

**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

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
2.2 Software	14
2.2.1 Header	14
2.2.2 Source	16
3 Didaktické príklady	25
4 Záver	26
Literatúra	27
Zdrojový kód AeroShield.h	iv
Zdrojový kód AeroShield.cpp	vi

Zoznam obrázkov

1.1	dočasný obrázok pokiaľ nebude hotovec final.	2
1.2	Aeropendulum značky Real Sim[1].	3
1.3	Arduino UNO.	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	akčný člen	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

Zoznam zdrojových kódov

2.1	Ukážka zdrojového kódu headeru.	15
2.2	Triedy a objekty.	15
2.3	Source volanie funkcie.	16
2.4	Zdrojový kód funkcie mapFloat.	17
2.5	Zdrojový kód funkcie serialPrint.	17
2.6	Zdrojový kód funkcie readOneByte.	18
2.7	Zdrojový kód funkcie readTwoBytes.	18
2.8	Zdrojový kód funkcie detectMagnet.	19
2.9	Zdrojový kód funkcie getMagnetStrength.	19
2.10	Zdrojový kód funkcie getRawAngle.	20
2.11	Zdrojový kód funkcie begin.	21
2.12	Zdrojový kód funkcie calibration.	21
2.13	Zdrojový kód funkcie convertRawAngleToDegrees.	22
2.14	Zdrojový kód funkcie referenceRead.	22
2.15	Zdrojový kód funkcie actuatorWrite.	23
2.16	Zdrojový kód funkcie currentMeasure.	23
4.1	Zdrojový kód súboru AeroShield.h.	iv
4.2	Zdrojový kód súboru AeroShield.cpp.	vi

Zoznam tabuliek

2.1 Dátové typy	16
---------------------------	----

Úvod

Cieľom tejto bakalárskej 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 drahé [2]. Z toho dôvodu, 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.

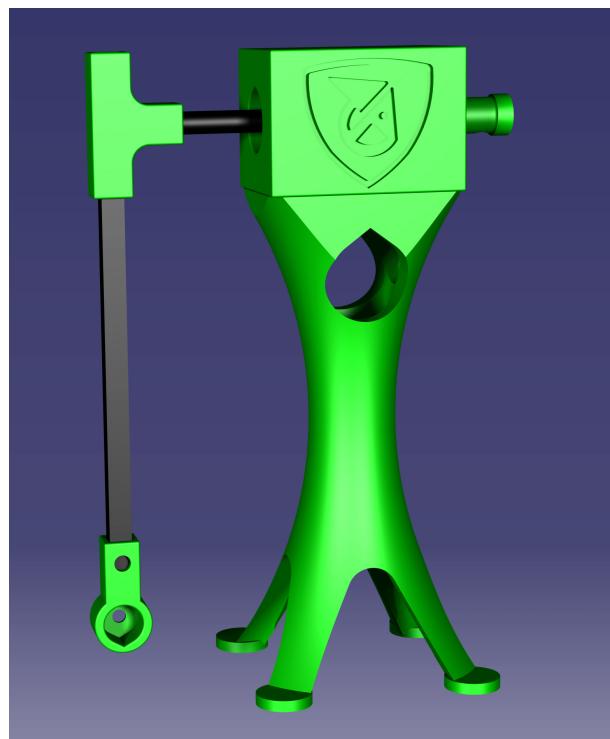
V bakalárskej práci je opísaný postup výroby a fungovania shieldu s dôrazom na zrozumiteľnosť jednotlivých aspektov aj čitateľom, ktorý o danej téme nie sú dokonale oboznámený. Na začiatku bakalárskej práce, v časti hardware, je opísaný základný princíp fungovania shieldu a následne jeho jednotlivé súčiastky. 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 Dip-Trace.

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

Cieľom tejto bakalárskej práce je návrh učebnej pomôcky AeroShield, na štýl experimentu celosvetovo známeho ako 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 (ďalej len „hall efekt“) alebo iné [3]. V našom prípade budeme používať senzor hall efektu ktorého fungovanie je opísané v časti hardware 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: dočasný obrázok pokiaľ nebude hotovec final.

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 pro-

jektu je tvorba rozširujúcich dosiek (shieldov) vyvýjaných pre populárny typ prototypizačných dosiek s mikrokontrolérmi Arduino, ktoré majú za cieľ lepšiu výučbu strojného inžinierstva, mechatroniky a riadenia [4].

Zdrojový kód k AeroShieldu, ako aj ku všetkým modulom AutomationShield, nájdeme na platforme GitHub [5], 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.

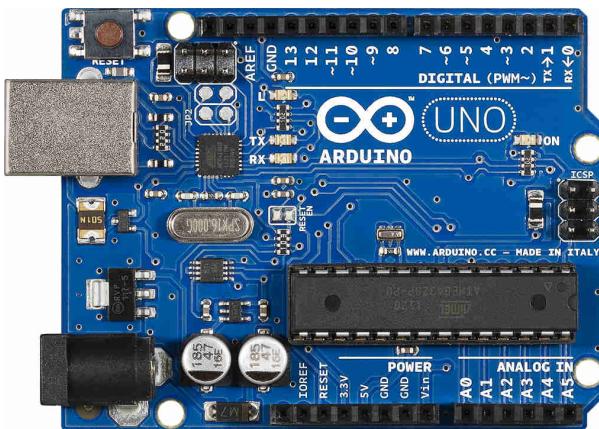
Ako už bolo spomenuté, hlavnou motiváciou tohto projektu je nízka dostupnosť a vysoká cena podobných učebných pomôcok. Výučba je preto častokrát až príliš zameraná na memorovanie faktov a teórie, namiesto praktických experimentov a skúseností typu pokus-omyl. Jediný podobný dostupný produkt na kúpu nie ako kit, je Aeropendulum od neznámej 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 [6], 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 [7]). 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 \sim , tieto piny sú schopné produkovať PWM³ signál ktorý potrebujeme na správne ovládanie motoru kyvadla.



Obr. 1.3: Arduino UNO.

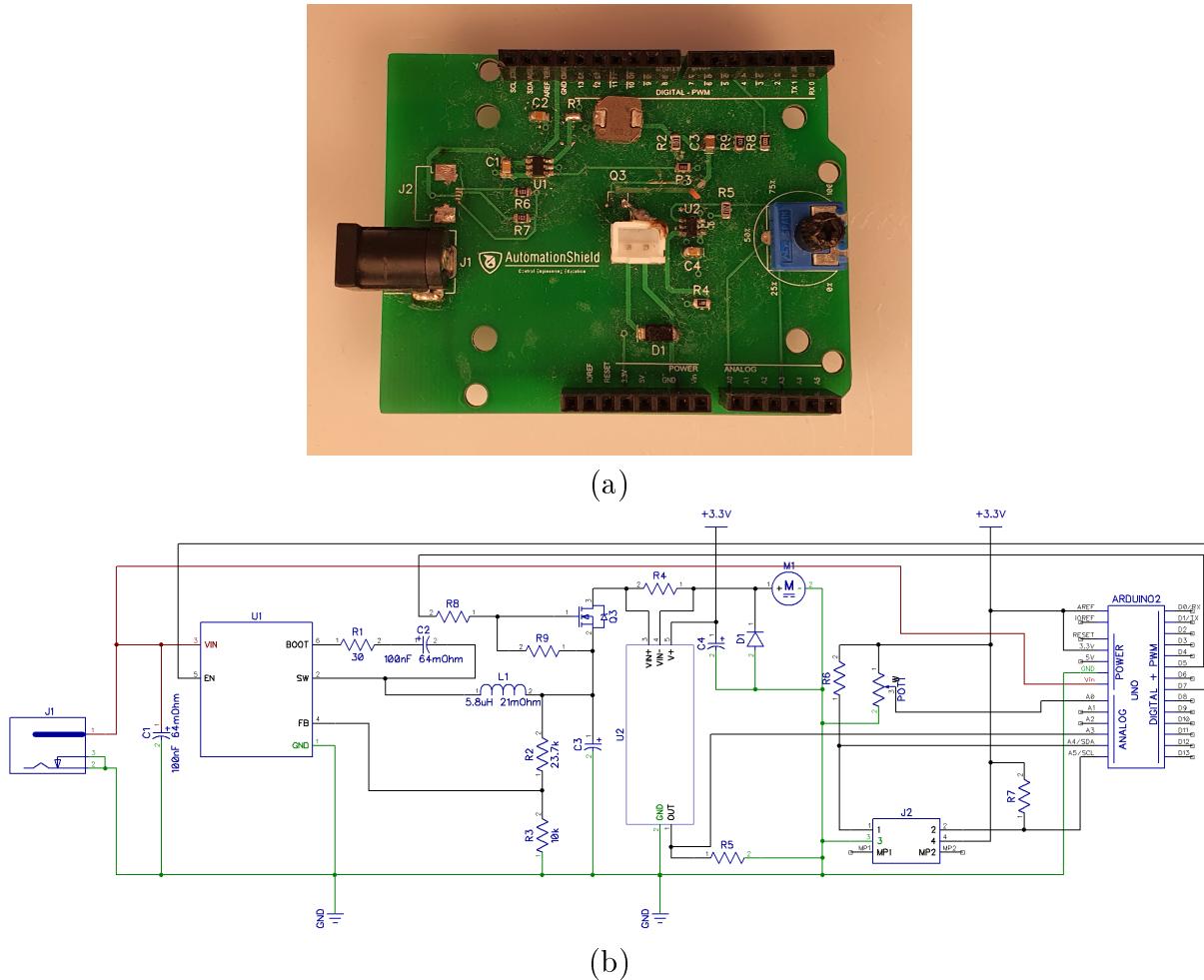
²Tento fakt platí pri mikročipoch typu DIP(Dual in-line package) ktoré stačí jednoducho vytiahnuť z konektora bez použitia spájkovania.

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 kvyadla vznikla ako záverečný projekt na predmet Mikroprocesorová technika. Na projekte pracovala päťica študentov:

. Schému zapojenia hlavnej dosky, ako aj fotografiu naspájkovej verzie môžeme vidieť na obr.2.1.

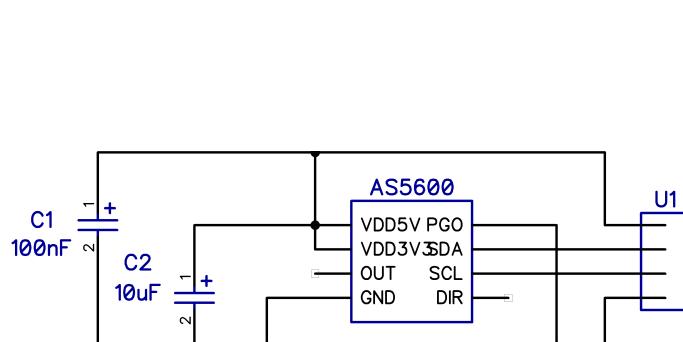


Obr. 2.1: (a) Prvá verzia AeroShieldu. (b) Schéma zapojenia prvej verzie AeroShieldu.

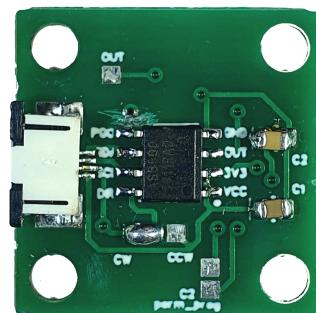
Prvá verzia dosky mala niekoľko nedostatkov, vďaka ktorým bola prakticky nepoužiteľná. Hlavnými nedostatkami boli:

- 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 najskôr pochopiť jednotlivé časti zapojenia, analyzovať chyby a ich následná oprava. V rámci školského projektu bola vytvorená hlavná doska na ktorej sa nachádza väčšina elektroniky, avšak bola vytvorená aj verzia menšej dosky 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

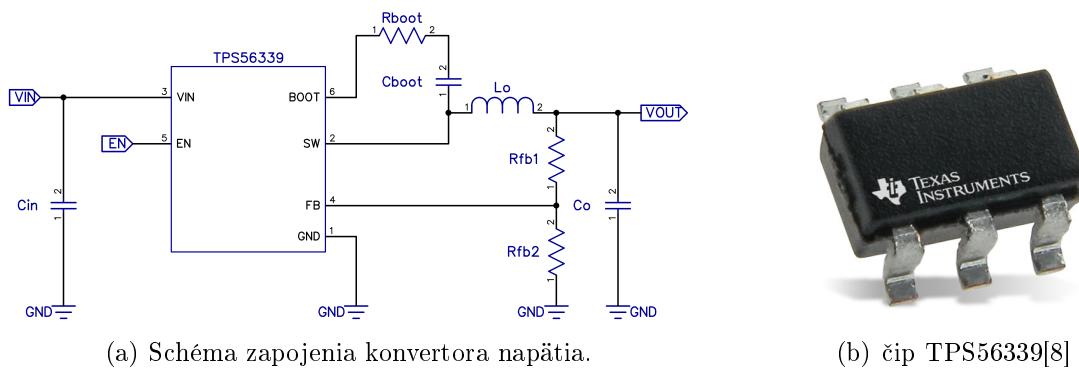
V tejto časti sa bližšie pozrieme na jednotlivé nevyhnutné súčasti zapojenia AeroS-shieldu. Konkrétnie sa jedná o tieto prvky:

- 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 ,2A, ktoré by motor v priebehu chvíle zničilo. Na zníženie 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 pomoc 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[8].



(a) Schéma zapojenia konvertora napäťia.

(b) čip TPS56339[8]

Obr. 2.3: buck converter

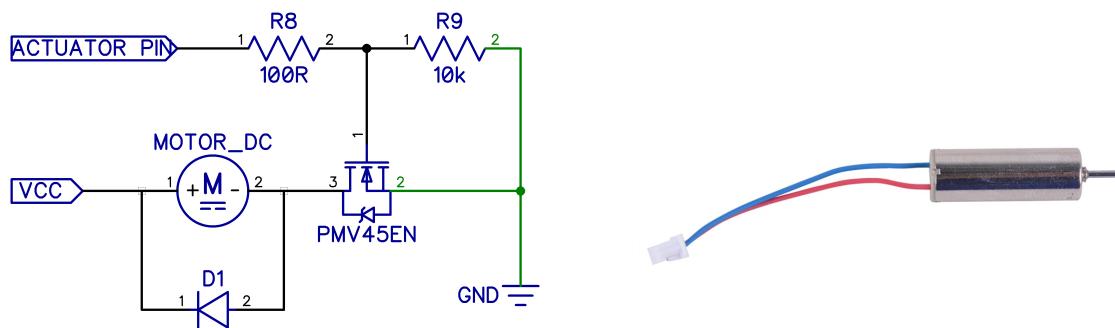
Na čip je privádzané napäťie 12V ktoré sa pomocou zapojenia viditeľného na schéme obr.2.3.a., znižuje na napäťie 3,7V. Napájanie motora musí byť realizované externe pomocou koaxiálneho napájacieho konektora, z dôvodu vysokého prúdu odoberaného motorom počas silného zaťaženia. Rovnaký konektor sa sice 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[9][10].

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[11]. 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%-[12]. 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[13].



(a) Schéma zapojenia motorčeka.

(b) Motorček[14]

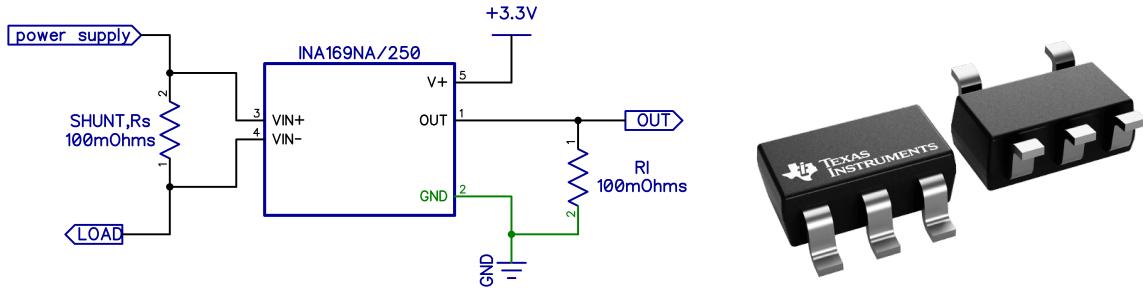
Obr. 2.4: akčný člen

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 napätia na stranách shunt rezistora obr.2.5.a. Na základe nameraného úbytku napätia, 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 $I_s = \frac{V_{OUT} x 1k\Omega}{R_s x R_l}$ 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 [15].



(a) Schéma zapojenia snímača prúdu.

(b) Senzor INA169NA/250[16]

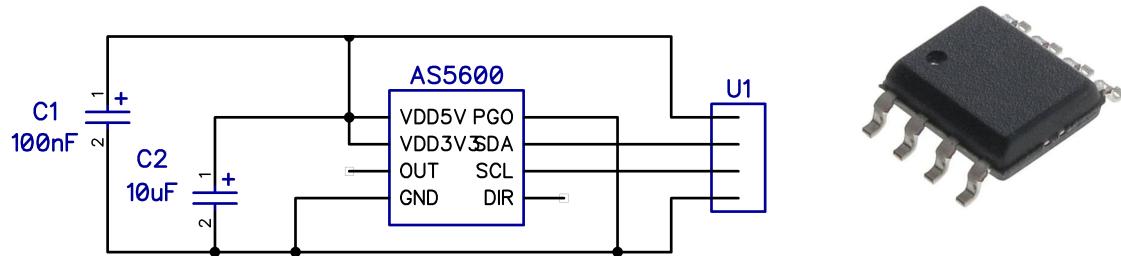
Obr. 2.5: meranie prúdu

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 bezkontaktnou 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[17]. 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 prevodníkom (ADC). Snímaná je aj intenzita magnetického poľa, ktorá sa ďalej používa na automatické riadenie zosilnenia(AGC), ktoré slúži na kompenzáciu teploty a veľkosti magnetického poľa.

Na výber sú dva typy výstupu a to analógový výstup alebo digitálny výstup s kódovaním PWM. Senzor má taktiež aj 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[18]

Obr. 2.6: meranie uhla kyvadla

2.1.2 Schéma zapojenia

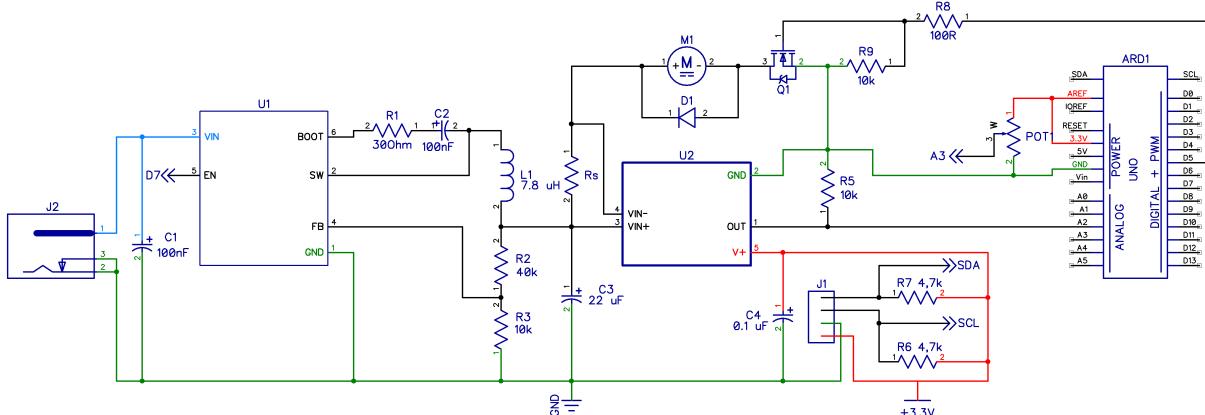
Všetky schémy zapojenia boli tvorené v bezplatnej verzii programu DipTrace. DipTrace slúži ako prostredie na tvorbu elektrotechnických schém, potrebných pre výrobu dosiek plošných spojov, ako aj pre účely prehľadnosti zapojenia komponentov na týchto doskách. Program v sebe zahŕňa časť pre tvorbu samotných 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 dobrou 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



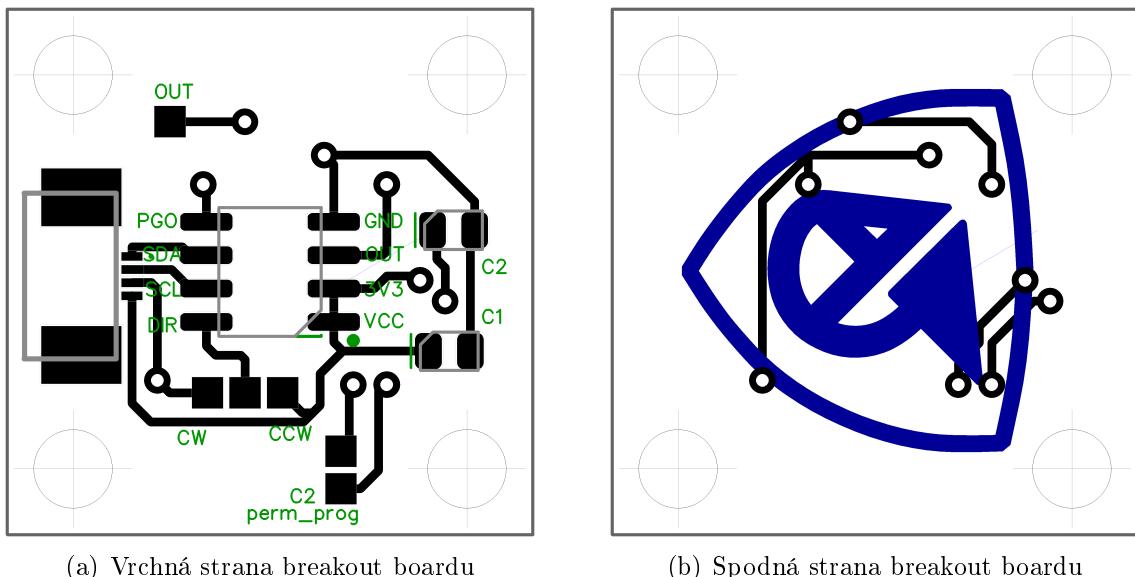
Obr. 2.7: Schéma zapojenia AeroShieldu

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[19].

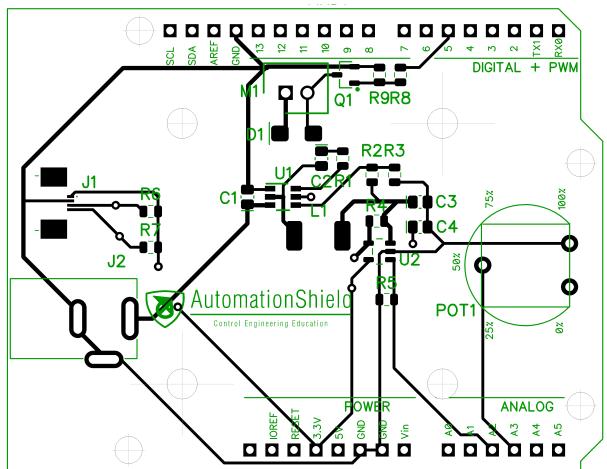
Výhodou je fakt, že vodivé spojenia medzi jednotlivými súčiastkami zapojenia, sú realizované vrstvou medzi ktorá je ukrytá pod ochrannými vrstvami povrchu dosky, na rozdiel od typických kálových spojení. Káble majú niekoľko nedostatkov ako to že sa vedia ľahko vypojiť, ľahko dochádza k ich porušeniu 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.

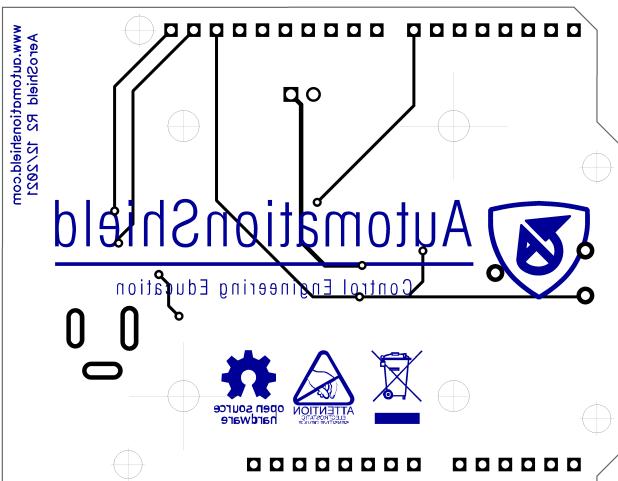


Obr. 2.8: Vedľajšia doska AeroShieldu- breakout board

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)

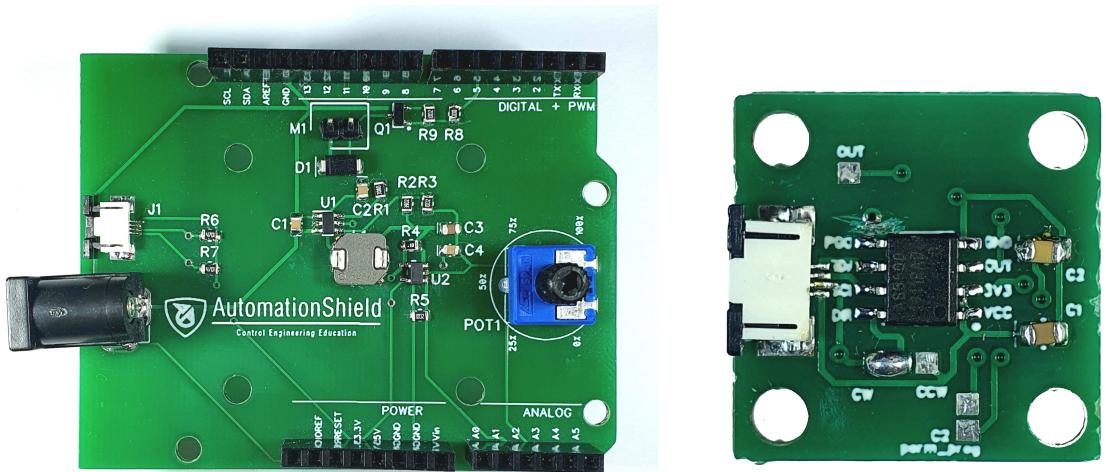
Obr. 2.9: (a) Vrchná strana AeroShieldu (b) Spodná strana AeroShieldu

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, veľkosti dier a priestoru okolo dier, veľkosti 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

2.2 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++[20]. 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
```

Zoznam zdrojových kódov 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
};


```

Zoznam zdrojových kódov 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()
```

Zoznam zdrojových kódov 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[21] (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 2.1: 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[22].

```
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;
}
```

Zoznam zdrojových kódov 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
}
```

Zoznam zdrojových kódov 2.5: 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 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. Ked' 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[23].

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_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); // Ziadost na odpoved
    while (Wire.available() == 0); // Cakaj pokial odpoved nepride

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

    return retVal; // Zaslanie odpovede
}
```

Zoznam zdrojových kódov 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_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); // Ziadost na odpoved
    while (Wire.available() == 0); // Cakaj pokial odpoved nepride

    int low = Wire.read();      // Uloženie prvého 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
```

⁷Bitový posun je operácia vykonávaná so všetkými bitmi binárnej hodnoty, pri ktorej sa posúvajú o určený počet miest doľava alebo doprava[24].

```

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
}

```

Zoznam zdrojových kódov 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
}

```

Zoznam zdrojových kódov 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
}

```

Zoznam zdrojových kódov 2.9: Zdrojový kód funkcie getMagnetStrength.

Funkcia getRawAngle()

Poslednou funkciu komunikácie s hall senzorom je word AeroShield::getRawAngle(). Funkcia slúži na čítanie samotného uhlia 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, ktory rovno vrati vysledok pomocou
                                                    prikazu return
}

```

Zoznam zdrojových kódov 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");
            }
        }
    }
}

```

Zoznam zdrojových kódov 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
}

```

Zoznam zdrojových kódov 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;      //  $360/4096=0.087 \times \text{rawHodnota}$ 
    return ang;                      // Vrat hodnotu
}

```

Zoznam zdrojových kódov 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
}

```

Zoznam zdrojových kódov 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
}

```

Zoznam zdrojových kódov 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 skresľuje výslednú hodnotu. Vďaka for cyklu získame priemernú hodnotu prúdu.

Výsledná hodnota prúdu prechádza úpravami, pomocou dvoch korekčných premeníných, ktoré sme získali vďaka meraniam prúdu pomocou multimetra, a následným porovnaním hodnôt zo senzora. Porovnaním hodnôt sme získali veľkosť korekčných premeníných, 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
}
```

Zoznam zdrojových kódov 2.16: Zdrojový kód funkcie currentMeasure.

píš aj o samplingu

3 Didaktické príklady

pid cislicovom riadeni grafy vystupov

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] Petr Horáček. Laboratory experiments for control theory courses: A survey. *Annual Reviews in Control*, 24:151–162, 2000.
- [3] Ed Edwards. All about position sensors. article. Online. 2021, <https://www.thomasnet.com/articles/instruments-controls/all-about-position-sensors>.
- [4] Gergely Takacs. Automationshield. Wiki. Online., 2021. 13.7.2021, <https://github.com/gergelytakacs/AutomationShield/wiki>.
- [5] Gergely Takacs. Automationshield. Code. Online., 2021. 23.12.2021, <https://github.com/gergelytakacs/AutomationShield>.
- [6] Harry Baggen. The javelin stamp. *Elector Electronics*, 1(1):0, 2003.
- [7] Arduino uno rev3. Info. Online., 2021. 2021, <https://store.arduino.cc/products/arduino-uno-rev3>.
- [8] Texas instruments tps56339 buck converters. store. Online., 2021. 2021, <https://www.mouser.ee/new/texas-instruments/ti-tps56339-buck-converters/>.
- [9] Arduino. Overview of the arduino uno components. article. Online., 2021. 2021, <https://docs.arduino.cc/tutorials/uno-rev3/intro-to-board>.
- [10] Arduino uno r3 - schematic with ch340. article. Online., 2021. 2021, http://electronoobs.com/eng_arduino_tut31_sch3.php.
- [11] DANIELLE COLLINS. What are coreless dc motors? article. Online., 2018. 09.10.2021, <https://www.motioncontrolltips.com/what-are-coreless-dc-motors/>.
- [12] Komatsu Yasuhiro, Tur-Amgalan Amarsanaa, Yoshihiko Araki, Syed Abdul Kadir Zawawi, and Takamura Keita. Design of the unidirectional current type coreless dc brushless motor for electrical vehicle with low cost and high efficiency. In *SPEEDAM 2010*, pages 1036–1039, 2010.

- [13] Pmv45en2. shop. Online., 2021. 2021, <https://www.nexperia.com/products/mosfets/small-signal-mosfets/PMV45EN2.html>.
- [14] 7mm diameter 720 coreless motor for quadcopter. shop. Online., 2021. 2021, <https://www.elecrow.com/7mm-diameter-720-coreless-motor-for-quadcopter.html>.
- [15] SHAWN HYMEL. Ina169 breakout board hookup guide. article. Online., 2021. 2021, <https://learn.sparkfun.com/tutorials/ina169-breakout-board-hookup-guide/all>.
- [16] Ina169na/250. store. Online., 2021. 2021, <https://www.ti.com/store/ti/en/product/?p=INA169NA/250>.
- [17] The Editors of Encyclopaedia Britannica. Hall effect. article. Online., 2021. 2021, <https://www.britannica.com/science/Hall-effect>.
- [18] 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>.
- [19] 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/>.
- [20] Writing a library for arduino. article. Online., 2022. 2022, <https://docs.arduino.cc/hacking/software/LibraryTutorial>.
- [21] Arduino - data types. article. Online., -. 2022, https://www.tutorialspoint.com/arduino/arduino_data_types.htm.
- [22] Michael Dellnitz Martin Golubitsky. Linear mappings and bases. article. Online., -. 2022, <https://ximera.osu.edu/laode/linearAlgebra/linearMapsAndChangesOfCoordinates/linearMappingsAndBases>.
- [23] Nicholas Zambetti. A guide to arduino & the i2c protocol (two wire). article. Online., 2022. 2022, <https://docs.arduino.cc/learn/communication/wire>.
- [24] Bit shifting. article. Online., 2017. 2022, <https://www.techopedia.com/definition/26846/bit-shifting>.

Zdrojový kód súboru AeroShield.h

```
#ifndef AEROSHIELD_H // Include guard
#define AEROSHIELD_H

// Defining libraries used by the AeroShield
#include "AutomationShield.h" // Include the main library
#include <Wire.h> // Include I2C protocol library
#include <Arduino.h> // Required Arduino API
    in libraries

// Defining pins used by the AeroShield
#define AERO_RPIN A3 // Input from potentiometer
#define VOLTAGE_SENSOR_PIN A2 // Input pin for measuring Vout
#define AERO_UPIN 5 // Motor (Actuator)

class AeroShield{ // Class for the AeroShield device
public:
    AeroShield(void);
    float begin(bool isDetected); // Board initialisation — initialisation of pin modes and variables
    void actuatorWrite(float PotPercent); // Write actuator — function takes input 0.0–100.0 percent and sets motor speed accordingly
    float calibration(word RawAngle); // Board calibration — finding out the 0 degree value in raw format
    float convertRawAngleToDegrees(word newAngle); // Convert raw angle from hall senzor to degrees
    float referenceRead(void); // Read value of potentiometer and converts it to percentual value
    float currentMeasure(void); // Read current from actuator
    int detectMagnet(); // AS5600 detect magnet
    int getMagnetStrength(); // AS5600 magnet strength
    word getRawAngle(); // AS5600 angle value

private:
    int ang; // Variable for angle reading in degrees
    float startangle; // Variable for storing zero angle position
    float referenceValue; // Variable for potentiometer value in percent
    float referencePercent; // Percentual value of potentiometer
```

```

float correction1= 4.1220; // Correction for measuring
    current
float correction2= 0.33; // Correction for measuring
    current
int repeatTimes= 100; // Number of repeats for
    current mean measuring
float voltageReference= 5.0; // Volatage reference in Volts
float ShuntRes= 0.1; // Value of shunt resistor in
    Ohms
float current; // Variable for
    storing current value in Amps
float voltageValue; // Auxiliary variable
int _ams5600_Address = 0x36; // AS5600 address
int _stat = 0x0b; // AS5600
    communication variable
int _raw_ang_hi = 0x0c; // AS5600 communication
    variable
int _raw_ang_lo = 0x0d; // AS5600 communication
    variable
int readOneByte(int in_adr); // AS5600 one byte
    communication
word readTwoBytes(int in_adr_hi, int in_adr_lo); // AS5600 two bytes communication
};

#endif

```

Zoznam zdrojových kódov 4.1: Zdrojový kód súboru AeroShield.h.

Zdrojový kód súboru AeroShield.cpp

```
#include "AeroShield.h" // Include header file

// Initializes hardware pins
float AeroShield::begin(bool isDetected){ // Board initialisation
    pinMode(AERO_UPIN,OUTPUT); // Actuator pin

#ifdef ARDUINO_ARCH_AVR // For AVR
    architecture boards
    Wire.begin(); // Use Wire object
#elif ARDUINO_ARCH_SAM // For SAM
    architecture boards
    Wire1.begin(); // Use Wire1 object
#elif ARDUINO_ARCH_SAMD // For SAMD
    architecture boards
    Wire.begin(); // Use Wire object
#endif

    if(isDetected == 0){ // If magnet not detected go on
        while(1){
            // Go forever until magnet detected
            if(isDetected == 1){ // If magnet detected
                AutomationShield.serialPrint("Magnet_detected\n");
                // Print information then break
                break;
            } else { // If magnet not detected
                AutomationShield.serialPrint("Can_not_detect_magnet\n");
                // Print information then go back to check while statement
            }
        }
    }
}
```

```

        }
    }

float AeroShield::convertRawAngleToDegrees(word newAngle) {
    // Function for converting raw angle(0-4096) to
    degrees(0-360 degree)
    float retVal = newAngle * 0.087;
    // 360 degree
    /4096=0.087 degree times the raw value
    ang = retVal;
    return ang;

    // Return angle value in degrees
}

float AeroShield::calibration(word RawAngle) {
    // Calibration
    AutomationShield.serialPrint("Calibration_running...\\n");
    // Print info
    startangle=0;

    // Zero out Variable(precaution)
    analogWrite(AERO_UPIN,50);
    // Power the
    actuator, swing the pendulum
    delay(250);

    // Wait for 0.25s
    analogWrite(AERO_UPIN,0);
    // Actuator
    powered off, pendulum goes to zero position
    delay(4000);

    // Wait for pendulum to stop oscilating

    startangle = RawAngle;
    // Save
    the value of zero pozition in raw format
    analogWrite(AERO_UPIN,0);
    // Actuator
    powered off(precaution)
    for(int i=0;i<3;i++){
        // Simple
        sound indication of successful calibration 3 beeps
        analogWrite(AERO_UPIN,1);
        //
        // Actuator powereded just a bit so the rotor
        // doesn't turn just beep
        delay(200);

        // wait
        analogWrite(AERO_UPIN,0);
}

```

```

        // Actuator powered off
delay(200);

        // wait
}

AutomationShield.serialPrint("Calibration done");
return startangle;
        // Return start angle
}

float AeroShield::referenceRead(void) {
    // Reference read
    referencePercent = AutomationShield.mapFloat(analogRead(
        AERO_RPIN), 0.0, 1024.0, 0.0, 100.0); // Remaps the
    // analog value from original range 0.0–1023 to percentual
    // range 0.0–100.0
    return referencePercent;
    // Returns the percentual position of potentiometer
    runner
}

void AeroShield::actuatorWrite(float PotPercent) {
    // Actuator write
    float mappedValue = AutomationShield.mapFloat(PotPercent,
        0.0, 100.0, 0.0, 255.0); // Takes the float type
    // percentual value 0.0–100.0 and remaps it to range
    0.0–255.0
    mappedValue = AutomationShield.constrainFloat(mappedValue
        , 0.0, 255.0); // Constrains the
    // remapped value to fit the range 0.0–255.0 — safety
    // precaution
    analogWrite(AERO_UPIN, (int)mappedValue);
    // Write
    remapped value to actuator pin
}

float AeroShield::currentMeasure(void){
    // Measuring
    current drawn by DC motor
    for(int i=0 ; i<repeatTimes ; i++){
        // Function for calculating mean current value
        voltageValue=analogRead(VOLTAGE_SENSOR_PIN);
        // Read
        // a value from the INA169
        voltageValue= (voltageValue * voltageReference) /
        1024; // Remap
        // the ADC value into a voltage number (5V
        // reference)
}

```

```

        current= current + correction1-(voltageValue /
        (10 * ShuntRes));                                //
        Equation given by the INA169 datasheet to
        // determine the current flowing through ShuntRes
        . RL = 10k
    }

    // Is = (Vout x 1k) / (RS x RL)

    float currentMean= current/repeatTimes;           //
    Calculating mean current value
    currentMean= currentMean-correction2;             //
    Small correction of current value(determined by
    multimeter)
    if (currentMean < 0.000){

        // Correction for occasional bug causing the value to
        be negative.
        currentMean= 0.000;

        // When it so happens, zero out the value.
    }

    current= 0;

    // Zero out current value
    voltageValue= 0;

    // Zero out voltage value
    return currentMean;

    // Return mean current value
}

word AeroShield::getRawAngle()                         //
{
    Function for getting raw pendulum angle data 0-4096
{
    return readTwoBytes(_raw_ang_hi, _raw_ang_lo);      //
    // Another
    function for communication with senzor, called from
    this library
}

int AeroShield::detectMagnet()                         //
{
    Function for detecting presence of magnet
{
    int magStatus;

    // Auxiliary variable
}

```

```

int retVal = 0;

    // Auxiliary variable
magStatus = readOneByte(_stat);

    // Another function for communication with senzor,
    // called from this library

if (magStatus & 0x20)
retVal = 1;

return retVal;

    // Return value
}

int AeroShield::getMagnetStrength()                                // Function
for getting the strength of magnet
{
    int magStatus;                                                 //
    Auxiliary variable
    int retVal = 0;                                                 //
    Auxiliary variable
    magStatus = readOneByte(_stat);                                //
    Another function for communication with senzor, called
    from this library

if (detectMagnet() == 1)                                         //
    Return 0 if no magnet is detected
{
    retVal = 2;                                                   //
    // Return 2 if magnet is just right
    if (magStatus & 0x10)
    retVal = 1;                                                 //
    // Return 1 if magnet is too weak
    else if (magStatus & 0x08)
    retVal = 3;                                                 //
    // Return 3 if magnet is too strong
}

return retVal;
    // Return value
}

int AeroShield::readOneByte(int in_addr)                            // Function
for communicating with the senzor using 1 Byte
{
    int retVal = -1;
Wire.beginTransmission(_ams5600_Address);                         //
    Initialise wire transmission
Wire.write(in_addr);                                              //
    Write 4 bits
Wire.endTransmission();                                            //
    // End

```

```

        wire transmission
Wire.requestFrom(_ams5600_Address, 1); // Request answer
while (Wire.available() == 0); // Wait for returning bits

retVal = Wire.read(); // Store returning bits

return retVal; // Return stored bits
}

word AeroShield::readTwoBytes(int in_adr_hi, int in_adr_lo)
    // Function for communicating with the senzor using 2 Bytes
{
    word retVal = -1;

    /* Read Low Byte */
    Wire.beginTransmission(_ams5600_Address); // Initialise wire transmission
    Wire.write(in_adr_lo); // Write 4 bits
    Wire.endTransmission(); // End wire transmission
    Wire.requestFrom(_ams5600_Address, 1); // Request answer
    while (Wire.available() == 0); // Wait for returning bits

    int low = Wire.read(); // Store first returning bits

    /* Read High Byte */
    Wire.beginTransmission(_ams5600_Address); // Initialise wire transmission
    Wire.write(in_adr_hi); // Write 4 bits
    Wire.endTransmission(); // End wire transmission
    Wire.requestFrom(_ams5600_Address, 1); // Request answer
    while (Wire.available() == 0); // Wait for returning bits

    word high = Wire.read(); // Store second returning bits

    high = high << 8; // bitwise left shift
    retVal = high | low;

    return retVal; // 
}

```

```
    Return stored bits  
}
```

Zoznam zdrojových kódov 4.2: Zdrojový kód súboru AeroShield.cpp.