

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
STROJNÍCKA FAKULTA
Ústav automatizácie, merania a aplikovanej informatiky

**AEROSHIELD: MINIATÚRNY EXPERIMENTÁLNY MODUL
AEROKYVADLA**

Práca Študentskej vedeckej konferencie SjF v sekcií:
Automatizácia a informatizácia strojov a procesov

Autor práce ŠVK: Peter Tibenský (3.r)

Študijný odbor: kybernetika

Študijný program: automatizácia a informatizácia strojov a procesov

Školiteľ práce ŠVK: Ing. Mgr. Anna Vargová

Bratislava, 6. apríla 2022

Obsah

Zoznam obrázkov	i
Zoznam zdrojových kódov	ii
Zoznam tabuliek	iii
Úvod	1
1 Motivácia	2
2 AeroShield	5
2.1 Hardware	7
2.1.1 Popis súčiastok	7
2.1.2 Schéma zapojenia	10
2.1.3 Doska plošných spojov	11
2.1.4 Model držiaku kyvadla	13
2.1.5 Cenová kalkulácia AeroShieldu	13
2.2 Software	15
2.2.1 Header	15
2.2.2 Source	17
2.2.3 MATLAB	25
3 Didaktické príklady	27
3.1 Programy v otvorenej slučke, bez späťnej väzby	27
3.1.1 <i>AeroShieldOpenLoop.ino</i>	27
3.1.2 <i>AeroShieldOpenLoop.m</i>	29
3.2 Programy v uzatvorennej slučke, so spätnou väzbou	31
4 Záver	32
Literatúra	33
Arduino IDE	iv
Zdrojový kód AeroShield.h	iv
Zdrojový kód AeroShield.cpp	v
Zdrojový kód AeroShieldOpenLoop.ino	vii

MATLAB	viii
Zdrojový kód AeroShield.m	viii
Zdrojový kód AeroShieldOpenLoop.m	ix

Zoznam obrázkov

1.1	Model kyvadla, tvorený v programe CATIA.	2
1.2	Aeropendulum značky Real Sim[1].	3
1.3	Arduino UNO.[2]	4
2.1	(a) Prvá verzia AeroShieldu. (b) Schéma zapojenia prvej verzie AeroShieldu.	5
2.2	meranie uhla kyvadla	6
2.3	buck converter	7
2.4	Zapojenie akčného člena a typ motorčeka	8
2.5	meranie prúdu	9
2.6	meranie uhla kyvadla	9
2.7	Schéma zapojenia AeroShieldu	10
2.8	Vedľajšia doska AeroShieldu- breakout board	11
2.9	(a) Vrchná strana AeroShieldu (b) Spodná strana AeroShieldu	12
2.10	Dosky plošných spojov AeroShieldu	13
3.1	Výstup z programu AeroShieldOpenLoop.ino.	28
3.2	Výstup z programu AeroShieldOpenLoop.m.	31

Zoznam zdrojových kódov

2.1	Ukážka zdrojového kódu headeru.	16
2.2	Triedy a objekty.	16
2.3	Source volanie funkcie.	17
2.4	Zdrojový kód funkcie mapFloat.	18
2.5	Zdrojový kód funkcie serialPrint.	18
2.6	Zdrojový kód funkcie readOneByte.	19
2.7	Zdrojový kód funkcie readTwoBytes.	19
2.8	Zdrojový kód funkcie detectMagnet.	20
2.9	Zdrojový kód funkcie getMagnetStrength.	20
2.10	Zdrojový kód funkcie getRawAngle.	21
2.11	Zdrojový kód funkcie begin.	22
2.12	Zdrojový kód funkcie calibration.	22
2.13	Zdrojový kód funkcie convertRawAngleToDegrees.	23
2.14	Zdrojový kód funkcie referenceRead.	23
2.15	Zdrojový kód funkcie actuatorWrite.	24
2.16	Zdrojový kód funkcie currentMeasure.	24
2.17	Knižnica AeroShield.m properties.	25
2.18	Knižnica AeroShield.m properties.	26
3.1	AeroShield open loop dekleracia.	27
3.2	AeroShield open loop setup().	27
3.3	AeroShield open loop loop().	28
3.4	AeroShield open loop inicializacia.	29
3.5	AeroShield open loop grafy.	29
3.6	AeroShield open loop, while cyklus.	30
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 AeroShield.m.	viii
4.5	Zdrojový kód súboru AeroShieldOpenLoop.m.	ix

Zoznam tabuliek

2.1	Cenová kalkulácia AeroShieldu.	14
2.2	Dátové typy	17

Úvod

Cieľom tejto práce je návrh, výroba a naprogramovanie modernej učebnej pomôcky AeroShieldu (ďalej len „shield”), ktorý slúži na výuku základov teórie riadenia a elektrotechniky.

Učebné pomôcky sú nevyhnutnou, no často zanedbávanou súčasťou výuky. Študenti si vďaka nim môžu lepšie predstaviť a pochopiť problematiku daného učiva, keďže môže 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 nepraktické.

Za cieľom sprístupnenia experimentálnych modulov širokej verejnosti bol založený projekt AutomationShield, ktorý ponúka pomerne jednoduché a cenovo dostupné experimentálne moduly ako open-source¹ študentské projekty.

Vhodnou platformou na implementáciu týchto modulov sú napríklad prototypizačné dosky Arduino ktoré sú taktiež open-source. Ich nízka cena a celosvetová popularita, spojená s obrovským množstvom návodov, informácií a pomocov, vytvára ideálnu platformu pre začínajúcich, ako aj pokročilých programátorov, elektrotechnikov alebo hobby nadšencov.

V tejto práci, je opísaný postup výroby a fungovania shieldu s dôrazom na zrozumiteľnosť jednotlivých aspektov aj čitateľom, ktorý o danej téme nie sú dokonale oboznámený. Na začiatku bakalárskej práce, v časti hardware, je opísaný základný princíp fungovania shieldu a následne jeho jednotlivé súčiastky. Pochopenie fungovania jednotlivých súčiastok shieldu je kritické pre správnu manipuláciu užívateľa s jeho jednotlivými časťami. Poslednú časť tvorí tvorba dosky plošných spojov pre shield v programe DipTrace.

V softvérovej časti sú bližšie predstavené jednotlivé charakteristické funkcie shieldu. Funkcie sú usporiadane do logických celkov, pre ľahšiu prácu s kódom.

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

1 Motivácia

Našim zámerom, bolo vytvorenie funkčnej školskej pomôcky, na štýl experimentu celosvetovo známeho pod názvom Aeropendulum, čo v doslovnom preklade znamená vzdušné kyvadlo. Jedná sa o pomerne jednoduché zariadenie pozostávajúce z niekoľkých častí. Akčným členom tohto zariadenia je motorček na jednosmerný prúd, 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 tyčky. Zariadenie na meranie pootočenia môže byť potenciometer, senzor hallovho javu(efektu), alebo iné [4]. V našom prípade budeme používať senzor hall efektu ktorého fungovanie je opísané v časti 2.1.1. Zariadenie na meranie uhlu je následne upevnené na podstavec, aby sa motor mohol voľne pohybovať. Podobu modelu Aeropendulum, môžete vidieť na obr. 1.1.



Obr. 1.1: Model kyvadla, tvorený v programe CATIA.

Open-source projekt AutomationShield vyvíjaný na ústave Automatizácie, merania a aplikovanej informatiky SJF STU, je zameraný na vývoj hardwarových a softwarových nástrojov určených na vzdelávanie a doplnenie vzdelávacieho procesu. Jadrom celého projektu je tvorba rozširujúcich dosiek (shieldov) vyvíjaných pre populárny typ prototypizačných dosiek s mikrokontrolérmi Arduino, ktoré majú za cieľ lepšiu výučbu strojného inžinierstva, mechatroniky a riadenia[5].

Zdrojový kód k AeroShieldu, ako aj ku všetkým modulom AutomationShield, nájdeme na platforme GitHub[6], ktorá slúži ako obrovská knižnica kódov, návodov a postupov pre kohokoľvek. Na samostatnej stránke AutomationShield nájdeme zoznam jednotlivých shieldov a to, v akom procese výroby sa nachádzajú. Ku každému shieldu nájdeme jeho podrobnejšiu dokumentáciu, knižnice, zdrojové kódy, ako aj pred programované ukážky fungovania. Tým že GitHub je open-source platforma, dokumenty na stránke môže ktokoľvek upravovať alebo vylepšovať, čo tvorí ideálny priestor pre rozvoj myšlienok a tvorivý proces. Na dokumentoch môže naraz pracovať niekoľko desiatok ľudí, čím sa častokrát mnohonásobne urýchľuje proces tvorby a hľadania chýb.

Ako už bolo spomenuté, hlavnou motiváciou tohto projektu je nízka dostupnosť a vysoká cena podobných učebných pomôcok. Výučba je preto častokrát až príliš zameraná na memorovanie faktov a teórie, namiesto praktických experimentov a skúseností typu pokus-omyl. Jediný podobný dostupný produkt na kúpu nie ako kit, je Aeropendulum od perzskej značky Real Sim ktoré je na obr.1.2. Študenti si omnoho rýchlejšie osvoja metódy programovania a automatizácie, pokiaľ majú možnosť experimenty sami tvoriť a skúmať vplyv reálnych výstupov na zvolené vstupy. S úmyslom priniesť širokej verejnosti lacnejšiu a výkonnejšiu alternatívu, vtedajším mnohonásobne drahším a menej výkonným prototypizačným doskám[7], prišla na trh v roku 2005 prototypizačná doska Arduino. Projekt vznikol v Taliansku ako kolaborácia medzi viacerými nadšencami elektrotechniky a programovania, na ktorých čele bol Massimo Banzi.



Obr. 1.2: Aeropendulum značky Real Sim[1].

Veľkou výhodou dosiek Arduino a ich nadstavbových shieldov je fakt, že sú pomerne lacné a majú malé rozmery (Arduino UNO: 68.6*53.4mm[8]). Tieto fakty umožňujú študentom pracovať na experimentoch nielen na pôde školy, ale experimenty si môžu zobrať domov a pracovať na nich aj mimo vyučovacieho procesu. Na správne fungovanie a programovanie dosky nám postačuje len USB kábel a samotná doska. Vzhľadom na nízky počet potrebných súčiastok a fakt, že mikročip arduina je v prípade poruchy jednoducho vymeniteľný², je ich používanie na školách príjemné a jednoduché. Pre naše účely je vhodná doska Arduino UNO, ktorú môžeme vidieť na obr.1.3. Na doske sa nachádza 14 digitálnych a 6 analógových pinov. Niektoré piny sú označené špeciálnym symbolom ~. Tieto piny sú schopné produkovať PWM³ signál, ktorý potrebujeme na správne ovládanie motoru kyvadla.



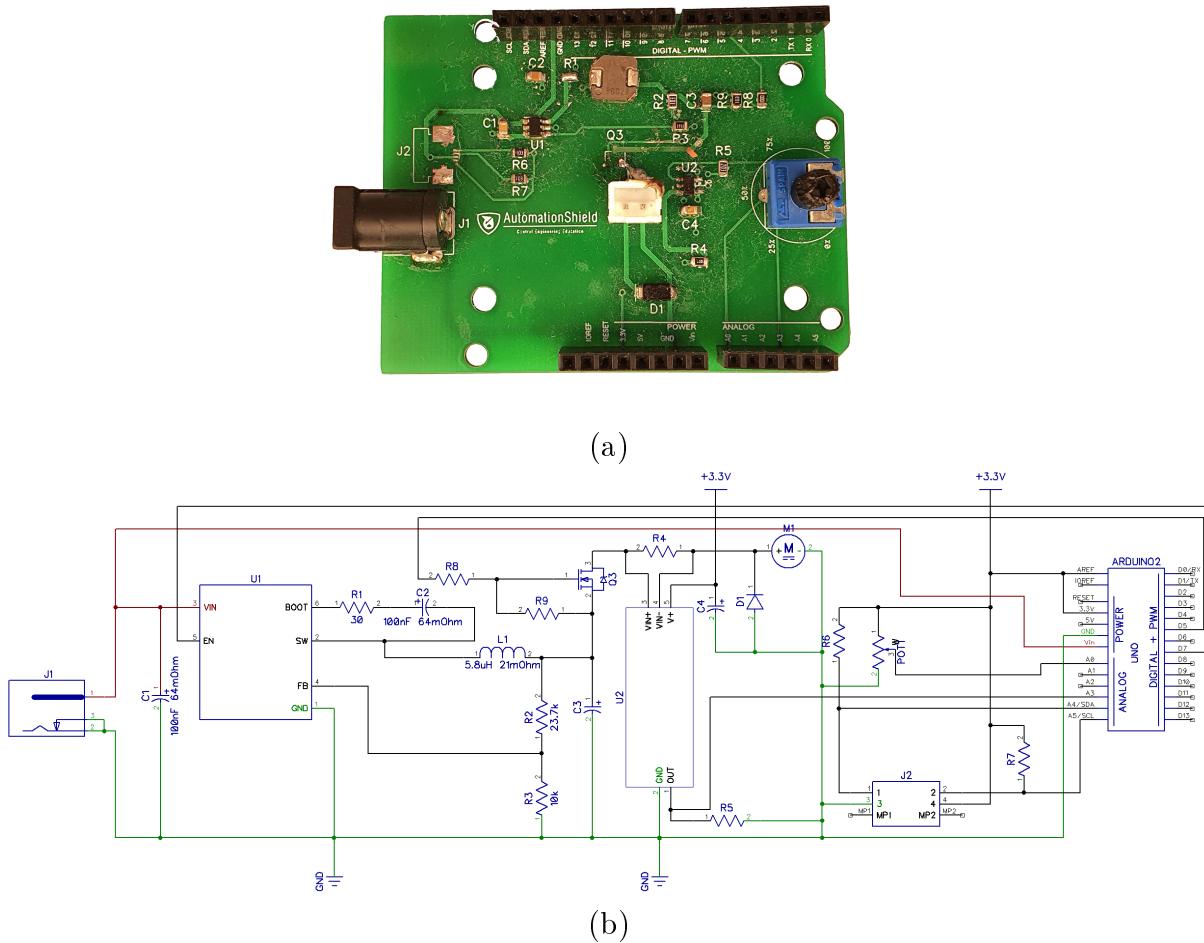
Obr. 1.3: Arduino UNO.[2]

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

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

2 AeroShield

Téma tejto bakalárskej práce vznikla ako pokračovanie, na už započatom projekte. Prvá verzia dosky a samotného kyvadla, vznikla ako záverečný projekt na predmet mikroprocesorová technika. Schému zapojenia hlavnej dosky, ako aj fotografiu zostavenej verzie, môžeme vidieť na obr.2.1.

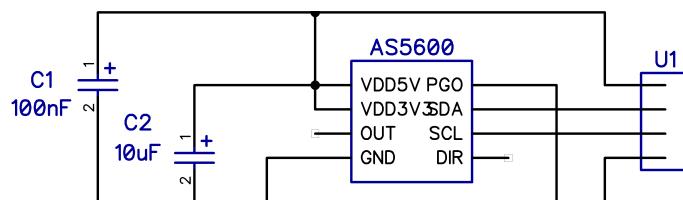


Obr. 2.1: (a) Prvá verzia AeroShieldu. (b) Schéma zapojenia prvej verzie AeroShieldu.

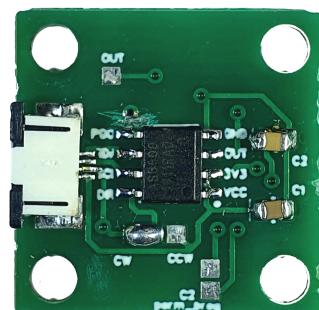
Prvá verzia dosky mala niekoľko nedostatkov, vďaka ktorým bola prakticky nepoužiteľná. Hlavnými nedostatkami boli:

- 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 školského projektu bola vytvorená hlavná doska, na ktorej sa nachádza väčšina elektroniky a menšia, vedľajšia doska, ktorá slúži na fungovanie senzoru hall efektu. Táto doska fungovala bezproblémovo a teda nebolo potrebné nijakým spôsobom meniť jej schému zapojenia, viditeľnú na obr.2.2.a. Tejto menšej doske sa budeme bližšie venovať v časti 2.1.3, no jej podoba je viditeľná na obr.2.2.b.



(a) Schéma zapojenia externej dosky.



(b) Doska slúžiaca na fungovanie senzoru hall efektu.

Obr. 2.2: meranie uhla kyvadla

2.1 Hardware

2.1.1 Popis súčiastok

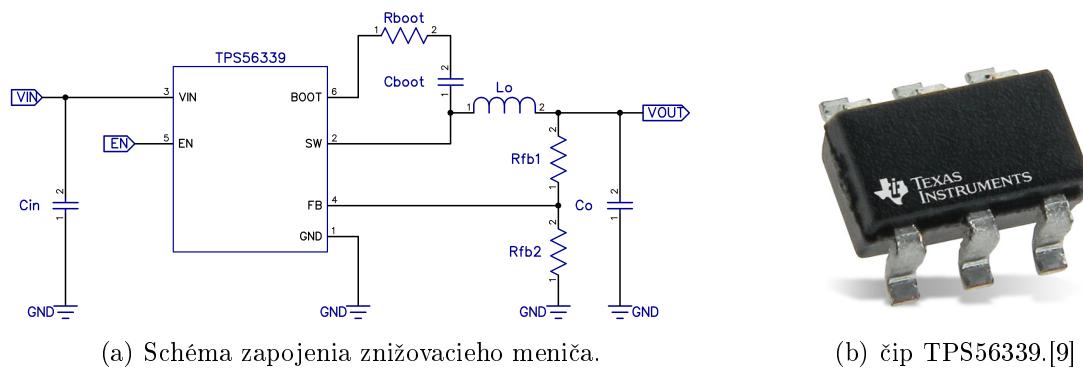
V tejto časti sa bližšie pozrieme na jednotlivé nevyhnutné súčasti zapojenia AeroS-shieldu. Konkrétnie sa jedná o tieto prvky:

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

Znižovací menič

Na správne napájanie akčného člena, motorčeka, potrebujeme napäťie v rozmedzí 0-3,7V. Na shield je však privádzané, pomocou koaxiálneho napájacieho konektora, napäťie 12V, ktoré by motor v priebehu chvíle zničilo. Na zniženie napäťia preto použijeme znižovací menič tzv. buck converter.

Hlavnou časťou konvertora je čip TPS56339 od výrobcu Texas Instruments obr.2.3.b. Znižovanie napäťia funguje za pomoci dvoch integrovaných N-kanálových 70-mΩ a 35-mΩ high-side mosfetov⁴, 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[9].



(a) Schéma zapojenia znižovacieho meniča.

(b) čip TPS56339.[9]

Obr. 2.3: buck converter

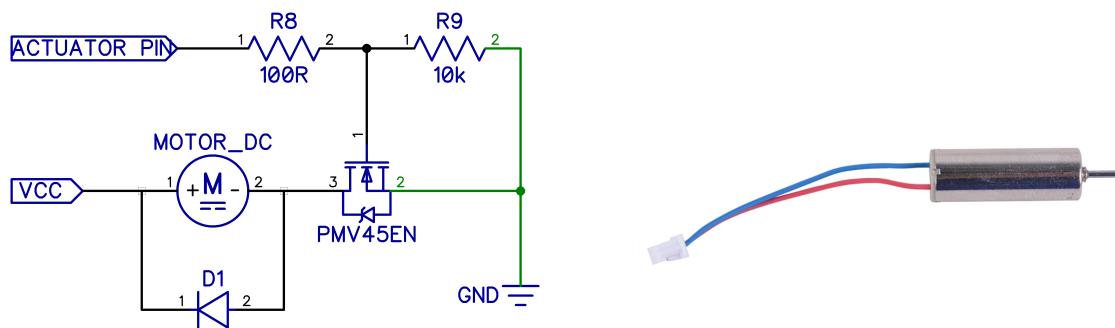
Na čip je privádzané napäťie 12V ktoré sa pomocou zapojenia viditeľného na schéme obr.2.3.a, znižuje na napäťie 3,7V. Napájanie motora musí byť realizované externe pomocou koaxiálneho napájacieho konektora, z dôvodu vysokého prúdu odoberaného motorom počas silného zaťaženia. Rovnaký konektor sa sice nachádza aj na doske Arduino UNO a pomocou VIN pinu sa z neho dajú napájať napäťím 12V aj iné zariadenia, avšak tento pin je napojený na diódu, obmedzujúcu prúd na 1A[10][11].

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

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[12]. Takéto jadro ale samé o sebe nie je veľmi pevné a nedrží dobre tvar, preto sa častokrát zalieva epoxidom. 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 znižuje hmotnosť a tým aj zotrvačnosť rotora, čo je dôležité pre naše použitie, kedy potrebujeme dosahovať vysokú akceleráciu a rýchle spomalenie rotora. Ďalšou výhodou je fakt, že nedochádza k stratám na železe a tým pádom sa účinnosť takýchto motorov blíži až ku 90%[13]. Motor, resp. otáčky motora sú riadené pomocou impulzovej šírkovej modulácie(PWM) a tieto impulzy do motoru prechádzajú cez N-kanálový mosfet PMV45EN2 od výrobcu Nexperia[14].



(a) Schéma zapojenia motorčeka.

(b) Akčný člen sústavy.[15]

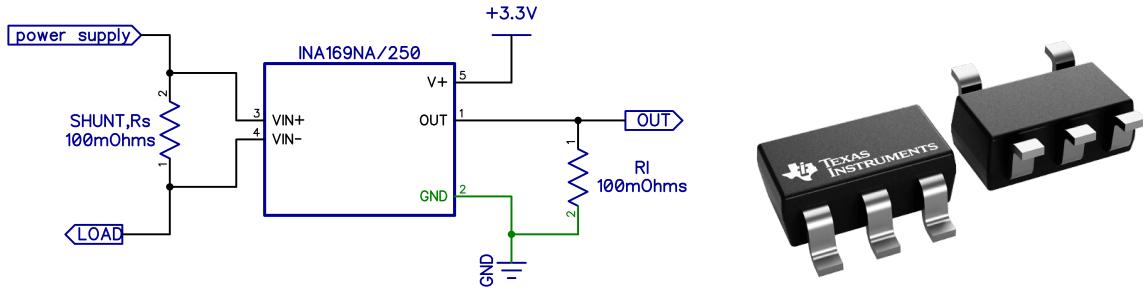
Obr. 2.4: Zapojenie akčného člena a typ motorčeka

meranie prúdu

Z dôvodu merania prúdu odoberaného motorom, bol do schémy pridaný monitor prúdu, takzvaný "current shunt monitor". V AeroShielde je použitý snímač INA169NA/250 od výrobcu Texas Instruments obr.2.5.b.

INA169 funguje na základe zaznamenávania zmien napäťia na stranách shunt rezistora obr.2.5.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 $I_s = \frac{V_{OUT} \times 1k\Omega}{R_s \times R_l}$ kde V_{OUT} je napätie namerané na výstupe, $1k\Omega$ je konštanta vnútorných odporov senzoru, R_s je hodnota shunt rezistora v Ω a R_l je hodnota rezistora na výstupe, taktiež v Ω [16].



(a) Schéma zapojenia snímača prúdu.

(b) Senzor INA169NA/250[17]

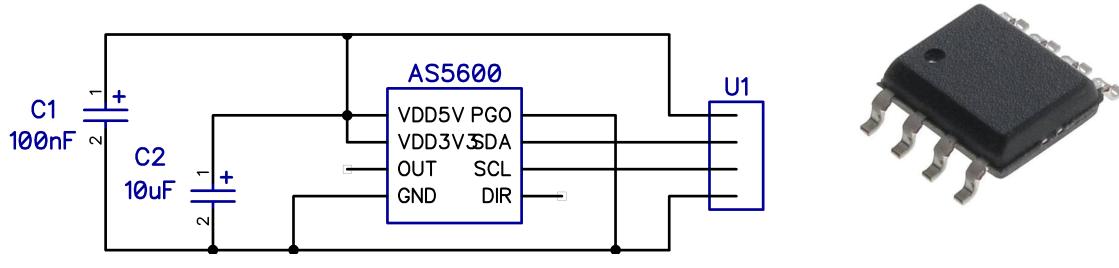
Obr. 2.5: meranie prúdu

meranie uhla kyvadla

Na správne fungovanie AeroShieldu je dôležité vedieť s vysokou presnosťou merať uhol naklonenia kyvadla. Na tento účel sme si zvolili meranie uhlu bez kontaktnou formou, pomocou snímača hall efektu. Hall efekt vieme opísť ako vznik priečného elektrického poľa v pevnom materiáli, keď ním preteká elektrický prúd a tento materiál je umiestnený v magnetickom poli, ktoré je kolmé na prúd[18]. Toto elektrické pole resp. vznik elektrického potenciálu vieme detegovať a na základe jeho zmeny vieme určiť rotáciu kyvadla. 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.2.6.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, ktorá sa ďalej používa na automatické riadenie zosilnenia(AGC), ktoré slúži na kompenzáciu teploty priestoru a magnetu a veľkosti 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.2.6.a.



(a) Schéma zapojenia čipu na meranie uhlu.

(b) čip AS5600[19]

Obr. 2.6: meranie uhla kyvadla

2.1.2 Schéma zapojenia

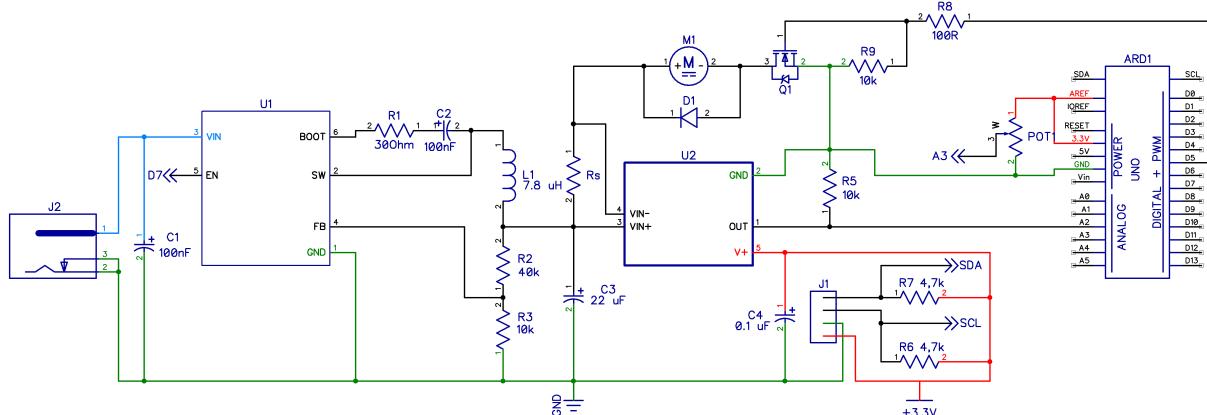
Všetky schémy zapojenia boli tvorené v bezplatnej verzii programu DipTrace. DipTrace slúži ako prostredie na tvorbu elektrotechnických schém, potrebných pre výrobu dosiek plošných spojov, ako aj pre účely prehľadnosti zapojenia komponentov na týchto doskách. Program v sebe zahŕňa časť pre tvorbu samotných komponentov, pokiaľ sa tieto už nenachádzajú v niektornej z knižníc programu, časť kde sa tvoria schémy zapojenia a časť na tvorbu dosiek plošných spojov.

Nie všetky komponenty potrebné na tvorbu AeroShieldu boli zahrnuté v knižniciach DipTracu, avšak tieto komponenty sa nachádzali na stiahnutie na stránkach výrobcov odkiaľ boli importované do novej knižnice, slúžiacej na účely tvorby schémy AeroShieldu. 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 štvorčekovú 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.

Veľmi dobrú vlastnosťou programu DipTrace je možnosť zafarbovania 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.2.7. Na schéme zapojenia môžeme vidieť všetky komponenty, potrebné na správne fungovanie AeroShieldu. Názvy komponentov sú uvádzané základnými značkami

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



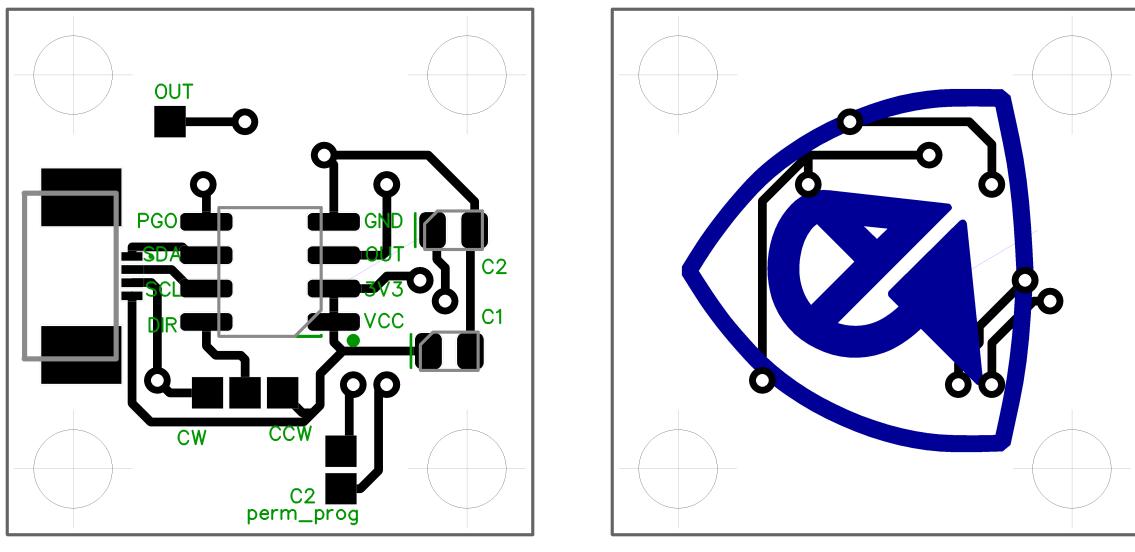
Obr. 2.7: Schéma zapojenia AeroShieldu

2.1.3 Doska plošných spojov

Po návrhu a kontrole schému 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 pinov na spájkovanie. Dosky plošných spojov majú niekoľko nevýhod, ale aj výhod oproti ponúkaným alternatívam[20].

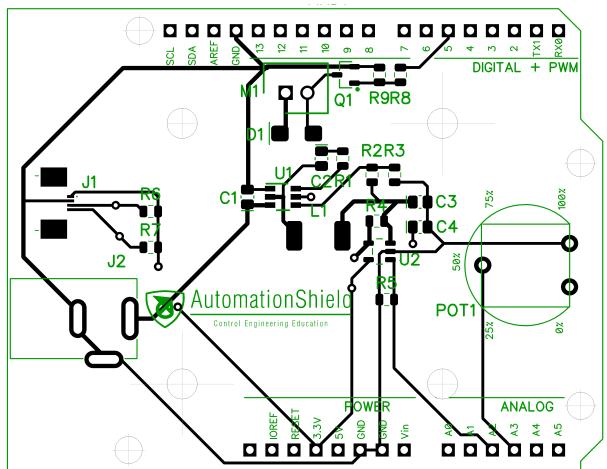
Výhodou je fakt, že vodivé spojenia medzi jednotlivými súčiastkami zapojenia, sú realizované vrstvou medzi ktorá je ukrytá pod ochrannými vrstvami povrchu dosky, na rozdiel od typických kálových spojení. Káble majú niekoľko nedostatkov ako to že sa vedia ľahko vypojiť, ľahko dochádza k ich porušeniu a v neposlednom rade, nepôsobia veľmi esteticky. V prípade nesprávneho prepojenia pinov má jednoznačnú výhodu spoj realizovaný káblami, keďže pri doskách plošných spojov sa s už hotovými cestami manipuluje obtiažne. Ďalšou výhodou dosiek plošných spojov je skutočnosť, že sú veľmi odolné a kompaktné. Tým že vodivé cesty môžu mať veľmi malé rozmer, ovplyvňujúcim faktorom veľkosti dosky plošných spojov je samotná veľkosť jej komponentov.

Po prenesení schém do DipTrace PCB, sú jednotlivé komponenty rozhádzané a nemajú žiadne logické rozloženie. Program ponúka možnosť automatického zoradenia komponentov na vyhradenej ploche, avšak táto funkcia komponenty uložila nie podľa našich potrieb a teda, využili sme možnosť manuálneho umiestnenia jednotlivých komponentov. Pri pohybovaní jednotlivými komponentami môžeme vidieť čiary, ktoré symbolizujú prepojenia s ostatnými komponentami a vďaka tomu vieme komponenty logicky pouklať.

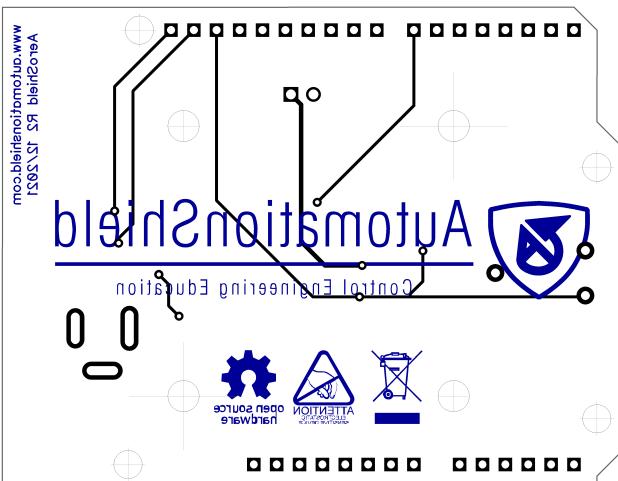


Obr. 2.8: Vedľajšia doska AeroShieldu- breakout board

Po zvolení optimálneho rozmiestnenia komponentov treba jednotlivé piny poprepájať vodivými cestami, ktoré nám nahradzajú funkciu kálov. Máme možnosť zvoliť automatické rozmiestnenie ciest alebo ich manuálnu tvorbu. V našom prípade sme zvolili manuálnu tvorbu ciest, pretože ich vieme čo najlepšie optimalizovať. Ako je viditeľné aj na obr.2.9.a, nie všetky cesty majú rovnakú šírkmu. Je to z toho titulu že niektorými cestami prúdi vyšší prúd a to až do 1A. V zásade sa používa pravidlo, čím vyšší prúd preteká



(a)



(b)

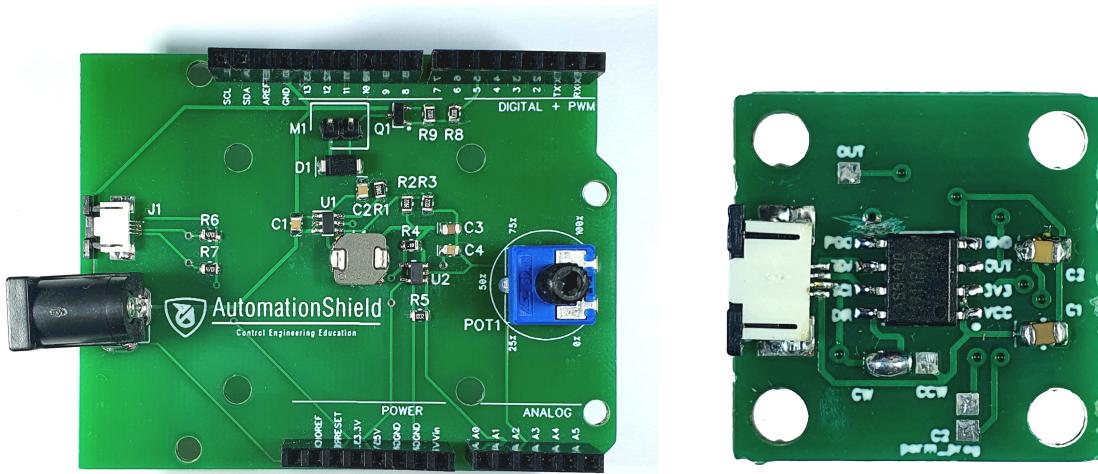
Obr. 2.9: (a) Vrchná strana AeroShieldu (b) Spodná strana AeroShieldu

vodičom, tým väčšiu plochu prierezu by mal mať. Prúdy pretekajúce vodičmi tieto vodiče zahrievajú a pokial je toto zahrievanie nadmerné, môže dôjsť k poškodeniu vodiča.

Tvorba ciest má niekoľko pravidiel, avšak najdôležitejšie z nich je že jednotlivé cesty ktoré v schéme zapojenia nie sú prepojené, sa nemôžu križovať inak dôjde k ich vzájomnému vyskratovaniu. 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, takzvané via, spájajúce obe strany dosky.

Pri výrobe dosky sa taktiež myšlelo na montáž držiaku kyvadla, pre ktoré boli vytvorené 4 diery na jeho následné prichytenie pomocou skrutiek. Finálna verzia hlavnej dosky je na obr.2.9. Zhotovená bola aj menšia doska tzv. breakout board, slúžiaca na meranie uhlu kyvadla, ktorá je na obr.2.8.

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



(a) Hlavná doska AeroShieldu

(b) Vedľajšia doska AeroShieldu

Obr. 2.10: Dosky plošných spojov AeroShieldu

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 via a iné. Gerber súbor ďalej posielame výrobcovi PCB dosiek kde si môžeme zvoliť ďalšie parametre dosky, ako jej farbu, možnosti 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.2.10.a a dosky breakout boardu na obr.2.10.b.

2.1.4 Model držiaku kyvadla

Tu ešte poviem čo to a designovaní držiaku pendulum

2.1.5 Cenová kalkulácia AeroShieldu

Hlavnou podmienku pri tvorbe AeroShieldu, bola jeho funkcionality, no zároveň nízka cena. Za účelom predstavy cenovej relácie jedného kusu AeroShieldu, bola zostavená tabuľka 2.1 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.

Názov	Popis	Ks.	Cena	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
				Spolu 31,95

Tabuľka 2.1: Cenová kalkulácia AeroShieldu.

2.2 Software

Programovacie rozhranie pre platformy arduino sa nazýva Arduino IDE⁵ a využíva programovací jazyk C++ resp. jeho podobu, s pridanými špecializovanými príkazmi a funkciami priamo pre arduino IDE. Príkazy sú na prvý pohľad zrozumiteľnejšie ako ich skomplilovaná⁶ podoba v jazyku C++, no funkcie resp. schopnosti príkazu sú rovnaké. Preto je arduino vhodným prostriedkom na programovanie ako pre začiatočníkov, tak aj pre skúsenejších programátorov.

Pri tvorbe programovej časti AeroShieldu je dôležité uvedomiť si fakt že doska vzniká v rámci projektu AutomationShield. Tým že je tento projekt opensource, ktokoľvek môže kód upravovať a vylepšovať, je preto dôležité aby funkcie navádzali používateľov na ich správne použitie a aby boli čo najviac prehľadné. Z tohoto dôvodu bola vytvorená knižnica AutomationShield ktorá v sebe zahŕňa najviac používané funkcie. Predstavme si situáciu kedy v programe ktorý píšeme potrebujeme premenu jednotiek z metrov na centimetre. Pokiaľ takúto funkciu použijeme v kóde jeden krát, môžeme túto funkciu napísať priamo do kódu. Avšak pokiaľ túto funkciu využívame častejšie, dáva zmysel uložiť ju mimo kód a následne túto funkciu zavolať naspäť v prípade jej potreby. Sprehľadňuje sa tak vzniknutý kód a znižuje sa možnosť chýb vďaka monotónnym kopírovaniam tej istej funkcie.

Takúto možnosť externých preddefinovaných funkcií prístupných na zavolanie ponúka objektovo orientované programovanie(OOP) v jazyku C++[21]. Zvyčajne sa vytvárajú dva súbory resp. knižnice, z ktorých jedna sa nazýva **headers** alebo hlavička s koncovkou .h a druhá, **source** alebo zdrojový dokument s koncovkou .cpp. Header slúži ako akýsi návádač a sklad pre premenné a funkcie, ktorý následne komunikuje so source dokumentom v ktorom sú uložené samotné funkcie.

2.2.1 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 kompliacii kódu. Z toho dôvodu používame funkciu **Include guard** ktorá zamedzuje niekoľkonásobne načítanie rovnakých súborov.

Hneď za definovaním knižnice AeroShield.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 <názovKnižnice.h> a na knižnice tvorené programátormi sa používajú úvodzovky "nazovDalsejKniznice.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.

⁵Arduino Integrated Development Environment.

⁶Kompilácia je preklad zdrojového kódu do podoby ktorú vie procesor prečítať a spracovať.

```

#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 2.1: Ukážka zdrojového kódu headeru.

V časti **Zdrojovy kod**, vytvárame **class** alebo triedu ktorá v sebe zahŕňa funkcie a premenné ktoré sa nazývajú **objects**, teda objekty. Class obsahuje podmnožinu objectov ktoré vieme prepájať a spájať vo väčšie celky, vďaka čomu vieme dosiahnuť veľmi komplexné funkcie. Tieto funkcie a premenné môžu byť buď **public**, teda verejné a prístupné aj mimo súbor, alebo **privat**, teda súkromné ktoré su prístupné len v knižniciach header a source. V header súbore sa môže nachádzať jedna, alebo viacero tried, záleží to od logicky deliteľných úsekov kódu, alebo od preferencie programátora. 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 2.2: 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.

2.2.2 Source

Ako sme už spomínali, v knižnici source sa nachádzajú všetky funkcie využívané v AeroShielde. Keďže knižnice sa už definovali a načítavali v súbore header, stačí nám načítať len tento jeden súbor a to pomocou príkazu `#include "AeroShield.h"`, ktorý vložíme na začiatok súboru. Ďalej v súbore deklarujeme jednotlivé samostatné funkcie. Funkcie zapisujeme pomocou už spomínaného classu, dátového typu a názvu funkcie v podobe:

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

Zdrojový kód 2.3: 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[22] (všetky hodnoty sú platné pre arduino UNO, pre iné typy arduina sa hodnoty môžu lísiť):

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

Tabuľka 2.2: 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[23].

```
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 2.4: 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 seriový monitor
    #endif                         // Koniec
}
```

Zdrojový kód 2.5: Zdrojový kód funkcie `serialPrint`.

Popis použitých funkcií z knižnice AeroShield

Kedže na AeroShielde využívame senzor hall efektu, musíme s ním v prvom rade nadviazať komunikáciu pomocou sériovej komunikácie I²C. Protokol I²C 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 posiela 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. Keď sa táto informácia celá odošle, slave vykoná požiadavku a ak je to vyžadované, môže späťne mastrovi poslať údaje. Všetky tieto bity informácií sa posielajú na linke SDA[24].

I²C funguje na princípe master-slave, kedy master je nadriadený a slave je podriadené zariadenie, s ktorým master komunikuje. Master môže naraz komunikovať s viacerými zariadeniami a to na základe jedinečných 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 informácií zo senzoru. Túto funkciu využívame napríklad na čítanie polohy kyvadla.

```
int AeroShield :: readOneByte(int in_adr)
{
    int retVal = -1;           // Zadefinovanie pomocnej premennej
    Wire.beginTransmission(_ams5600_Address); // Zaciatoč komunikacie
    Wire.write(in_adr);        // 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 2.6: 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 bitov⁷.

```
word AeroShield :: readTwoBytes(int in_adr_hi, int in_adr_lo)
{
    word retVal = -1;           // Zadefinovanie pomocnej premennej
    /* citanie "Low" bajtu */
    Wire.beginTransmission(_ams5600_Address); // Zaciatoč komunikacie
    Wire.write(in_adr);        // Poziadavka na zaznamenanie uhlu 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();      // Uloženie prveho bajtu
    /* citanie "High" bajtu */
    Wire.beginTransmission(_ams5600_Address); // Zaciatoč komunikacie
    Wire.write(in_adr);        // Poziadavka na zaznamenanie uhlu kyvadla
    Wire.endTransmission();   // Koniec komunikacie zo strany mastra
```

⁷Bitový posun je operácia vykonávaná so všetkými bitmi binárnej hodnoty, pri ktorej sa posúvajú o určený počet miest doľava alebo doprava[25].

```

Wire.requestFrom(_ams5600_Address, 1); // Ziadost na odpoved
while (Wire.available() == 0); // Cakaj pokial odpoved nepride

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

return retVal; // Zaslanie odpovede
}

```

Zdrojový kód 2.7: 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 2.8: 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 vzdialenosťi. 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 2.9: 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 2.1.1.

```

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

```

Zdrojový kód 2.10: 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 I²C. Príkaz na započatie komunikácie I²C sa pri rôznych typoch dosiek, resp. architektúr mikroradiča Arduino líši. Použijeme preto podmienku `#ifdef` za ktorou nasleduje typ architektúry daného mikroradiča a príslušný príkaz pre začiatok sériovej komunikácie I²C. V prípade Arduino UNO, je to príkaz `Wire.begin()`.

Zároveň je vo funkcií `begin()`, pomocou `if` podmienky, kontrolovaná premenná `isDetected`. Pokiaľ bol magnet detegovaný, vypíše na sériový port správu "Magnet detected" a while⁸ cyklus, sa pomocou príkazu `break` ukončí. Pokiaľ magnet detegovaný neboli, vypíše "Can not detect magnet", no while cyklus pokračuje.

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

```

float AeroShield :: begin(bool isDetected){
    pinMode(AERO_UPIN,OUTPUT);           // Pin aktuatora

    #ifdef 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 2.11: 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 východznej 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...ä 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 2.12: Zdrojový kód funkcie calibration.

Funkcia convertRawAngleToDegrees()

Ako sme už spomínali v sekcií 2.1.1, zaznamenaná hodnota uhlu kyvadla je v rozmedzí od 0 do 4096 a tieto hodnoty priamo korešpondujú zo stupňami od 0° do 360° . 1° 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 2.13: 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 2.14: 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 2.15: 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 2.1.1. 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 premenívych, ktoré sme získali vďaka meraniam prúdu pomocou multimetra, a následným porovnaním hodnôt zo senzora. Porovnaním hodnôt sme získali veľkosť korekčných premenívych, vďaka čomu sa naše merania zhodujú s meraniami pomocou multimetra, na dve desatinné miesta.

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 0.000A.

```

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

    float currentMean= current/repeatTimes;
    // Vypocet priemernej hodnoty
    currentMean= currentMean-correction2; // Korekcia
    if(currentMean < 0.000) currentMean= 0.000;
    // Korekcia nulovej hodnoty
}

```

```

        current= 0;           // Vynulovanie pomocnych premennych
        voltageValue= 0;      // Vynulovanie pomocnych premennych
        return currentMean; // Vrat hodnotu prudu v amperoch
    }
}

```

Zdrojový kód 2.16: Zdrojový kód funkcie currentMeasure.

2.2.3 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 rezitora v ohmoch
correction1 = 4.1220;        % korekcia
correction2 = 0.33;          % korekcia
repeatTimes = 50;            % pocet cyklov pre vypocet priemer
                             % hodnoty prudu
end

```

Zdrojový kód 2.17: 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`, nebudem 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 I²C. Poslednou funkciou je vypísanie správy a konfiguráciu pinu `AERO_UPIN` ako výstup.

```

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

```

Zdrojový kód 2.18: Knižnica AeroShield.m properties.

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

3 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.

3.1 Programy v otvorenej slučke, bez späťnej väzby

3.1.1 *AeroShieldOpenLoop.ino*

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 3.1: 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. 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 začiatokného a koncového uhla.

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

```

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

```

Zdrojový kód 3.2: 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.3.1, 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 || ");

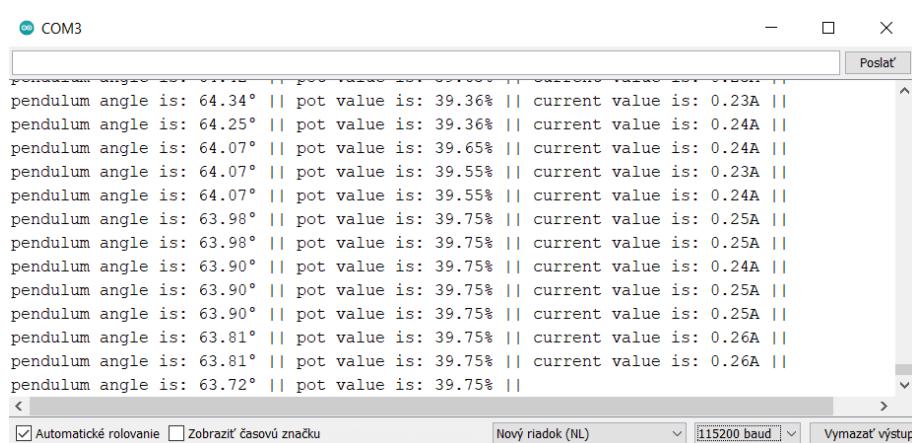
    referencePercent= AeroShield.referenceRead(); // Citanie
    potenciometra
    Serial.print("pot value is: ");
    Serial.print(referencePercent);
    Serial.print("% || ");

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

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

```

Zdrojový kód 3.3: AeroShield open loop loop().



Obr. 3.1: Výstup z programu AeroShieldOpenLoop.ino.

3.1.2 AeroShieldOpenLoop.m

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 3.4: AeroShield open loop inicializacia.

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

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

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

```

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

tic % time keeping

```

Zdrojový kód 3.5: 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.3.2. 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° , 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

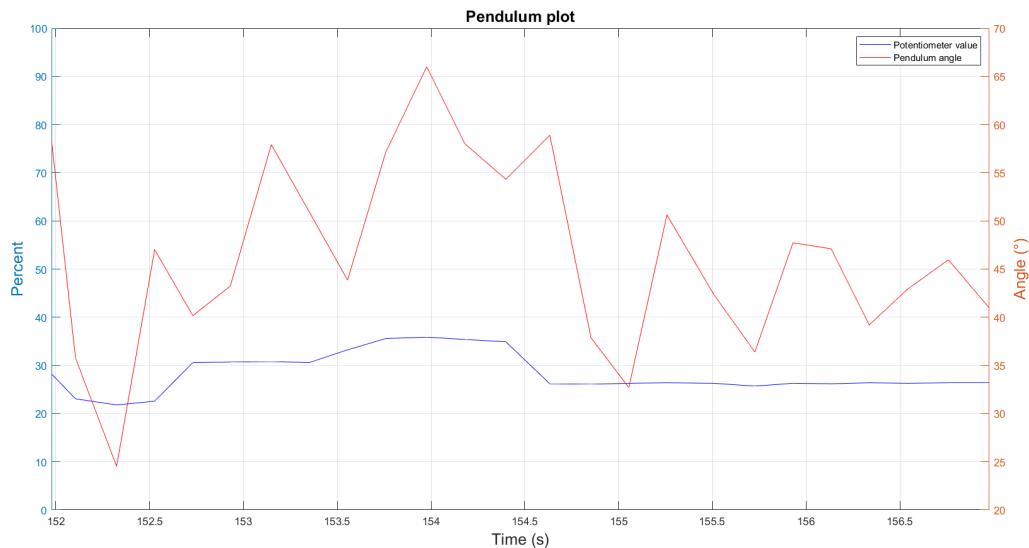
    count = count + 1; % cycle counter
    time(count) = toc; % time keeping
    angle(count) = angle_(1); % angle value in
    time
    percenta= mapped(pwm, 0.0, 5.0, 0.0, 100.0); % map pwm to
    percent
    potentiometer(count) = percenta(1); % pententiometer
    value in time
    set(plotGraph, 'XData',time,'YData',angle); % 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 3.6: AeroShield open loop, while cyklus.



Obr. 3.2: Výstup z programu AeroShieldOpenLoop.m.

3.2 Programy v uzavorennej slučke, so spätnou väzbou

sampling aj

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] Fanavarjan Sharif. Aero pendulum control system (pr22). Store. Online., 2021. 2021, <https://www.zoodel.com/en/product/ZP22344/Aero-Pendulum-Control-System-PR22>.
- [2] 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>.
- [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] Gergely Takacs. Automationshield. Wiki. Online., 2021. 13.7.2021, <https://github.com/gergelytakacs/AutomationShield/wiki>.
- [6] Gergely Takacs. Automationshield. Code. Online., 2021. 23.12.2021, <https://github.com/gergelytakacs/AutomationShield>.
- [7] Harry Baggen. The javelin stamp. *Elector Electronics*, 1(1):0, 2003.
- [8] Arduino uno rev3. Info. Online., 2021. 2021, <https://store.arduino.cc/products/arduino-uno-rev3>.
- [9] Texas instruments tps56339 buck converters. store. Online., 2021. 2021, <https://www.mouser.ee/new/texas-instruments/ti-tps56339-buck-converters/>.
- [10] Arduino. Overview of the arduino uno components. article. Online., 2021. 2021, <https://docs.arduino.cc/tutorials/uno-rev3/intro-to-board>.
- [11] Arduino uno r3 - schematic with ch340. article. Online., 2021. 2021, http://electrooobs.com/eng_arduino_tut31_sch3.php.
- [12] DANIELLE COLLINS. What are coreless dc motors? article. Online., 2018. 09.10.2021, <https://www.motioncontrolltips.com/what-are-coreless-dc-motors/>.

- [13] 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.
- [14] Pmv45en2. shop. Online., 2021. 2021, <https://www.nexperia.com/products/mosfets/small-signal-mosfets/PMV45EN2.html>.
- [15] 7mm diameter 720 coreless motor for quadcopter. shop. Online., 2021. 2021, <https://www.elecrow.com/7mm-diameter-720-coreless-motor-for-quadcopter.html>.
- [16] SHAWN HYMEL. Ina169 breakout board hookup guide. article. Online., 2021. 2021, <https://learn.sparkfun.com/tutorials/ina169-breakout-board-hookup-guide/all>.
- [17] Ina169na/250. store. Online., 2021. 2021, <https://www.ti.com/store/ti/en/p/product/?p=INA169NA/250>.
- [18] The Editors of Encyclopaedia Britannica. Hall effect. article. Online., 2021. 2021, <https://www.britannica.com/science/Hall-effect>.
- [19] 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>.
- [20] 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/>.
- [21] Writing a library for arduino. article. Online., 2022. 2022, <https://docs.arduino.cc/hacking/software/LibraryTutorial>.
- [22] Arduino - data types. article. Online., -. 2022, https://www.tutorialspoint.com/arduino/arduino_data_types.htm.
- [23] Michael Dellnitz Martin Golubitsky. Linear mappings and bases. article. Online., -. 2022, <https://ximera.osu.edu/laode/linearAlgebra/linearMapsAndChangesOfCoordinates/linearMappingsAndBases>.
- [24] Nicholas Zambetti. A guide to arduino & the i2c protocol (two wire). article. Online., 2022. 2022, <https://docs.arduino.cc/learn/communication/wire>.
- [25] 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
19     pendulumAngle= AutomationShield.mapFloat(AeroShield.getRawAngle(),
20         startangle ,lastangle ,0.00 ,90.00 );
21     Serial.print("pendulum angle is: ");
22     Serial.print(pendulumAngle);
23     Serial.print(" degree || ");
24
25     referencePercent= AeroShield.referenceRead();
26     Serial.print("pot value is: ");
27     Serial.print(referencePercent);
28     Serial.print(" percent || ");
29
30     AeroShield.actuatorWrite(referencePercent);
31
32     CurrentMean= AeroShield.currentMeasure();
33     Serial.print("current value is: ");
34     Serial.print(CurrentMean);
35     Serial.println("A || ");
```

Zdrojový kód 4.3: Zdrojový kód súboru AeroShieldOpenLoop.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 4.4: 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 4.5: Zdrojový kód súboru AeroShieldOpenLoop.m.