odošielanie a prijímanie údajov dva vodiče resp. dve linky:

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

Hodinový pin udáva tempo komunikácie a je ovládaný **mastrom**. Mení stav v pravidelných impulzoch z low-nízkeho na high-vysoký stav. Pri každej takejto zmene je na dátový pin poslaný jeden bit informácie. Tieto bity najskôr obsahujú adresu zariadenia slave s ktorým chce master komunikovať, následne sa odosielajú bity príkazov. Ked' sa táto informácia celá odošle, slave vykoná požiadavku a ak je to vyžadované, môže späť mastrovi poslať údaje. Všetky tieto bity informácií sa posielajú na linke `SDA`[35].

$I^2C$  funguje na princípe master-slave, kedy master je nadriadený a slave je podriadene zariadenie, s ktorým master komunikuje. Master môže naraz komunikovať s viacerými zariadeniami a to na základe jedinečných adries zariadení, ktoré sa medzi sebou striedajú v komunikácii.

## Funkcia `readOneByte()`

Pre naše účely postačuje vedieť čítať jeden alebo dva bajty informácií. Z toho dôvodu sme vytvorili dve funkcie: `int AeroShield :: readOneByte()` a `word AeroShield :: readTwoBytes()`. Funkcia `int AeroShield :: readOneByte()`, ako jej

názov napovedá, získava 1 bajt informáciu zo senzoru. Túto funkciu využívame napríklad na čítanie polohy kyvadla.

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

    retVal = Wire.read();     // Zaznamenanie odpovede

    return retVal; // Zaslanie odpovede
}
```

Zdrojový kód 1.6: Zdrojový kód funkcie readOneByte.

Ako môžeme vidieť v kóde funkcie, master najskôr osloví zariadenie slave, pomocou jeho adresy. Pri tomto konkrétnom čipe je adresa zariadenia v hexadecimálnej podobe 0x36. Následne zašle od výrobcu predprogramovanú žiadosť, ktorá zaznamená aktuálnu polohu magnetu resp. v našom prípade kyvadla. Následne je komunikácia ukončená a je zaslaná požiadavka na odpoveď zo strany slave zariadenia. Táto odpoveď je zaznamenaná a odoslaná späť, na miesto z ktorého bola funkcia privolaná.

### Funkcia readTwoBytes()

Funkcia word AeroShield :: readTwoBytes() je podobná predošej funkcií, s rozdielom, že získané sú dva bajty informácií narozdiel od jedného a na konci funkcie prebieha posun bitov<sup>7</sup>.

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

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

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

---

<sup>7</sup>Bitový posun je operácia vykonávaná so všetkými bitmi binárnej hodnoty, pri ktorej sa posúvajú o určený počet miest doľava alebo doprava[36].

```

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

    return retVal;             // Zaslanie odpovede
}

```

Zdrojový kód 1.7: Zdrojový kód funkcie readTwoBytes.

### Funkcia detectMagnet()

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

```

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

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

    return retVal;           // Zaslanie odpovede
}

```

Zdrojový kód 1.8: Zdrojový kód funkcie detectMagnet.

### Funkcia getMagnetStrength()

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

```

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 1.9: Zdrojový kód funkcie getMagnetStrength.

### Funkcia getRawAngle()

Poslednou funkciu komunikácie s hall senzorom je `word AeroShield::getRawAngle()`. Funkcia slúži na čítanie samotného uhlu kyvadla. Výsledkom tejto funkcie je číslo s rozsahom 12bitov, teda číslo od 0 do 4096 ktoré udáva momentálnu polohu kyvadla. O tomto senzore, ako aj o jeho fungovaní sme už hovorili bližšie v časti 1.1.1.

```

word AeroShield :: getRawAngle()
{
    return readTwoBytes(_raw_ang_hi, _raw_ang_lo); // Prebieha
                                                    komunikacia so senzorom, ktorý rovno vrati vysledok pomocou
                                                    prikazu return
}

```

Zdrojový kód 1.10: Zdrojový kód funkcie getRawAngle.

### Funkcia begin()

Prvou z funkcií mimo komunikácie s hall senzorom je `float AeroShield::begin(bool isDetected)`, do ktorej vstupuje výsledok z funkcie `detectMagnet()` ako premenná `isDetected`. Funkcia `begin()` nastaví pin potrebný na ovládanie akčného člena, pomocou príkazu `pinMode`, ako výstup, teda `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>8</sup> cyklus, sa pomocou príkazu `break` ukončí. Pokiaľ magnet detegovaný neboli, vypíše "Can not detect magnet", no while cyklus pokračuje.

```

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

```

---

<sup>8</sup>Cyklus while sa bude opakovať nepretržite, pokiaľ sa výraz vnútri zátvoriek () nestane nepravdivým.

```

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

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

```

Zdrojový kód 1.11: Zdrojový kód funkcie begin.

## Funkcia calibration()

Ďalšou v poradí je funkcia `calibration()`. Slúži na prepočet a zaznamenanie nulovej polohy kyvadla, teda takej polohy v ktorej sa kyvadlo nachádza, pokiaľ motorček nie je poháňaný. V ideálnom prípade by sa kyvadlo vždy po vypnutí motora vrátilo do rovnakej východzejcej polohy. Avšak kyvadlo je prepojené s motorom a shieldom pomocou káblov, ktoré tvoria odpor a teda kyvadlo sa vždy zastaví v inej nulovej polohe. Funkcia `calibration()` teda slúži na zaznamenanie tejto nulovej polohy a všetky následné výpočty sa odvolávajú práve na túto hodnotu.

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

```

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;             // Ulož 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 1.12: Zdrojový kód funkcie calibration.

### Funkcia convertRawAngleToDegrees()

Ako sme už spomínali v sekcií 1.1.1, zaznamenaná hodnota uhlu kyvadla je v rozmedzí od 0 do 4096 a tieto hodnoty priamo korešpondujú zo stupňami od  $0^\circ$  do  $360^\circ$ .  $1^\circ$  predstavuje hodnotu približne 11.77 vo formáte raw. Funkcia convertRawAngleToDegrees() teda len zoberie raw hodnotu uhlu a vynásobí ju hodnotou  $\frac{360}{4096} = 0.087^\circ$ . Funkcia teda naspäť vráti prepočítanú hodnotu uhla kyvadla v stupňoch.

```

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

```

Zdrojový kód 1.13: Zdrojový kód funkcie convertRawAngleToDegrees.

### Funkcia referenceRead()

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

```

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

```

Zdrojový kód 1.14: Zdrojový kód funkcie referenceRead.

### Funkcia actuatorWrite()

Na ovládanie motora resp. jeho otáčok, používame funkciu actuatorWrite(). Do funkcie vstupuje percentuálna hodnota výkonu motora, táto hodnota je premapovaná na PWM signál, potrebný na správne 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. Táto hodnota je potom zapísaná na príslušný pin motora, v našom prípade je to AERO\_UPIN.

```

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

```

Zdrojový kód 1.15: Zdrojový kód funkcie actuatorWrite.

### Funkcia currentMeasure()

Poslednou funkciou AeroShieldu je currentMeasure(). Funkcia slúži na zaznamenávanie prúdu, ktorý odoberá motor kyvadla. Fungovanie snímača je opísané v sekcií 1.1.1. Funkcia slúži na prepočítanie zaznamenanej hodnoty napäťa na prúd. Funkcia beží vo for cykle, ktorý sa opakuje repeatTimes-krát, z dôvodu presnejšieho merania, keďže meraná hodnota sa priveľa mení, čo skresluje výslednú hodnotu. Vďaka for cyklu získame priemernú hodnotu prúdu.

Výsledná hodnota prúdu prechádza úpravami, pomocou dvoch korekčných premenív, ktoré sme získali vďaka meraniam prúdu pomocou multimetra, a následným porovnaním hodnôt zo senzora. Porovnaním hodnôt sme získali veľkosť korekčných premenív, vďaka čomu sa naše merania zhodujú s meraniami pomocou multimetra, na dve desatinné miesta.

Nepresnosť vzniká v dôsledku ovládania motora PWM signálom. Prúd je teda prerušovaný a jeho meraná hodnota v čase skáče medzi hodnotou skutočnou a nulovou. Musíme preto vykonávať viacero meraní, ktoré sú sčítavaní. Z tejto sčítanej hodnoty nameraných prúdov získame priemernú hodnotu prúdu, na ktorú sú následne aplikované korekčné premenné.

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, ktorý priamo premieňa prúd na napätie.

Kedže prúd nemôže vykazovať záporné hodnoty (iba pokiaľ prúdi opačným smerom, ako sme zamýšľali), na záver funkcie ešte prechádza if podmienkou, ktorá zmení záporné hodnoty prúdu na hodnotu 0.0A.

```

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 napatие 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 1.16: Zdrojový kód funkcie currentMeasure.

## 1.2.2 MATLAB

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

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

Každá trieda môže následne obsahovať jeden alebo viacero triednych blokov:

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

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

```

classdef AeroShield < handle

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

```

```

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

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

```

Zdrojový kód 1.17: Knižnica AeroShield.m properties.

Nasleduje triedy blok **methods**, v ktorom sú všetky funkcie, ktoré budeme volať do hlavnej časti kódu. V bloku **methods** sa nachádzajú jednotlivé funkcie, ktoré definujeme ako **function nazovFunkcie(AeroShieldObject)** a za týmto zápisom, píšeme samotné príkazy. Keďže logika funkcií je rovnaká ako pri súbore **Source**, nebudeme si funkcie vysvetľovať po jednom. Dôležité je poznamenať, že pokiaľ nám do funkcie nejaká premenná vstupuje, zapisujeme ju v zátvorke za príkazom funkcie **function nazovFunkcie (AeroShieldObject , vstupnaPremenna)**. Naopak, pokiaľ z funkcie nejaká premenná vychádza, túto premennú deklarujeme pred názvom funkcie **function vystupnaPremenna = nazovFunkcie (AeroShieldObject)**. Pre pochopenie si ešte ukážeme časť kódu **function begin (AeroShieldObject)**, v ktorej prebieha tvorba objektov pre arduino a hall senzor. Proces odvolávania sa na objekt **arduino**, funguje cez názov triedy **AeroShield**, v ktorom sa odvolávame na **Object**. Príkaz **arduino()** slúži na vyhľadanie dosky arduino, pripojenej do počítača a nastavenie parametrov potrebných na komunikáciu. 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 1.18: Knižnica AeroShield.m properties.

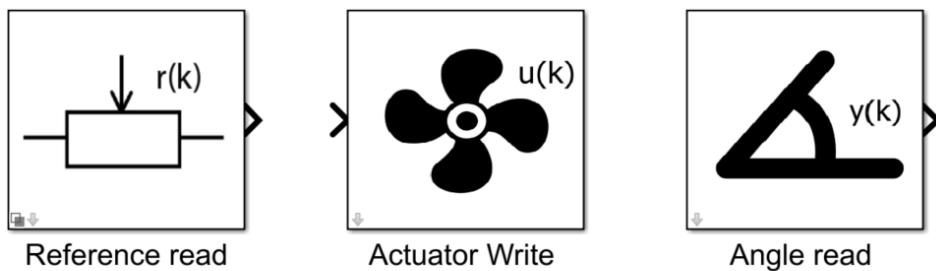
Zostávajúce funkcie, ako aj celý zdrojový kód **AeroShield.m** sa nachádza v prílohe na strane x.

### 1.2.3 Simulink

Simulink, narozený od MATLABU alebo Arduino IDE, využíva grafické programátor-ské prostredie, ktoré je založené na prepájaní funkčných blokov, vyberaných z prostredia knižnice. Pre prácu s Arduinom, v rámci Simulinku, existuje knižnica *Simulink Support Package for Arduino Hardware*, ako aj knižnica AeroLibrary obr.1.13, 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 reguloowanej veličiny.

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



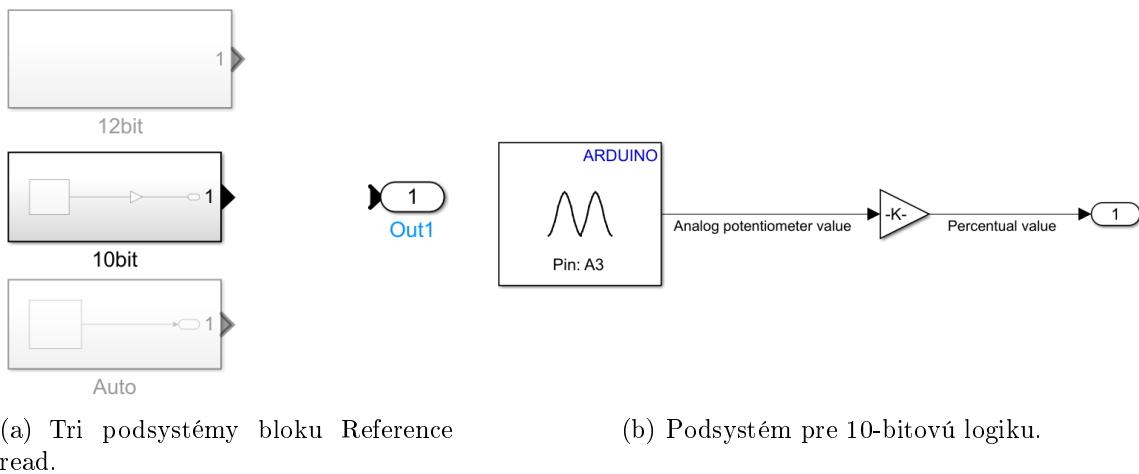
Obr. 1.13: Knižnica AeroLibrary.

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 posiela 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.1.14 sú takmer totožné. Rozdiel medzi nimi je taký, že pri použití niektorých druhov Arduina<sup>11</sup> je pri použití bloku Analog input, výstup v tvare 12-bitového čísla, narozený od 10-bitového, ako je tomu pri ostatných podporovaných modeloch Arduina. Načítaná hodnota je ďalej upravovaná do percentuálnej podoby pomocou pre násobenia konštantou v bloku gain.

Podsystém pre automatickú trajektóriu obr.?? má v sebe prednastavené pole referenčných hodnôt.



Obr. 1.14: Reference read- Simulink.

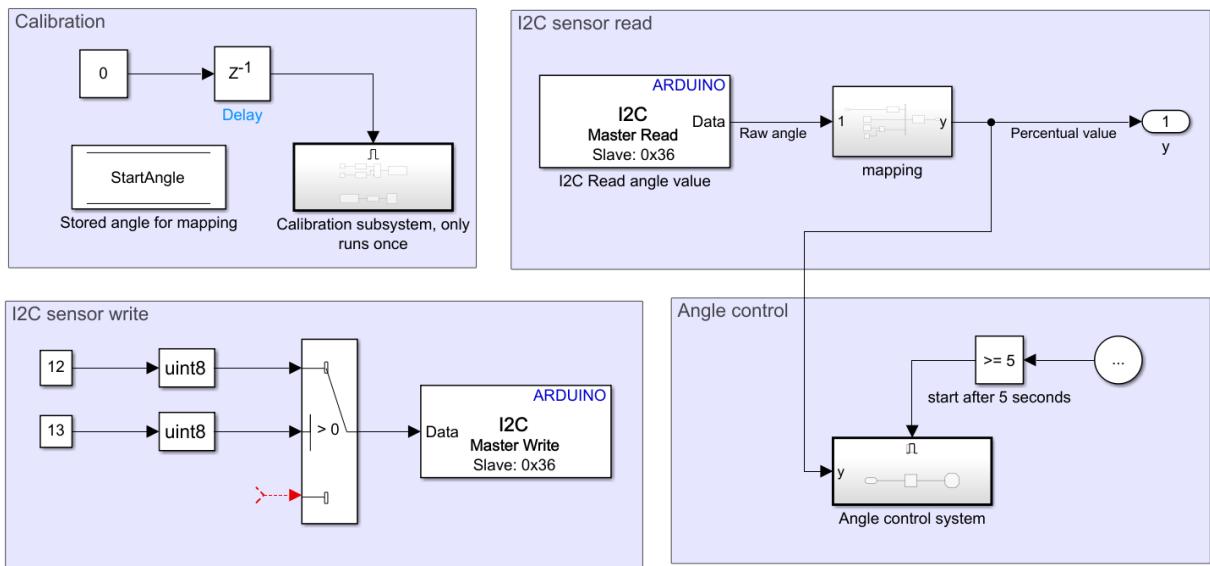
## Angle read

Tento blok obr.1.15 slúži na čítanie uhlu kyvadla, ako aj pre bezpečnostnú kontrolu uhlu kyvadla. Jeho vnútorná štruktúra sa dá rozdeliť na štyri menšie celky.

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

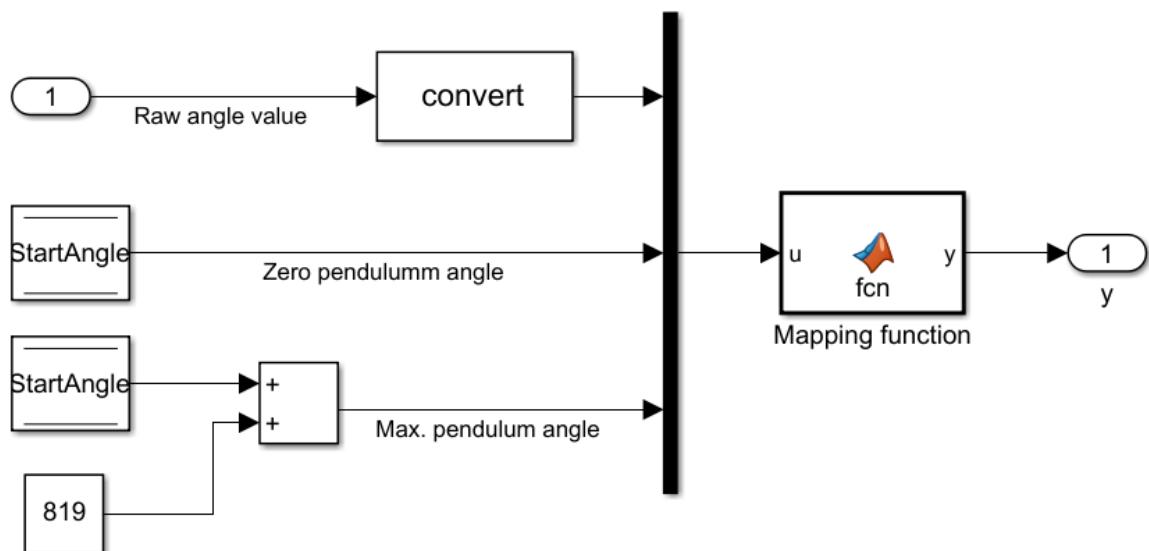
**I2C zápis** Blok I2C Write obr.1.15(ľavý dolný roh), z knižnice *Simulink Support Package for Arduino Hardware*, zapisuje na senzor striedavo hodnoty 12 a 13, ktoré slúžia ako príkaz pre čítanie a následné odoslanie hodnoty pootočenia kyvadla. Prepínanie medzi týmito dvomi hodnotami zabezpečuje automatický prepínač (Automatic switch).

<sup>11</sup>Due, MKR1000, MKR1010, MKRZero, Nano 33IoT.



Obr. 1.15: Angle read- Simulink.

**I2C čítanie** Samotné čítanie uhlu zabezpečuje blok I2C Read obr.1.15(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.1.16. Mapovanie funguje na rovnakom princípe ako v príklade 2.1.2. Jednotlivé premenné vchádzajú do bloku MUX, ktorý slúži ako multiplexer<sup>12</sup>. Signál z multiplexoru vstupuje do fcn bloku, ktorý prepája Simulink s MATLABOM. Zdrojový kód 1.19 zobrazuje funkciu vo vnútri tohto bloku.



Obr. 1.16: Mapovanie uhlu kyvadla- Simulink.

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

```

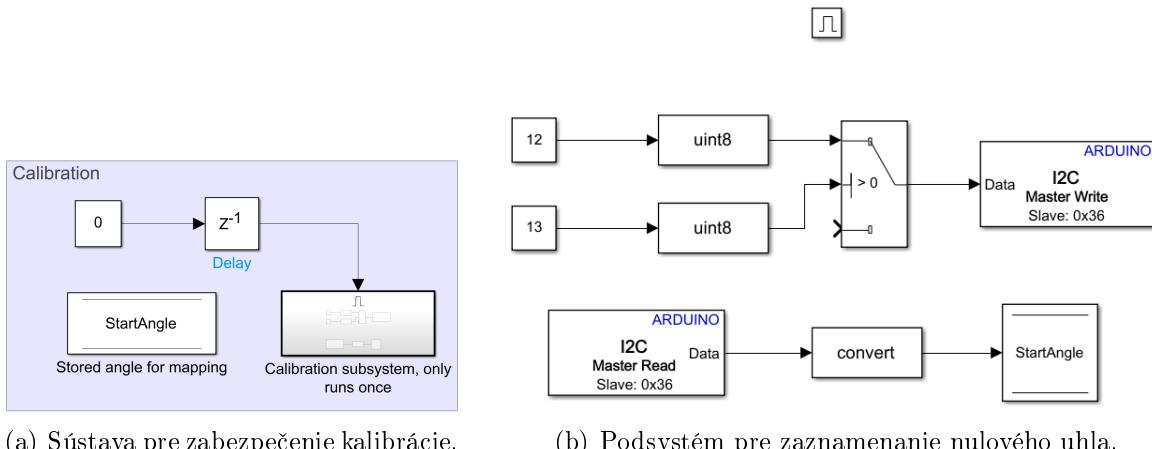
function y = fcn(u)
angMin=0;
angMax=72;
y = ((u(1) - u(2)) * (angMax - angMin)) / (u(3) - u(2)) + angMin ;

```

Zdrojový kód 1.19: Mapovacia funkcia vo fcn bloku.

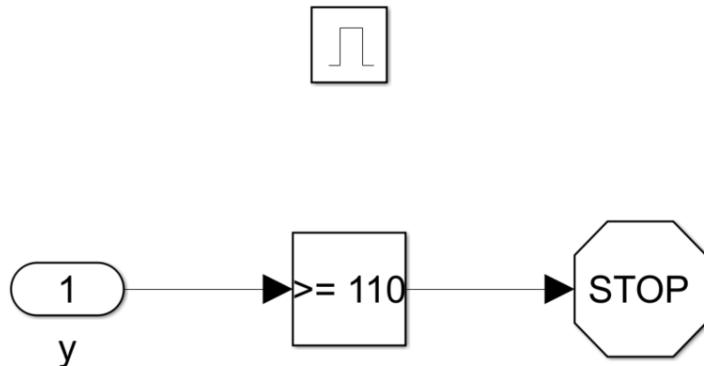
Proces získania premennej **StartAngle**, ktorú využívame pri mapovaní, je opísaný v nasledujúcim odstavci.

**Kalibrácia** Kalibrácia prebieha vždy len raz a to pri spustení simulácie. V podsystéme (Enabled Subsystem) obr.1.17.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.1.17.b.



Obr. 1.17: Kalibrácia- Simulink.

**Kontrola uhlia** Kontrola uhlia kyvadla prebieha v podsystéme, ktorý je spustený päť sekúnd po začatí simulácie obr.1.15(pravý dolný roh). Časový posun je použitý z dôvodu občasného šumu premenných počas začiatku simulácie. Tento šum môže spôsobiť nechcené výkyvy hodnoty uhlia kyvadla, ktoré by splnili podmienku v podsystéme a tým by zastavili priebeh simulácie. V podsystéme na obr.1.18, sa nachádza blok **Compare To Constant**, ktorý porovnáva hodnotu uhlia kyvadla, s daným maximálnym uhlom kyvadla. Táto hodnota je v našom prípade  $110^\circ$ . Pokiaľ kyvadlo dosiahne väčší uhol, ako je uhol dovolený, blok **Stop Simulation** zastaví simuláciu.



Obr. 1.18: 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 nápovedami. Vďaka maske blokov, sa tieto viac podobajú na bloky, ktoré nájdeme v predprogramovaných knižniciach Simulinku. Masku upravujeme pomocou editora, ktorý spustíme pravým kliknutím na blok, s následným výberom možnosti **Mask**. V maske sa nachádzajú štyri hlavné editovacie rozhrania.

Prvým z nich je záložka **Icon & Ports**. V tomto okne upravujeme hlavné vzhľad bloku, teda jeho zobrazenie, rotáciu, inicializáciu alebo aj ikona bloku. Pre všetky bloky v knižnici AeroLibrary, boli vytvorené vlastné ikony, ktoré sa do masky vkladajú nasledujúcim 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.1.19, 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. Jedná sa o roletové, alebo "popup"menu, v ktorom si vyberieme typ dosky s ktorou používame AeroShield. Dôvod tohto výberu je opísaný v časti 1.2.3, na strane 30. Pokiaľ si zvolíme Automatickú referenčnú trajektóriu, možnosť voľby dosky sa nezobrazí. Táto funkcia je vykonávaná automaticky, zdrojovým kódom 1.20, ktorý sa nachádza v časti **Callback**, v nastaveniach tlačidiel voľby trajektórie.

```

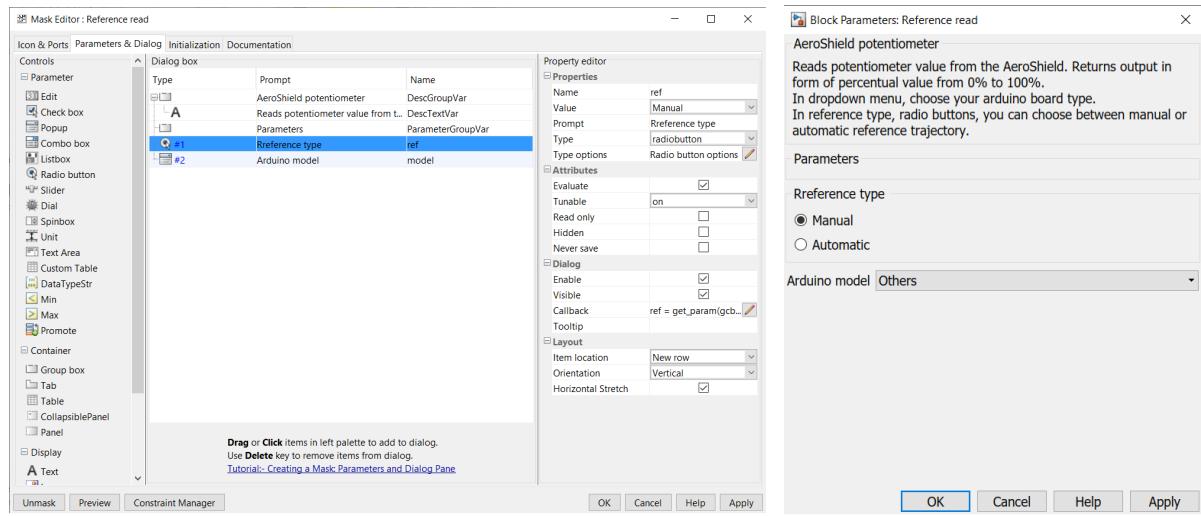
ref = get_param(gcb,'ref');
if strcmp(ref(1),'M')
set_param(gcb,'MaskVisibilities',{ 'on'; 'on' }) ,
else
set_param(gcb,'MaskVisibilities',{ 'on'; 'off' }) ,
end

```

Zdrojový kód 1.20: 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` kontroluje ktorú z možností sme si vybrali, pomocou príkazu `strcmp`, ktorý porovnáva prvé písmeno z premennej `ref`, s písmenom 'M' (Manual). Pokiaľ sa tieto zhodujú, príkaz `set_param`, spolu s príkazom `MaskVisibilities`, nastaví obe polia tlačidiel ako viditeľné, teda s hodnotou 'on'. Pokiaľ sa písmená nezhodujú a teda bola vybraná trajektória automatická, možnosť výberu dosky Arduino, je skrytá príkazom 'off'.



Obr. 1.19: Reference read- Simulink.

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 1.21. Podľa pravdivosti `if` podmienky, príkaz `set_param(gcb, 'LabelModeActiveChoice')` aktivuje podsystém bloku, ktorého názov je uvedený ako tretí parameter v zátvorke príkazu.

```

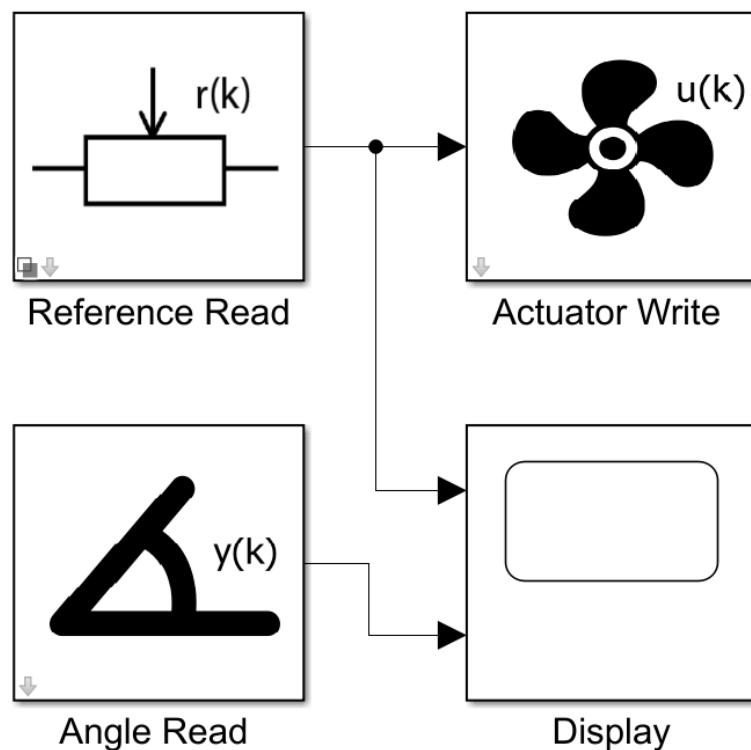
ref = get_param(gcb, 'ref');
model = get_param(gcb, 'model');
if strcmp(ref, 'Manual')
if strcmp(model, 'Due')
set_param(gcb, 'LabelModeActiveChoice', '12bit');
elseif strcmp(model, 'MKR1000')
set_param(gcb, 'LabelModeActiveChoice', '12bit');
elseif strcmp(model, 'MKR1010')
set_param(gcb, 'LabelModeActiveChoice', '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 1.21: Initialization- maska Reference read.

Pre lepšie porozumenie blokov a ich funkcií, bol v API Simulink zostavený inštrukčný príklad **AeroShieldOpenLoop** obr.1.20. Tento príklad funguje na princípe merania hodnoty bežca potenciometra na Shielde, a následného zapisovania nameranej hodnoty na akčný člen sústavy. Zároveň je meraný uhol v ktorom sa kyvadlo nachádza. Obe tieto hodnoty sú priebežne zobrazované na grafe.



Obr. 1.20: AeroShield\_OpenLoop.

## 2 Didaktické príklady

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

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

### 2.1 Programy v otvorenej slučke, bez späťnej väzby

#### 2.1.1 Arduino IDE

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

```
#include "AeroShield.h"           // Inicializacia hlavnej kniznice

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

Zdrojový kód 2.1: AeroShield open loop dekleracia.

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

```
void setup() {                      // Setup prebehne len jeden krat
    Serial.begin(115200);             // Zaciatok seriovej komunikacie
```

```

AeroShield.begin(); // Inicializacia AeroShieldu
startangle = AeroShield.calibration(AeroShield.getRawAngle()); // 
    Kalibracia kyvadla
lastangle=startangle+1024; // Kalkulacia uhlu kyvadla pre map function
}

```

Zdrojový kód 2.2: AeroShield open loop setup().

V časti `loop()` sa program opakuje dookola. Ako prvé, prebehne mapovanie uhlu kyvadla pomocou funkcie `AutomationShield.mapFloat()` a získaná hodnota uhlu sa vypíše na sériový monitor, spolu s názvom a premennou danej veličiny. Nasleduje čítanie hodnoty potenciometra, ktorá slúži na ovládanie akčného člena pomocou funkcie `AeroShield.actuatorWrite()`. Na sériový port sa vypíše hodnota potenciometra obr.2.1, za ktorou nasleduje veľkosť prúdu odoberaného motorom `AeroShield.currentMeasure()`.

```

void loop() {
    pendulumAngle= AutomationShield.mapFloat(AeroShield.getRawAngle(),
                                                startangle ,lastangle ,0.00 ,90.00); // Mapovanie uhlu kyvadla
    Serial.print("pendulum angle is: ");
    Serial.print(pendulumAngle);
    Serial.print("Degrees || ");

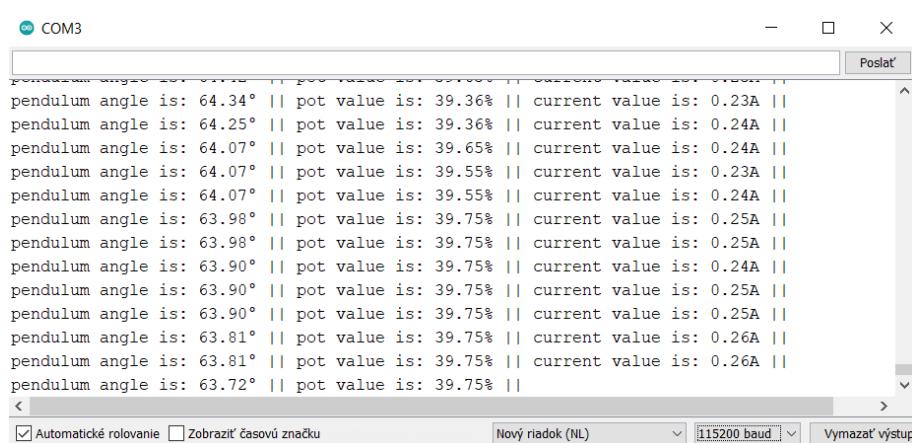
    referencePercent= AeroShield.referenceRead(); // Citanie
    potenciometra
    Serial.print("pot value is: ");
    Serial.print(referencePercent);
    Serial.print("% || ");

    AeroShield.actuatorWrite(referencePercent); // Pohyb akcneho clenu

    CurrentMean= AeroShield.currentMeasure(); // Meranie prudu
    Serial.print("current value is: ");
    Serial.print(CurrentMean);
    Serial.println("A || ");
}

```

Zdrojový kód 2.3: AeroShield open loop loop().



Obr. 2.1: Výstup z programu AeroShieldOpenLoop.ino.

## 2.1.2 MATLAB

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

Na začiatku kódu vymazame všetky premenné a objekty, pomocou súriedu príkazov `Clear all`, `clear a`, `clc`. Následne načítame knižnicu AeroShieldu a vykonáme funkciu `AeroShield.begin()`. Nasleduje kalibrácia nulového uhlu kyvadla a zadefinovanie premenných na počítanie času, ako aj premenné na ukladanie hodnôt potenciometra a uhlu kyvadla.

```
% vymazanie premennych a objektov
clear all;
clear a;
clc

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

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

Zdrojový kód 2.4: AeroShield open loop inicializacia.

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

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

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

```

legend( 'Potentiometer value' , 'Pendulum angle' ) % legend for plots
grid( 'on' ); % grid for plot 'off' to turn off
grid

tic % time keeping

```

Zdrojový kód 2.5: AeroShield open loop grafy.

Nasleduje while cyklus ktorý má podmienku ukončenia zatvorenie vykreslovaného grafu tj. v momente kedy zatvoríme graf, podmienka prestane byť splnená a program sa ukončí. V cykle najskôr čítame hodnotu potenciometra pomocou `AeroShield.referenceRead()` a túto hodnotu zapisujeme na akčný člen vďaka príkazu `AeroShield.actuatorWrite()`. Nasleduje čítanie uhlu kyvadla `AeroShield.getRawAngle()`, za ktorím prebehne mapovanie premennej z hodnoty raw na stupne. Premenná `count` slúži na počítanie počtu prejdencích cyklov, ako aj na tvorbu usporiadaneho radu premenných, a využívame ju 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. Ak ten nadobudne hodnotu väčšiu ako  $110^\circ$ , proces sa automaticky ukončí a vypíše sa upozornenie. Posledný príkaz `clear AeroShield.arduino` vymaže objekt `arduino` a pripraví MATLAB na spustenie ďalšieho programu.

```

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

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

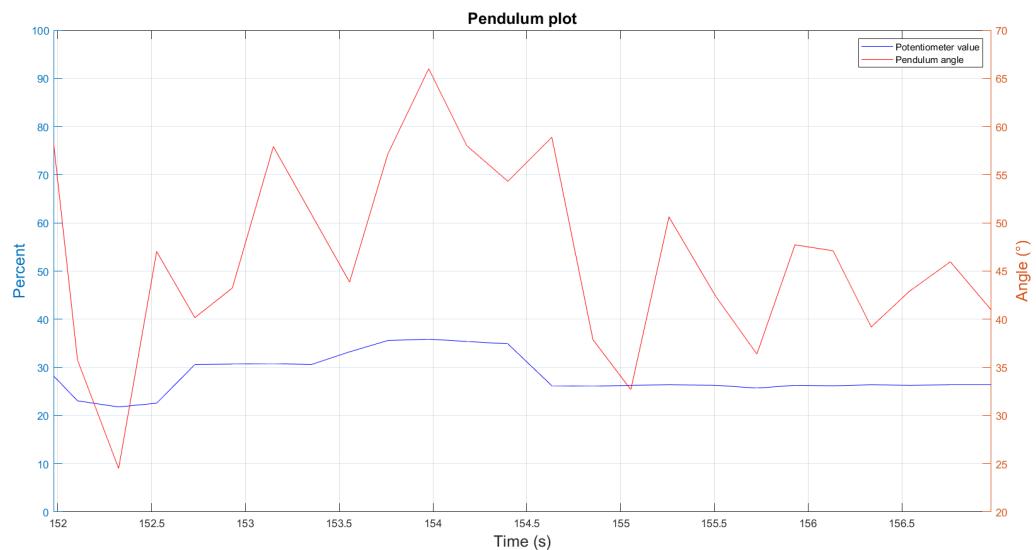
    count = count + 1; % cycle counter
    time(count) = toc; % time keeping
    angle(count) = angle_(1); % angle value in
    time
    percenta= mapped(pwm, 0.0, 5.0, 0.0, 100.0); % map pwm to
    percent
    potentiometer(count) = percenta(1); % pententiometer
    value in time
    set(plotGraph, 'XData',time,'YData',angle);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

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

```

```
clear AeroShield.arduino;
```

Zdrojový kód 2.6: AeroShield open loop, while cyklus.

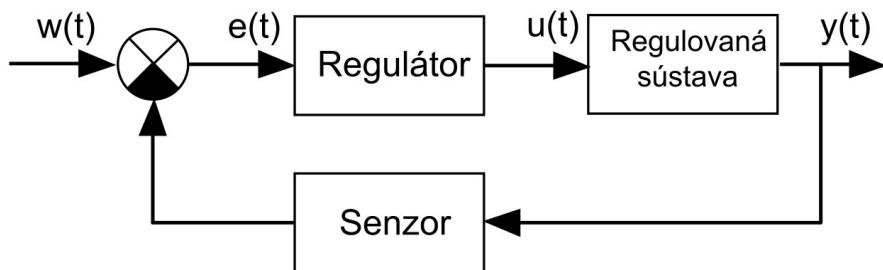


Obr. 2.2: Výstup z programu AeroShieldOpenLoop.m.

## 3 PID regulácia

Skôr ako si ukážeme príklady na PID riadenie, musíme si vysvetliť ako takéto riadenie funguje. Základom takého riadenia 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 práve nachádza. Pri takomto riadení existuje množstvo algoritmov, ktoré ovládajú správanie sa systému. Medzi tieto algoritmy patrí napríklad: Modelové prediktívne riadenie (MPC), Lineárno-kvadratické riadenie (LQ), Lineárne riadenie s premenlivým parametrom (LPV), Proporcionálno-integračno-derivačné riadenie (PID)...

Bližšie sa ideme baviť o PID regulátore obr.3.1. Je to typ riadenia, ktoré ovplyvňuje akčné zásahy do sústavy  $u(t)$ , na základe zaznamenaných výstupných informácií zo sústavy  $y(t)$ . Veľkosť akčného zásahu  $u(t)$  vypočítame na základe rozdielu medzi požadovanou hodnotou  $w(t)$  a hodnotou reálnej  $y(t)$ . Tento rozdiel označujeme tiež ako regulačná odchýlka  $e(t)$ . Písmeno "t" v zátvorkách predstavuje, časovú závislosť premenných.



Obr. 3.1: Schéma riadenia PID regulátorom.

Skratka PID odzrkadľuje zložky, z ktorých sa regulátor skladá:

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

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

Konštanta  $K_p$  je veľmi dôležitou pri nastavovaní parametrov PID regulátora, lebo znižuje trvalú regulačnú odchýlku. Zmenou proporcionálnej zložky vieme taktiež urýchliť, alebo spomaliť nábeh na požadovanú hodnotu. Pre malé hodnoty  $K_p$  je nábeh pomalý, a regulačná odchýlka veľká. Pri zvyšovaní hodnoty  $K_p$  sa regulačná odchýlka zmenšuje, zrýchluje sa nábeh, no zároveň narastá nežiadúce kmitanie sústavy. Zvyšovanie hodnoty  $K_p$  má svoj limit, za ktorým sa sústava dostáva za hranicu stability a ďalšia regulácia nie je možná.

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

Veľkosť integrálnej zložky PID regulátora teda ovplyvňuje veľkosť regulačnej odchýlky, doba jej trvania ako aj integračná konštanta  $K_i$ .

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

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

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

Pre lepšie chápanie je v tab.3.1[?] zhrnutie vplyvu jednotlivých zložiek PID na odozvu systému v uzavorennej slučke, pri zvyšovaní hodnoty  $K_p$ ,  $K_i$  a  $K_d$ .

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

Tabuľka 3.1: Odozva systému na zmenu konštánt.

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

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

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

Rovnica 3.4 predstavuje jeden z možných tvarov zápisu vzťahu pre PID regulátor. V praxi sa často využíva tvar rov.3.5, z dôvodu lepšej interpretácie parametrov ktoré 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)[?].

Spomenuté vzťahy PID regulátorov fungujú pri spojitých procesoch. Avšak pri implementácii PID pomocou číslicových regulátorov, musíme rovnicu transformovať do jej diskrétnej podoby rov.3.6.

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

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

## 3.1 Programy v uzavorennej slučke, so spätnou väzbou

### 3.1.1 Arduino IDE

V tomto príklade využívame na riadenie PID regulátora, vopred pripravenú knižnicu **PIDAbs**, ktorá je volaná z knižnice **AeroShield**. Za účelom vzorkovania, využívame knižnicu **Sampling**, ktorú načítame do príkladu príkazom `#include <Sampling.h>`. Pri voľbe referenčnej trajektórie máme na výber z dvoch možností. Prvou je voľba manuálnej trajektórie, ktorej referenčnú hodnotu nastavujeme pomocou potenciometra na shielde. Druhou voľbou je automatická trajektória, ktorá má vopred naprogramované referenčné hodnoty. Voľbu trajektórie meníme zmenou hodnoty premennej **MANUAL** z 0 na 1 v príkaze `#define MANUAL 0`. Parametre regulátora  $K_p$ ,  $T_i$  a  $T_d$  slúžia ako symbolické parametre, pre lepší prehľad. Ich hodnotu zapisujeme do PID knižnice pomocou metódy **PIDAbs.setKp(KP)**. Vzorkovacia periódia **Ts** je nastavená na hodnotu troch milisekúnd. Periódou 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.setKp(KP);              // Nastavenie Kp
    PIDAbs.setTi(TI);              // Nastavenie Ti
    PIDAbs.setTd(TD);              // Nastavenie Td
    PIDAbs.setTs(Sampling.samplingPeriod); // Vzorkovacia perioda
    Sampling.interrupt(stepEnable); // Nasatavenie nazvu funkcie stepEnable, v kniznici sampling
}

```

Zdrojový kód 3.2: Organizačná funkcia setup.

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

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

```

void stepEnable() {
    if(nextStep == true) {           // Pokial predosla vzorka stale trva
        realTimeViolation = true; // Nastala chyba vzorkovania
        Serial.println("Real-time samples violated.");
        // Vypis chybovu hlasku
    }
}

```

```

        analogWrite(5,0);           // Vypni motor
        while(1);                  // Ukonci vykonavanie programu
    }
    nextStep = true;             // Povol nasledujucu vzorku
}

```

Zdrojový kód 3.3: Funkcia stepEnable().

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

```

void loop() {
    if (pendulumAngle > 120){           // Bezpecnostna podmienka kyvadla
        AeroShield.actuatorWrite(0); // Pokial je uhol vacsi ako
        120
        while(1);                   // stupnov, motor sa vypne
    }
    if (nextStep) {                   // Pokial nextStep == 1
        step();                      // Spusti fUnciu step()
        nextStep = false;            // 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 pred programovanej trajektórie. Táto kontrola je vykonávaná pomocou porovnávania veľkosti premennej `i`, ktorá zaznamenáva počet vykonaných sekcií trajektórie, oproti veľkosti pola `R[]`, v ktorom sú zapísané hodnoty jednotlivých sekcií. Príkaz `sizeof(R)` / `sizeof(R[0])`, vráti počet prvkov pola `R[]`. Zároveň sa kontroluje dĺžka chodu sekcie. Pokiaľ výraz `k % (T*i)`, dosiahne hodnotu 0, nastaví sa ako trajektória nasledujúca sekcia.

Následne je mapovaný uhol kyvadla na percentuálnu hodnotu od 0% do 100%, ktorá je uložená ako premenná `y`. Veľkosť regulačnej odchýlky, obmedzenie integračného nasýtenia<sup>9</sup> (angl. anti-windup), ako aj hodnotu saturácie systému<sup>10</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

```

<sup>9</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 nekontrolovačne stúpať.

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

```

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 postup o jednu sekciu
}
#endif

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

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

```

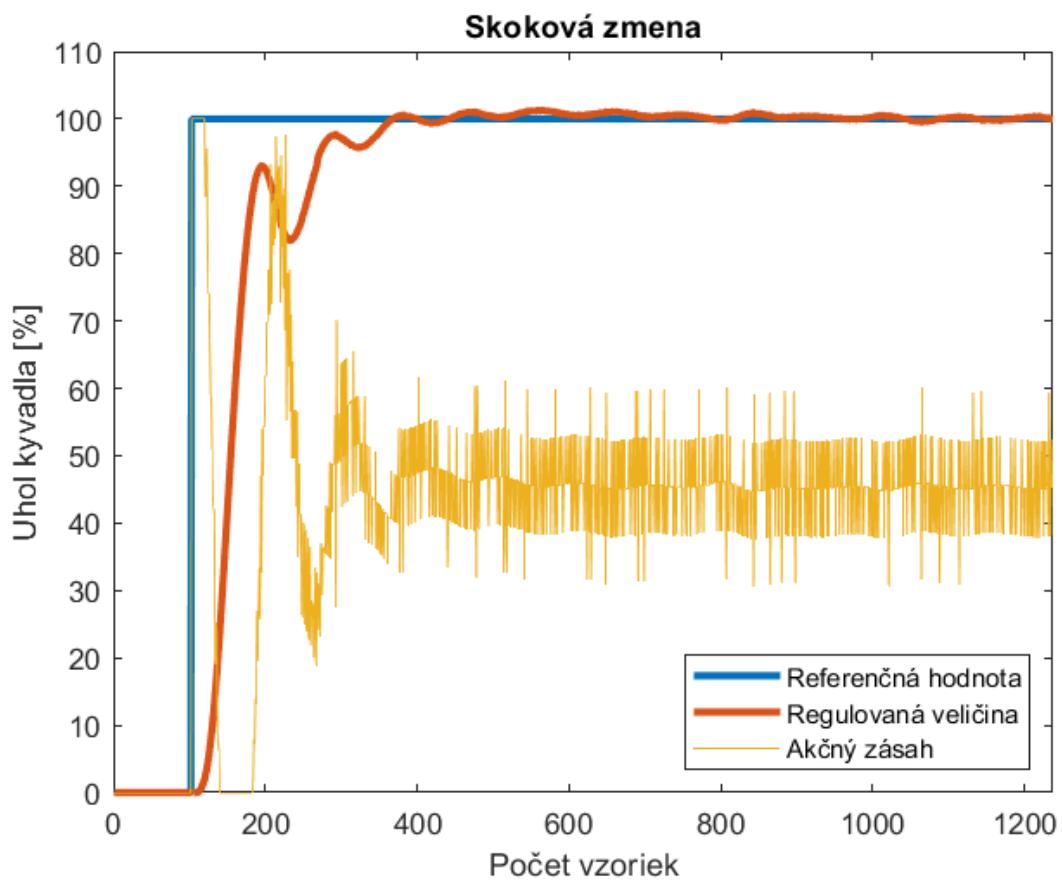
Zdrojový kód 3.5: Funkcia step().

## Výstupy

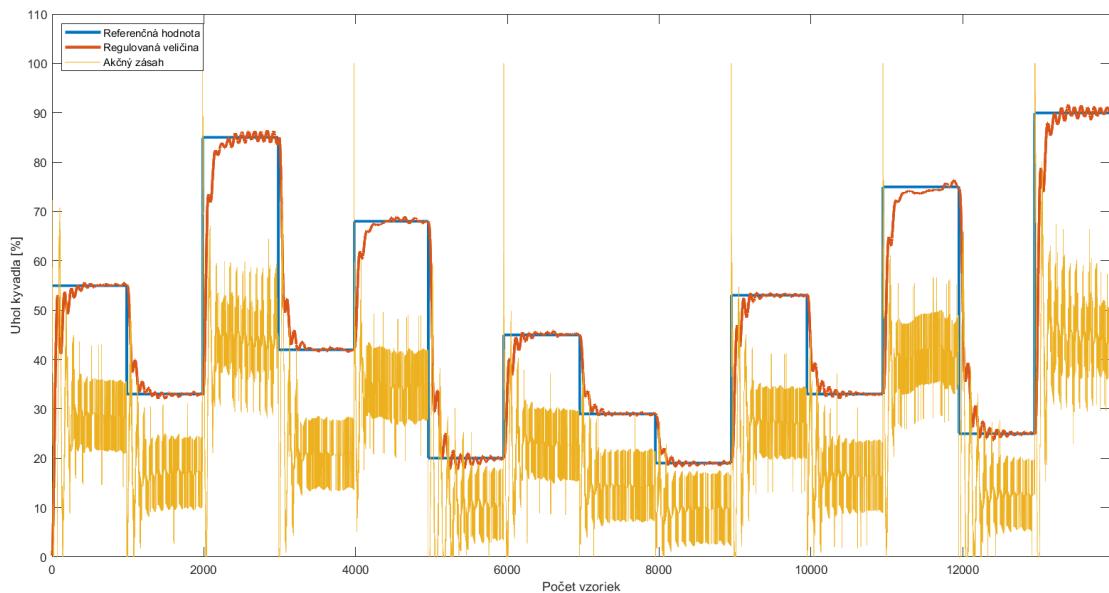
Všetky výstupy z API Arduino IDE, boli zaznamenávané programom CoolTerm a následne vykreslené do grafov v prostredí MATLAB.

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

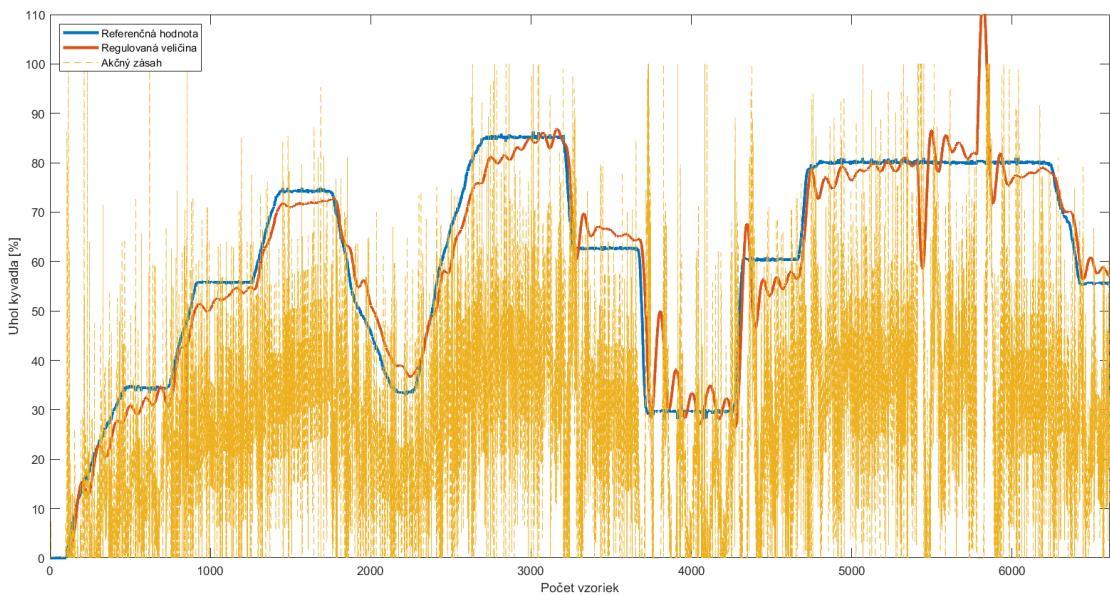
Manuálna trajektória obr.3.4, má oproti automatickej trajektórii obr.3.3, väčšie rozdiely v hodnote akčného zásahu. Je to spôsobené kontinuálnou zmenou referenčnej hodnoty, ako aj miernym šumom signálu z potenciometra. V rámci obr.3.4 si ešte môžeme všimnúť časť medzi vzorkami 5000-6000, ktorá je priblížená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



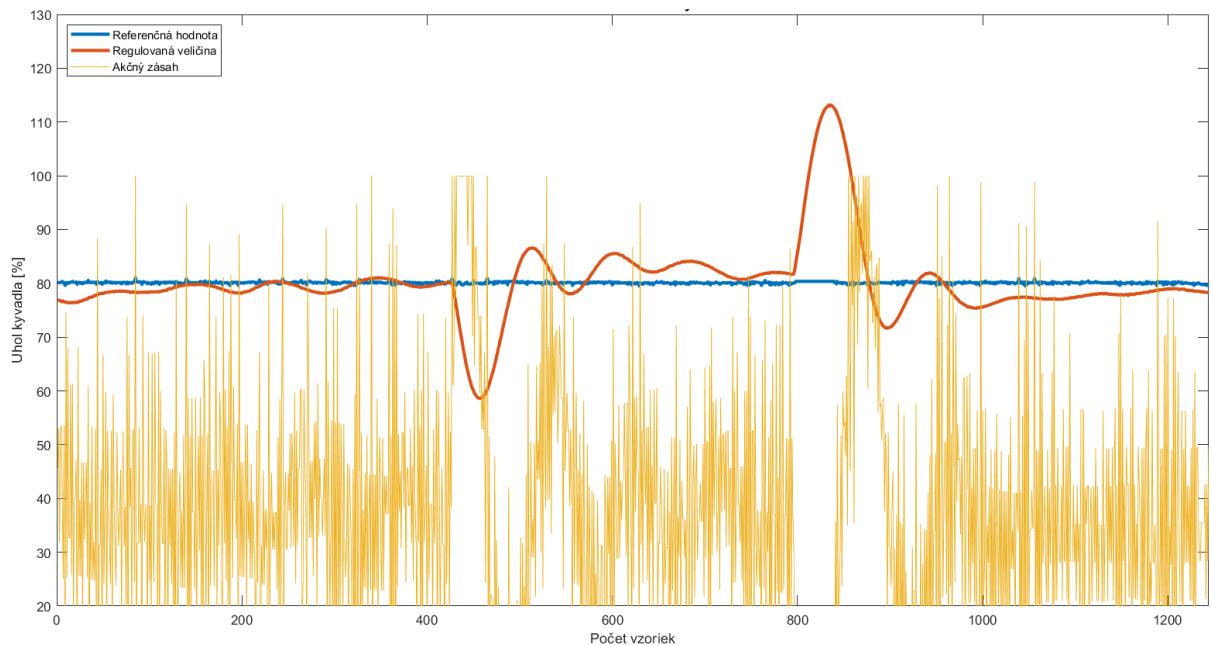
Obr. 3.2: Reakcia systému na jednotkový skok.



Obr. 3.3: Automatická trajektória.



Obr. 3.4: Manuálna trajektória.



Obr. 3.5: Manuálne zavedenie výchylky.

### 3.1.2 MATLAB

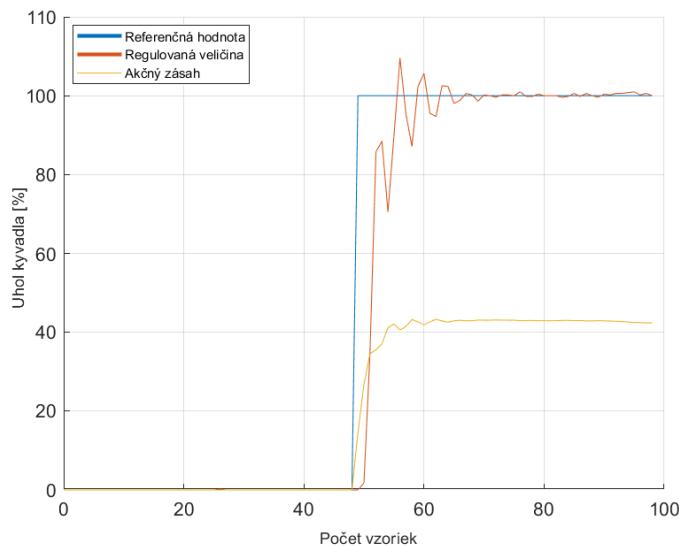
Príklad na ukážku fungovania PID regulátora bol taktiež vytvorený v prostredí MATLAB a simulink. Princíp fungovania PID regulátora je rovnaký ako pri Arduino IDE, avšak mení sa spôsob vzorkovania a nastavovania parametrov PID. V prípade MATLABU využívame, na výpočet akčného zásahu, knižnicu PID.m, ktorá bola taktiež vytvorená v rámci programu AutomationShield. Na nastavenie parametrov PID slúži príkaz PID.setParameters(Kp, Ti, Td, Ts), Ts udáva vzorkovaciu periódu.

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 dátá uložené, a teda sú prístupné aj po ukončení programu. Kompletný zdrojový kód sa nachádza v prílohe na strane xii.

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

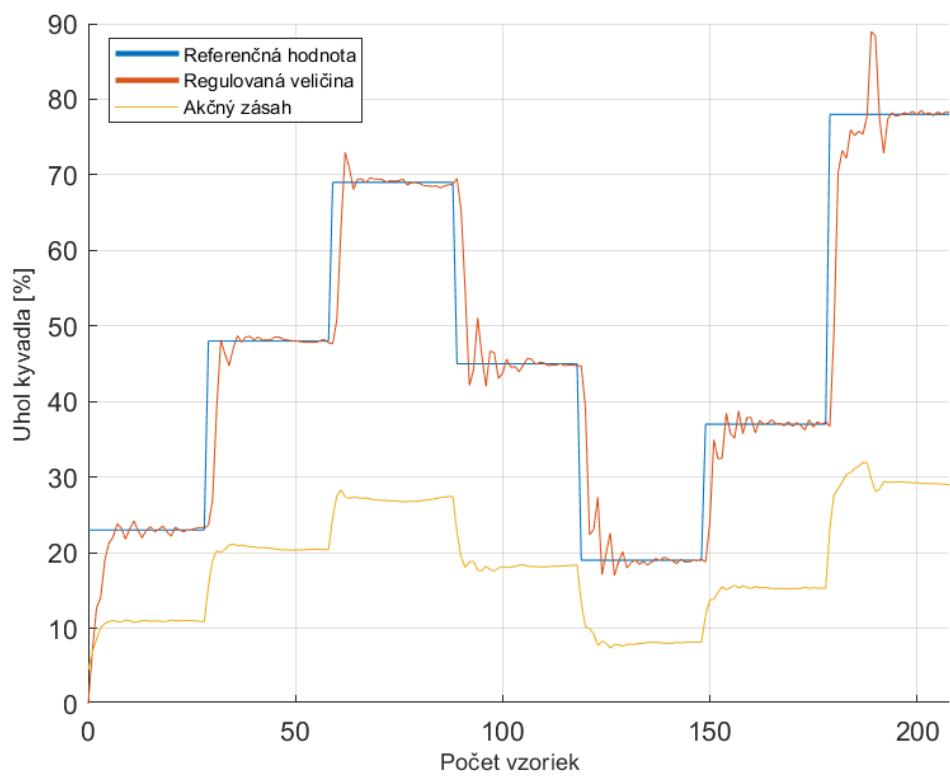
## Výstupy

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

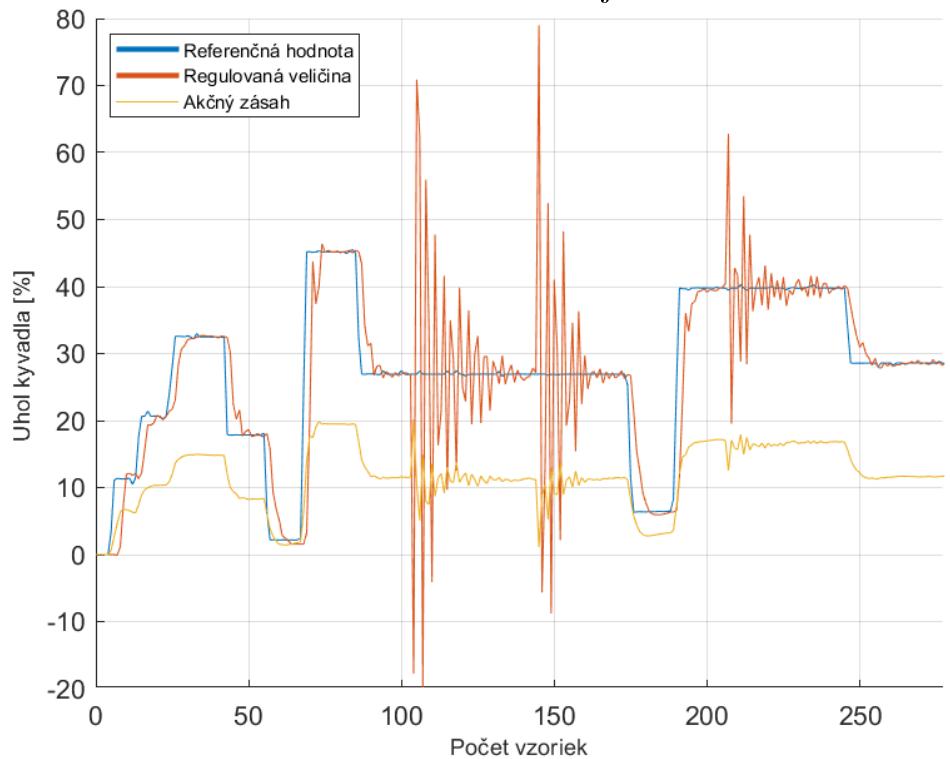


Obr. 3.6: Reakcia systému na jednotkový skok.

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



Obr. 3.7: Automatická trajektória.

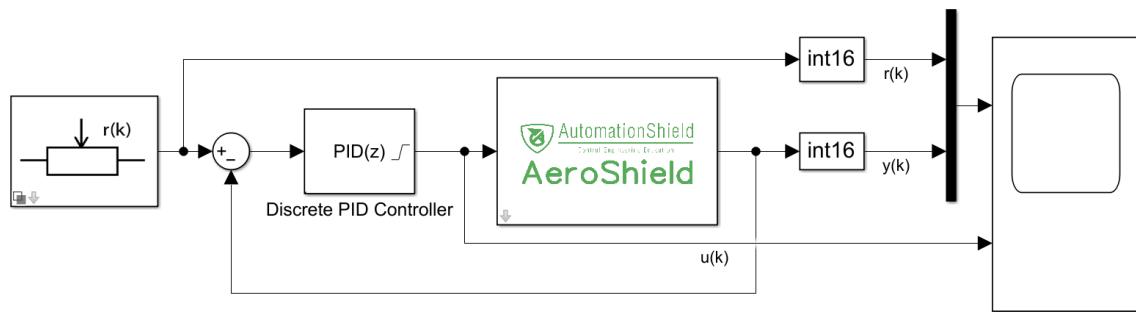


Obr. 3.8: Manuálna trajektória.

### 3.1.3 Simulink

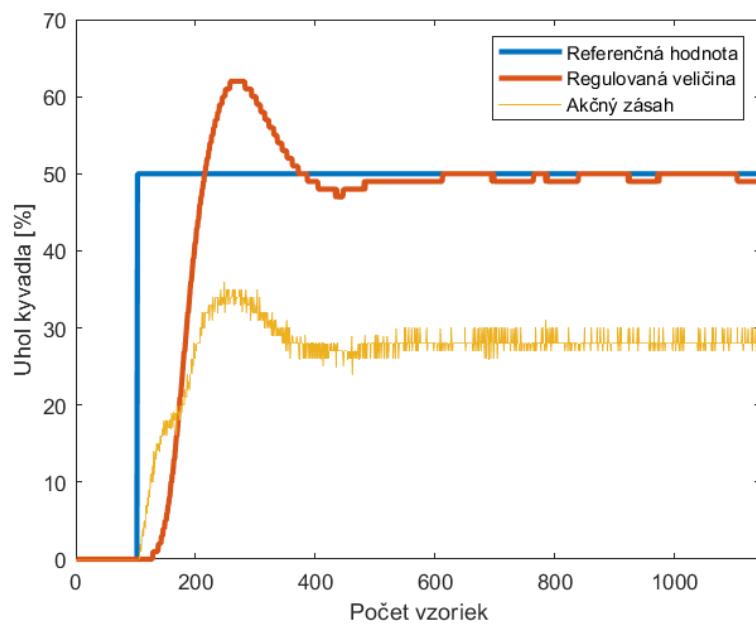
Vzhľadovo pôsobí príklad PID riadenia v API Simulink jednoduchšie, ako tomu bolo pri tom istom riadení v MATLABE alebo Arduino IDE. Pred pripravené bloky z knižnice AeroLibrary stačí v príklade logicky pospájať, a následne zvoliť vhodné parametre v maskách blokov. Regulovanú sústavu v tomto príklade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

blablabla este prid=am kecy

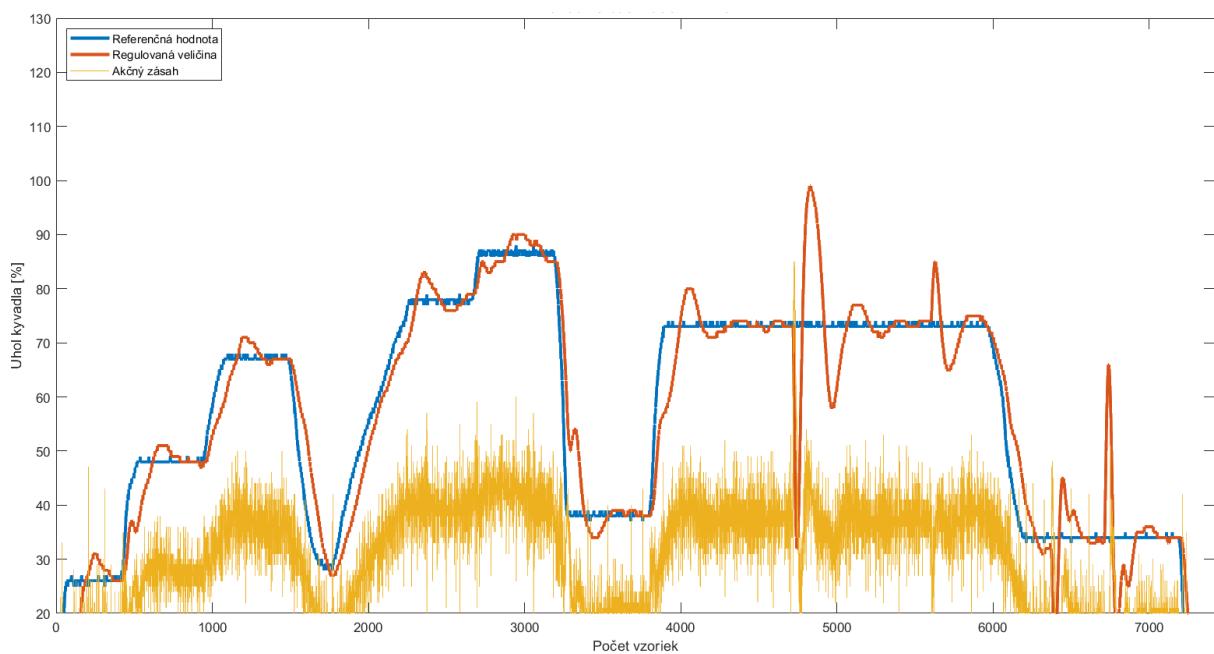


Obr. 3.9: Ukážka riadenia systému pomocou PID regulátora v API Simulink.

### Výstupy



Obr. 3.10: Reakcia systému na skokovú zmenu.



Obr. 3.11: Manuálna trajektória.

## 4 Záver

Táto časť diplomovej práce je povinná. Autor práce uvedie zhodnotenie riešenia, jeho výhody resp. nevýhody, použitie výsledkov, ďalšie možnosti a podobne. Môže aj načrtnúť iný spôsob riešenia úloh, resp. uvedie, prečo postupoval uvedeným spôsobom.

# Literatúra

- [1] Arduino uno r3 development board microcontroller for diy project. Store. Online., -. -, <https://sunhokey.en.made-in-china.com/product/bjkxIyAKQdhF/China-Arduino-Uno-R3-Development-Board-Microcontroller-for-DIY-Project.html>.
- [2] Arduino mega 2560 rev3. Store. Online., -. -, <https://store.arduino.cc/collections/boards/products/arduino-mega-2560-rev3>.
- [3] Petr Horáček. Laboratory experiments for control theory courses: A survey. *Annual Reviews in Control*, 24:151–162, 2000.
- [4] Ed Edwards. All about position sensors. article. Online., -. 2021, <https://www.thomasnet.com/articles/instruments-controls/all-about-position-sensors>.
- [5] Mila Mary Job and P. Subha Hency Jose. Modeling and control of mechatronic aeropendulum. In *2015 International Conference on Innovations in Information, Embedded and Communication Systems (ICIIECS)*, pages 1–5, 2015.
- [6] Eniko T. Enikov and Giampiero Campa. Mechatronic aeropendulum: Demonstration of linear and nonlinear feedback control principles with matlab/simulink real-time windows target. *IEEE Transactions on Education*, 55(4):538–545, Nov 2012.
- [7] Andrej Polák. Spracovanie meraných údajov, modelovanie a identifikácia mechatronického systému aerokyvadlo.
- [8] Jakub Ondera. Návrh konštrukcie, riadiacich prvkov a algoritmov mechatronického systému aerokyvadlo.
- [9] Fanavararan Sharif. Aero pendulum control system (pr22). Store. Online., 2021. 2021, <https://www.zoodel.com/en/product/ZP22344/Aero-Pendulum-Control-System-PR22>.
- [10] Gergelytakacs. 2022, <https://github.com/gergelytakacs>.
- [11] Mgulan. 2022, <https://github.com/mgulan>.
- [12] Gergely Takacs. Automationshield. Wiki. Online., 2021. 13.7.2021, <https://github.com/gergelytakacs/AutomationShield/wiki>.

- [13] Gergely Takacs. Automationshield. Code. Online., 2021. 23.12.2021, <https://github.com/gergelytakacs/AutomationShield>.
- [14] Saroja Dhanapal and Evelyn Wan Zi Shan. A study on the effectiveness of hands-on experiments in learning science among year 4 students. 2013.
- [15] Harry Baggen. The javelin stamp. *Elector Electronics*, 1(1):0, 2003.
- [16] Arduino uno rev3. Info. Online., 2021. 2021, <https://store.arduino.cc/products/arduino-uno-rev3>.
- [17] Texas instruments tps56339 buck converters. store. Online., 2021. 2021, <https://www.mouser.ee/new/texas-instruments/ti-tps56339-buck-converters/>.
- [18] Arduino. Overview of the arduino uno components. article. Online., 2021. 2021, <https://docs.arduino.cc/tutorials/uno-rev3/intro-to-board>.
- [19] 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).
- [20] DANIELLE COLLINS. What are coreless dc motors? article. Online., 2018. 09.10.2021, <https://www.motioncontrolltips.com/what-are-coreless-dc-motors/>.
- [21] 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*, volume -, pages 1036–1039, 2010.
- [22] Pmv45en2. shop. Online., 2021. 2021, <https://www.nexperia.com/products/mosfets/small-signal-mosfets/PMV45EN2.html>.
- [23] 7mm diameter 720 coreless motor for quadcopter. shop. Online., 2021. 2021, <https://www.elecrow.com/7mm-diameter-720-coreless-motor-for-quadcopter.html>.
- [24] SHAWN HYMEL. Ina169 breakout board hookup guide. article. Online., 2021. 2021, <https://learn.sparkfun.com/tutorials/ina169-breakout-board-hookup-guide/all>.
- [25] Ina169na/250. store. Online., 2021. 2021, <https://www.ti.com/store/ti/en/product/?p=INA169NA/250>.
- [26] The Editors of Encyclopaedia Britannica. Hall effect. article. Online., 2021. 2021, <https://www.britannica.com/science/Hall-effect>.
- [27] Halluv 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>.
- [28] As5600. 2022, <https://ams.com/en/as5600>.

- [29] Tps56339. 2022, <https://www.ti.com/product/TPS56339>.
- [30] Prototyping circuit boards: Everything you need to know before you start. article. Online., 2020. 2020, <https://www.pcbnet.com/blog/prototyping-circuit-boards-everything-you-need-to-know-before-you-start/>.
- [31] What are the factors that will affect the pcb lifespan and how to extend it: 2021 newest.
- [32] Writing a library for arduino. article. Online., 2022. 2022, <https://docs.arduino.cc/hacking/software/LibraryTutorial>.
- [33] Arduino - data types. article. Online., -. 2022, [https://www.tutorialspoint.com/arduino/arduino\\_data\\_types.htm](https://www.tutorialspoint.com/arduino/arduino_data_types.htm).
- [34] Michael Dellnitz Martin Golubitsky. Linear mappings and bases. article. Online., -. 2022, <https://ximera.osu.edu/laode/linearAlgebra/linearMapsAndChangesOfCoordinates/linearMappingsAndBases>.
- [35] Nicholas Zambetti. A guide to arduino & the i2c protocol (two wire). article. Online., 2022. 2022, <https://docs.arduino.cc/learn/communication/wire>.
- [36] Bit shifting. article. Online., 2017. 2022, <https://www.techopedia.com/definition/26846/bit-shifting>.

# Zdrojový kód súboru AeroShield.h

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

Zdrojový kód 4.1: Zdrojový kód súboru AeroShield.h.

# Zdrojový kód súboru AeroShield.cpp

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

```

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

```

Zdrojový kód 4.2: Zdrojový kód súboru AeroShield.cpp.

# Zdrojový kód súboru AeroShieldOpenLoop.ino

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

Zdrojový kód 4.3: Zdrojový kód súboru AeroShieldOpenLoop.ino.

# Zdrojový kód súboru AeroShieldPID.ino

```
1 #include "AeroShield.h"
2 #include <Sampling.h>
3
4 #define MANUAL 0
5 #define KP 1.7
6 #define TI 3.8
7 #define TD 0.25
8
9 float startangle=0;
10 float lastangle=0;
11 float pendulumAngle;
12
13 unsigned long Ts = 3;
14 unsigned long k = 0;
15 bool nextStep = false;
16 bool realTimeViolation = false;
17
18 int i=i;
19 int T=1000;
20 float R
21 []={45.0,23.0,75.0,32.0,58.0,10.0,35.0,19.0,9.0,43.0,23.0,65.0,15.0,80.0};
22 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(AeroShield.detectMagnet());
29     startangle = AeroShield.calibration(AeroShield.getRawAngle());
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]))) {
           analogWrite(5,0);
```

```

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

Zdrojový kód 4.4: Zdrojový kód súboru AeroShieldPID.ino.

# Zdrojový kód súboru AeroShield.m

```
1 classdef AeroShield < handle
2
3 properties
4     arduino;
5     as5600;
6 end
7 properties (Constant)
8     AERO_UPIN = 'D5';
9     AERO_RPIN = 'A3';
10    VOLTAGE_SENSOR_PIN = 'A2';
11    voltageReference = 5.0;
12    ShuntRes = 0.1;
13    correction1 = 4.1220;
14    correction2 = 0.33;
15    repeatTimes = 50;
16 end
17
18 methods
19     function begin(AeroShieldObject)
20         AeroShieldObject.arduino = arduino();
21         AeroShieldObject.as5600 = device(AeroShieldObject.arduino, '
22             I2CAddress', 0x36);
23         configurePin(AeroShieldObject.arduino, AeroShieldObject.
24             AERO_UPIN, 'DigitalOutput')
25         disp('AeroShield initialized.')
26     end
27     function startangle = calibration(AeroShieldObject)
28         write(AeroShieldObject.as5600, 0x0c, 'uint8');
29         write(AeroShieldObject.as5600, 0x0d, '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, 0x0d, '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                 ));
52         end
53         currentMean= Current/repeatTimes;
54         currentMean= currentMean-correction2;
55         if currentMean < 0.000
56             currentMean= 0.000;
57         end
58         current= 0;
59         voltageValue=0;
60     end
61 end
62 end
```

Zdrojový kód 4.5: Zdrojový kód súboru AeroShield.m.

# Zdrojový kód súboru AeroShieldOpenLoop.m

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

Zdrojový kód 4.6: Zdrojový kód súboru AeroShieldOpenLoop.m.

# Zdrojový kód súboru AeroShieldPID.m

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

Zdrojový kód 4.7: Zdrojový kód súboru AeroShieldPID.m.