Středoškolská technika 2014

Setkání a prezentace prací středoškolských studentů na ČVUT

Automaticky stabilizované 4kolové vozidlo

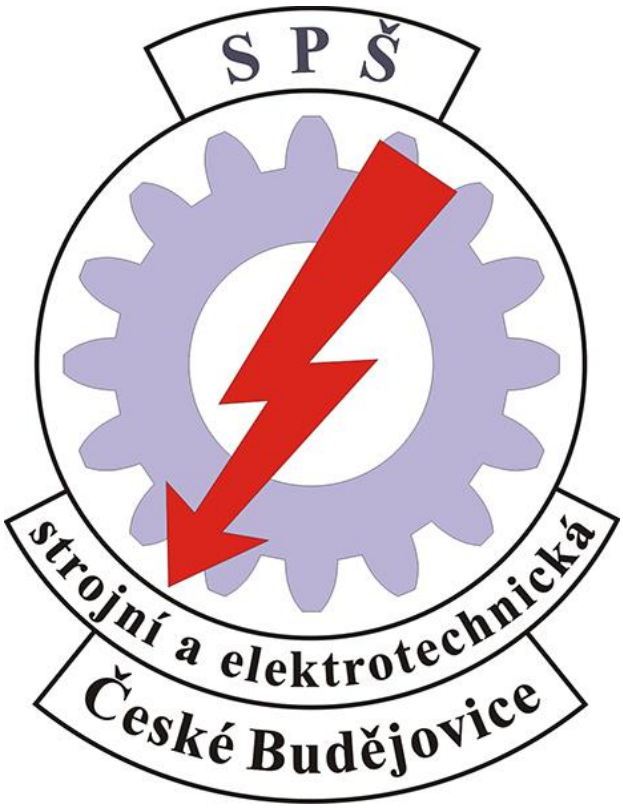
Jan Mánek

STŘEDNÍ PRŮMYSLOVÁ ŠKOLA STROJNÍ A ELEKTROTECHNICKÁ

ČESKÉ BUDĚJOVICE, DUKELSKÁ 13

STŘEDNÍ PRŮMYSLOVÁ ŠKOLA STROJNÍ A ELEKTROTECHNICKÁ

ČESKÉ BUDĚJOVICE, DUKELSKÁ 13



Návrh a realizace dálkově ovládaného, automaticky stabilizovaného 4kolového vozidla

Autor práce: Jan Mánek

Vedoucí práce: Ing. Jan Janoud

Oponent: Ing. Oldřich Smutný

Školní rok: 2013/2014

Prohlašuji, že jsem tuto maturitní práci vypracoval samostatně. Veškeré použité podklady ze kterých jsem čerpal informace, jsou uvedeny v seznamu použité literatury a citovány v textu podle normy ČSN ISO 690.

V Č. Budějovicích dne: ……

Jan Mánek

Děkuji ing. Janu Janoudovi za odborné vedení práce, věcné připomínky, dobré rady a vstřícnost při konzultacích a vypracovávání maturitní práce.

Střední průmyslová škola strojní a elektrotechnická České Budějovice, Dukelská 13

Zadání odborné maturitní práce

Zadání je vystaveno pro žáka

Jméno a příjmení: **Jan Mánek**
Datum narození: **31. 5. 1995**
Žák třídy: **4. EA**

Téma maturitní práce:

Návrh a realizace dálkově ovládaného, automaticky stabilizovaného 4kolového vozidla

Požadované dílčí body práce:

1. **Návrh a výroba podvozku pro vozidlo**
2. **Výběr vhodného mikroprocesoru pro řízení vozidla**
3. **Výběr vhodných modelářských serv**
4. **Úprava modelářských serv pro použití v roli pohonu**
5. **Výběr vhodných senzorů**
6. **Návrh a napsání programu pro vybraný mikroprocesor**

Cíl maturitní odborné práce:

Návrh a výroba všech součástí, zajištění vzájemné funkčnosti a zajištění co nejpřesnější a nejrychlejší stabilizace.

Rozsah práce: **minimálně 20 stran textu + elektrické schéma**

Termín odevzdání práce: **do 21. dubna 2014 ve dvou vyhotoveních**

Vedoucí práce: **Ing. Jan Janoud**

Oponent: **Ing. Oldřich Smutný**

Kritéria hodnocení práce: **Původnost, technická úroveň návrhu, formální úroveň zprávy, úroveň ústní prezentace při obhajobě práce.**

V Č. Budějovicích dne: 20. 12. 2013

Ing. Čestmír Tschauder,
ředitel školy

Tato práce se zabývá možnostmi dnešních mikroprocesorů, senzorů a dalších obvodů. Práce se dále zabývá řešením problémů, které se vyskytly při konstrukci robota, psaní programu a následném testování.

OBSAH

1. Úvod

Dnešní mikroprocesory zažívají v poslední době velký nárůst uživatelů, díky tomu se velmi snížila jejich cena, rozšiřuje se komunita, která je používá, tudíž je výrazně jednodušší nalézt příčinu problému, když se nějaký vyskytne, je jednodušší zprovoznit daný podpůrný obvod, či senzor, je možné sehnat určitý obvod, který je se základními součástkami již osazen na desce a je plně kompatibilní s daným mikroprocesorem.

Díky projektu Arduino je dnes možné nahrát program do mikroprocesoru bez potřeby programátoru procesoru, stačí vlastnit obvod pro sériovou komunikaci s PC a díky jazyku Wiring se výrazně zjednodušilo napsání samotného programu pro mikroprocesory řady ATMega od firmy Atmel.

V této práci bych rád přiblížil některé vlastnosti těchto mikroprocesorů, jejich možnosti a dále aspekty některých senzorů a pomocných obvodů.

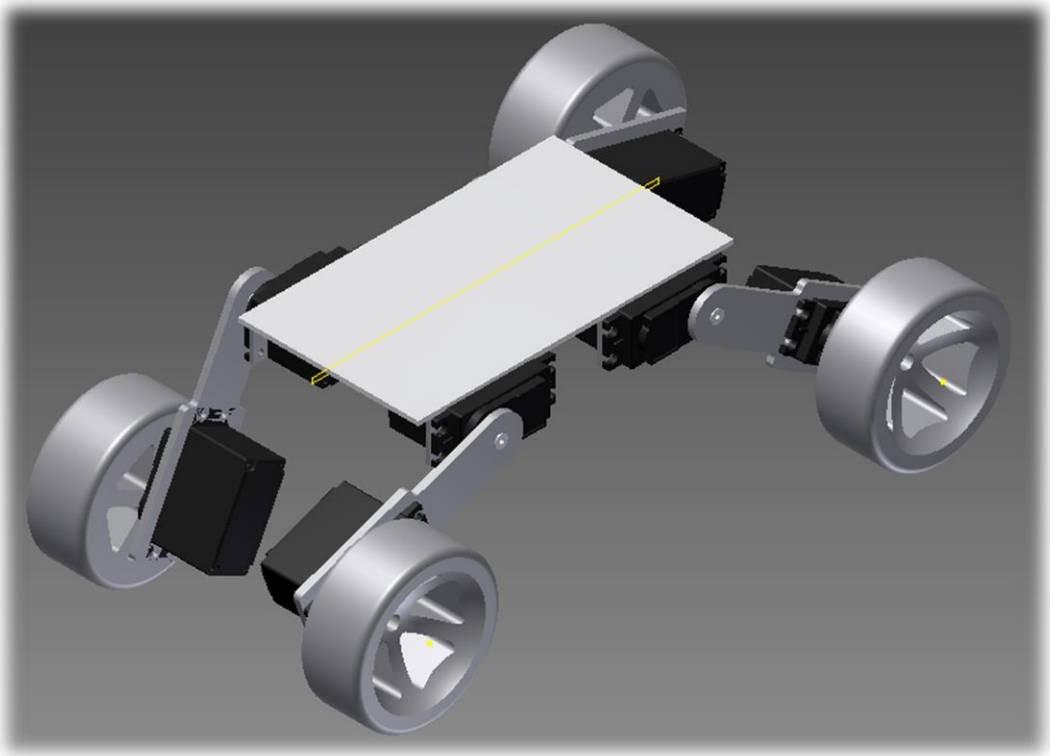
Tento projekt jsem si vybral z toho důvodu, že jsem žádný podobný, realizovaný v amatérských podmínkách, nikde nenašel a zdálo se mi, že výsledek by mohl být velice zajímavý.

2. Mechanická konstrukce

2.1. Podvozek pro vozidlo

Mechanické provedení podvozku je zhotoveno v domácích podmínkách, a použité díly jsou vybírány s ohledem na cenu, proto ne vše je naprosto přesné, avšak vše pasuje a na provoz vozidla tato skutečnost nemá pozorovatelný vliv.

Původní návrh vozidla je vidět na následujícím obrázku. Z důvodu různých nových poznatků, zjištěných v průběhu stavby se návrh měnil ad hoc, ale veškeré další úpravy byly realizovány pouze na papíře a proto nebudou součástí tohoto dokumentu. Základní rozmístění pohyblivých komponent zůstalo nezměněno.



Původní návrh - 3D model

Základní nosnou desku jsem zvolil ze 4 mm polykarbonátu a to kvůli jeho snadné obrobitelnosti a pro mne i snadné dostupnosti.

Spodní držáky serv jsou zhotoveny z hliníkových čtverhranných tyčí o rozměrech 20x6mm.

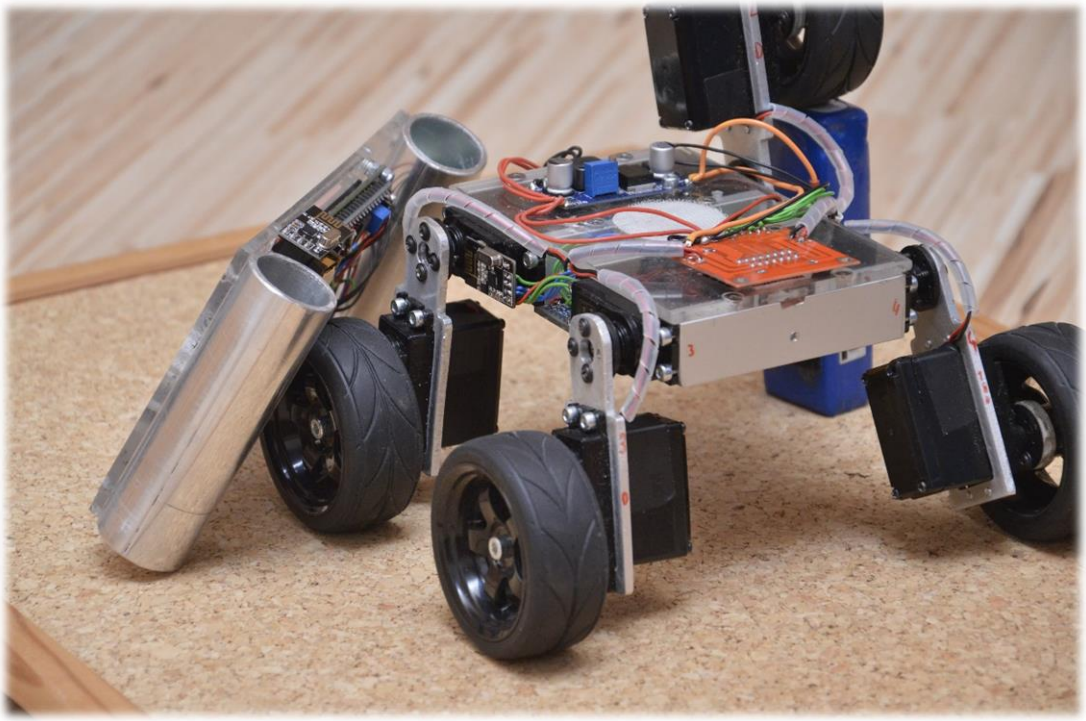
Spodní výztuha, která spojuje prostřední držáky serv je vyrobena z duralového plechu o síle 4 mm.

Ramena (držáky jednotlivých pohonných serv) jsou zhotovena z 3 mm duralového plechu.

Posledními věcmi, které bylo zapotřebí vyrobit, jsou šestihranné unašeče kol. Ty jsem zhotovil pomocí přetlakového lisování za tepla do formy. Jako materiál jsem zvolil mezi

modeláři velmi rozšířený PSH (houževnatý polystyren). Vše bylo prováděno v domácích podmínkách, proto povrch, ani celkový vzhled není dokonalý, avšak s trochou snahy nebyl problém dodržet správný tvar a soustřednost unašeče. Výsledný unašeč je nalepen vteřinovým lepidlem, za použití aktivátoru vteřinového lepidla, na kulatou páku serva, která je se servem standardně dodávána.

Vše je sešroubováno imbusovými šrouby velikosti M3, až na spoj ramen a páky serv, zde jsou použity šrouby velikosti M2 a u spoje kol s unašeči jsou použity šrouby M4.



Téměř finální vzhled

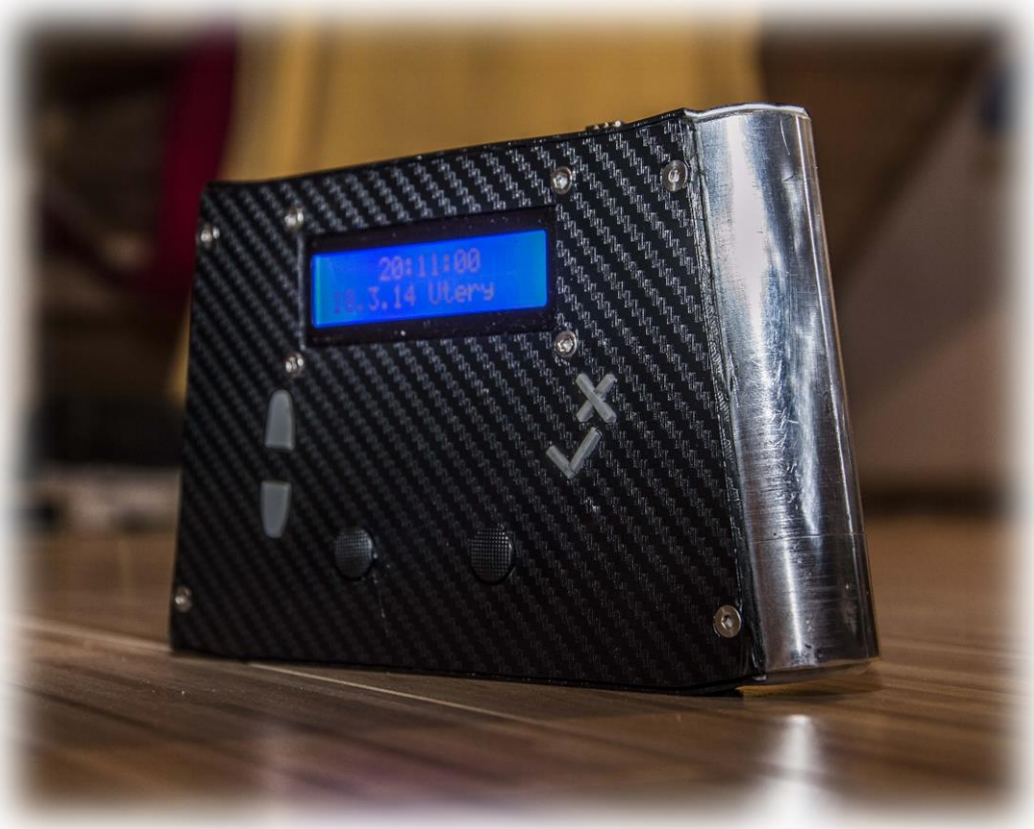
Unašeč kola s upraveným servem

2.2. Konstrukce vysílače

Základními prvky ovladače jsou:

- polykarbonátová deska – důvod výběru je zmíněn v návrhu podvozku
- hliníková trubka s průměrem cca 30mm – zvolil jsem ji z důvodu snadné dostupnosti a kvůli lesklému, až zrcadlovému povrchu
- kryt ovladače z PSH
- tlačítka z PSH
- fólie s karbonovým vzhledem, kterou je potažen téměř celý ovladač

Celá konstrukce je sešroubována šrouby velikosti M3, některé prvky jsou lepeny pomocí vteřinového lepidla a aktivátoru.



Vysílač

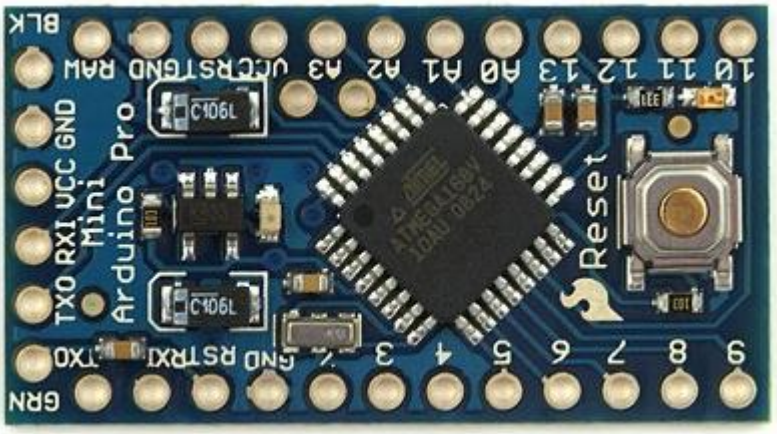
3. Výběr vhodných komponent

3.1. Výběr mikroprocesorů

Mikroprocesor je „mozkem“ jak vozidla, tak vysílače, proto je jeho výběr jednou z nejpodstatnějších voleb v celém projektu. Díky jednoduchosti a určitým zkušenostem jsem zvolil procesory od firmy Atmel řady ATmega. Prvotní výběr pro vozidlo padl na procesor ATmega2560, který disponuje několika desítkami vstupně – výstupních portů a 256kB vnitřní paměti. Po nějaké době se ale ukázalo, že tento procesor je zbytečně velký a ani nemám možnost využít všechny jeho funkce. Z výše uvedeného důvodu jsem dospěl k rozhodnutí projekt předělat a použít procesor ATmega328p, jenž disponuje až 23 vstupně-výstupními porty, z nichž je možné 8 portů využít pro měření analogového napětí. Dále tento procesor disponuje 32kB paměti, což se také ukázalo jako dostatečné místo pro můj program. Tento procesor jsem zvolil také proto, že spotřebuje malé množství proudu a jeho další výhodou jsou malé rozměry. Ukázal se též jako vhodný pro vysílač, a to především kvůli jeho malým rozměrům.

Specifikace procesoru:

- Flash paměť: 32kB
- Počet pinů: 32
- Maximální frekvence: 20MHz
- Typ CPU: 8bitový AVR
- Počet I/O pinů: 23
- Sběrnice: SPI, I^2C, UART
- Počet A/D kanálů: 8
- Rozlišení A/D převodníku: 10 bitů
- EEPROM paměť: 1kB
- Napájecí napětí: 1,8 – 5,5V
- Počet časovačů: 3
- Počet PWM výstupů: 6
- Interní oscilátor: ano – 8MHz



Arduino Pro Mini, zdroj: sparkfun.com

Mikroprocesor jsem zakoupil již osazený na desce s několika základními součástkami, které jsou potřeba pro jeho bezproblémový provoz.

Pro programování procesoru používám kvůli absenci sériového portu u svého PC převodník USB na UART.

V procesoru je pro zjednodušení programování nahrán program zvaný bootloader, který se pokouší při každém zapnutí spojit s počítačem přes sériovou sběrnici, a z té si do sebe nahrát program. Pokud sériová komunikace není dostupná, tak se proces přeskočí na standartní program, kterým je v mém případě řízení vozidla, respektive řízení vysílačky.

Procesor je programován v jazyce Wiring, což je jazyk, jenž je velmi podobný jazyku C. Programování v něm je ale oproti čistému C mnohem jednodušší.

Jazyk Wiring vznikl s projektem Arduino a je díky němu možné přenášet jeden program napříč několika (desítkami) procesory od firmy Atmel. Při kompilaci programu ve Wiringu dochází k přeložení programu pro konkrétní nastavený procesor a i když některé procesory nejsou oficiálně podporovány, stejně je velice jednoduché je naprogramovat ve Wiringu. Toto je možné hlavně kvůli tomu, že Arduino je open-source projekt.

Vývojové prostředí Arduino

3.2. Výběr komunikačních modulů

Ke komunikaci bylo možné použít několik systémů.

Mezi nejjednodušší použitelné patří komunikace za pomocí páru diod, z nichž jedna je vysílací, LED, druhá je přijímací, fotocitlivá. Tento systém má výhodu ve velmi nízké ceně, avšak má několik nevýhod. Mezi ně patří např. nízký dosah signálu, velká směrová závislost, citlivost na okolní světlo.

Jako další systém by bylo možné použít vysílač a přijímač na frekvenci 433, nebo 868 MHz. Dosah tohoto systému je oproti infračervenému přenosu také vyšší, avšak přenos je relativně pomalý a ne příliš spolehlivý.

Třetí možností bylo použití bluetooth modulů. Toto řešení je jedno ze dvou, ze kterých jsem nakonec vybíral. Bluetooth je spolehlivý systém, s dostatečnou přenosovou rychlostí a bylo by možné při napsání správné aplikace pro chytrý telefon ovládat vozidlo telefonem. Nevýhodou tohoto systému je však vyšší cena.

Předposlední možnost je Wi-Fi systém, který by měl pro tuto aplikaci téměř dokonalé vlastnosti, avšak cena tohoto systému je vysoká.

Poslední, nakonec zvolenou možností byl 2.4GHz přenos signálu s modulací GFSK, který je velice podobný systému bluetooth, ale přenosové moduly jsou cenově dostupnější. Použity byly moduly s integrovanými obvody nRF24L01+.

Hlavními výhodami těchto modulů jsou:

- využití bezlicenčního 2,4GHz pásma
- dostatečná přenosová rychlost
- dostatečný dosah signálu
- možnost obousměrné komunikace
- velmi nízká spotřeba

Obvod nRF24L01+ používá pro komunikaci s mikroprocesorem sběrnici SPI (Serial Peripheral Interface).

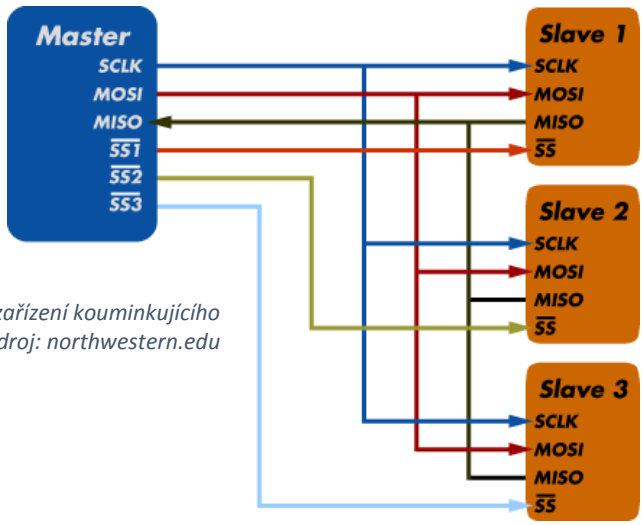


Schéma zapojení zařízení kouminkujícího pomocí SPI sběrnice, zdroj: northwestern.edu

3.3. Výběr senzorů vozidla

Ve vozidle jsou použity dva senzory, jeden pro snímání polohy, druhý např. pro sledování černé čáry na zemi, na využití tohoto senzoru se však nedostalo, integrace tohoto systému je plánována na pozdější dobu.

Senzorem polohy je obvod MPU-6050, který obsahuje 3osý gyroskop a 3osý akcelerometr. Ze dvou výše uvedených je využíván pouze akcelerometr, který umožňuje snímat polohu vůči zemskému povrchu. Obvod komunikuje s mikroprocesorem pomocí sběrnice I^2C (Inter-Integrated Circuit). Tato sběrnice, vyvinutá firmou Phillips má největší výhodu v tom, že je možné komunikovat pouze pomocí dvou signálových vodičů.

Specifikace obvodu MPU-6050

- Napájecí napětí: 2,4 – 3,4V
- Logické napětí: 1.71V – napájecí napětí
- Komunikační sběrnice I^2C
- Citlivost akcelerometru: až +- 2g
- Citlivost gyroskopu: až +-250°/sec
- Integrovaný 16 bitový A/D převodník
- Odolnost až do 10 000g

3.4. Výběr dalších obvodů vozidla

Pro vozidlo bylo zapotřebí snížit a stabilizovat napětí z baterií a to hned třikrát. Poprvé bylo zapotřebí snížit napětí baterií z cca 12V na cca 6V pro napájení modelářských serv. Podruhé bylo potřeba napájet mikroprocesor. Stabilizace na 5V je zajištěna stabilizátorem, který je na desce s mikroprocesorem. Poslední stabilizace byla na 3,3V pro napájení obvodu pro bezdrátovou komunikaci a pro napájení senzoru polohy.

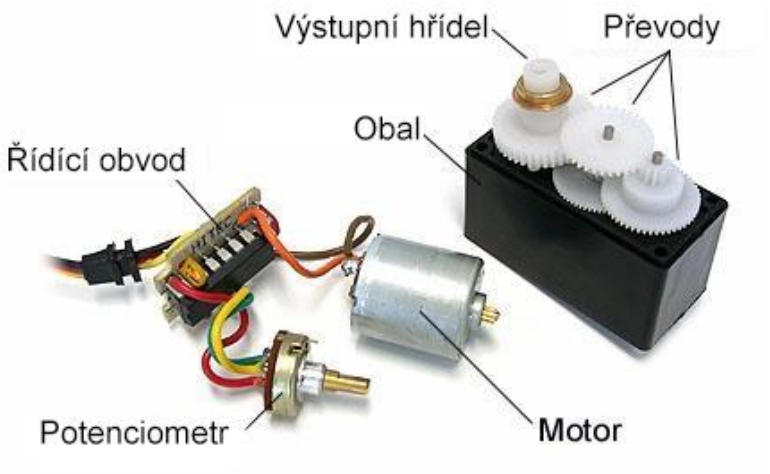
První stabilizace je kvůli účinnosti řešena pomocí spínaného měniče, který má účinnost kolem 90% (hodnota, kterou odhaduji z množství odevzdaného tepla stabilizátoru, nemám ji ale podloženou praktickým měřením). Při použití lineárního stabilizátoru by bylo nutné přebytečnou energii proměnit na teplo a využívala by se tedy pouze cca polovina energie, kterou dodá baterie.

Druhý stabilizátor, na 5V, je lineární, z důvodu odběru maximálně desítek mA.

Stejně tak třetí stabilizátor je lineární, zde je odběr ještě menší, řádově jednotky mA. Jako stabilizátor je použit integrovaný obvod LF33CV.

3.5. Výběr modelářských serv

Modelářská serva jsou vlastně malé stejnosměrné motory s převodovkou, zpětnou vazbou v podobě potenciometru, který je pevně spojený s výstupní hřídelí. Poslední částí serv je řídící elektronika, která vyhodnocuje příchozí číslicový signál a podle signálu nastavuje příslušný úhel výstupního hřídele.



Rozebrané servo, zdroj: gekogeek.com

Výběr vhodných serv

Pro můj projekt nebylo zapotřebí nikterak silných ani přesných serv, navíc jich bylo zapotřebí 8, proto jsem pořizoval nejsnáze dostupná serva v příslušné velikosti. Volba padla na serva Futaba S3003.

Specifikace serv:

- Typ ložisek: kluzná
- Materiál převodů: plast
- Rychlost otáčení při 6V: 0,19 sec/60°
- Síla při 6V: 4,1 kg/cm
- Délka: 40mm
- Šířka: 20mm
- Výška: 44mm

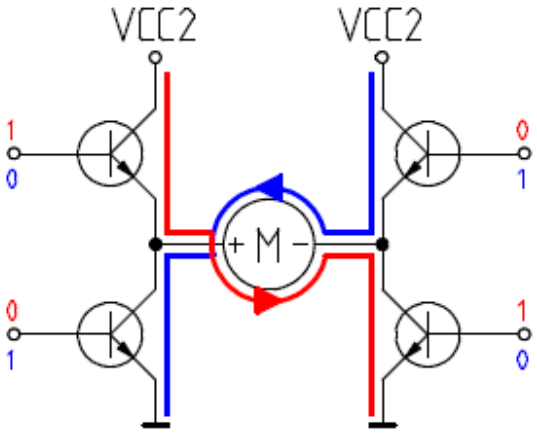
3.5.1. Úprava modelářských serv pro funkci pohonu

Aby bylo možné použít serva jako stále se otáčející pohony, bylo je nutné jak mechanicky, tak elektronicky upravit. Jako první bylo důležité rozebrat každé servo a na výstupním ozubeném kole upilovat mechanický doraz, aby se mohlo volně otáčet dokola. Další úprava spočívala ve vyjmutí potenciometru. Jednalo se pouze o odšroubování vrutu, který držel potenciometr v obalu serva a jeho následném vytáhnutí. Dále bylo nutné odstranit řídící elektroniku. Pro to bylo zapotřebí pouze odpájet kontakty na motoru, na které jsem dále připájel své, delší kabely, které vedou do H-můstku na těle vozítka. Nyní bylo možné používat servo pouze jako stejnosměrný motor s převodovkou.

3.6. Výběr H-můstků

H-můstky slouží v mém případě k obousměrnému řízení stejnosměrných motorů. H-můstek je možné představit si jako čtveřici spínačů, kde je jeden vypínač připojen mezi kladné napětí a jeden pól motoru, druhý vypínač je připojen mezi nulový potenciál a jeden pól motoru, třetí vypínač je připojen mezi kladné napětí a druhý pól motoru, čtvrtý vypínač je připojen mezi nulový potenciál a druhý pól motoru. Vhodným spínáním těchto spínačů je možné dosáhnout čtyř stavů stejnosměrného motoru:

1) Motor se pohybuje vpřed
2) Motor se pohybuje vzad
3) Motor brzdí
4) Motor se otáčí téměř bez odporu



Principielní schéma H-můstku, zdroj: eckhard-gosch.de

H-můstky je možné sestavit např.
z tranzistorů, ale jelikož se vyrábí integrované obvody, které obsahují v jednom pouzdře více, správně zapojených tranzistorů, tak je výhodnější zakoupit integrovaný obvod. Obvod, který používám, nese jméno L293 a obsahuje ve svém pouzdře hned 2 H-můstky. Obvod je osazen na desce s pomocnými součástkami (stabilizátor na 5V pro logiku obvodu, ochranné diody, které odepnou obvod v případě, že se na motoru vygeneruje vyšší napětí, apod.). Celý obvod je přišroubován na jeden z hliníkových držáků serv pro dobrý odvod tepla a pro větší chladící plochu. Obvod napájím přímo z baterie, pro vetší rychlost motorů. Obvod má 6 řídících vstupů:

- Vstupy A, B, C, D – slouží k ovládání párů spínačů (tzv. polovičních můstků)
- Vstupy ENABLE AB, CD – slouží k ovládání aktivity jednotlivých H-můstku, používají se pro možnost řízení rychlosti motorů pomocí modulace PWM.

3.7. Výběr displeje

Pro vysílačku bylo nutné použít displej pro jednodušší ovládání a snazší orientaci.

Displej musel splňovat následující požadavky:

- Dobrá čitelnost
- Podsvícení
- Ne příliš velké rozměry

Kvůli následujícím požadavkům jsem zvolil modře podsvícený displej s bílým písmem typu 1602.

Jak je patrné z názvu, tak displej má možnost zobrazit 32 znaků v 16 sloupcích a 2 řádcích. Každá buňka na displeji obsahuje 40 bodů v 5 sloupcích a 8 řádcích. Displej má kontrast nastavitelný ve velmi širokém pásmu a při stabilizovaném vstupním napětí kontrast nekolísá. Jas je také dostatečný a je možné ho regulovat např. pomocí PWM modulace z mikroprocesoru. Rozměry pasuje do vysílače, tudíž se mi zdál tento displej jako velice vhodný.

Dalšími možnostmi by byl displej s podobným vzhledem, ale s 20 sloupci a 4 řádky, ten byl ale již moc velký. Dále by bylo možné použít displej starého mobilního telefonu, nebo podobný, zde je ale problém s čitelností kvůli malým rozměrům.

3.8. Výběr mechanických senzorů ovladače

Základními ovládacími prvky vysílače jsou dva joysticky (někdy nazývané analogové páčky). Jako joysticky jsou použity ovladače z konzole PSP-1000, které se dají samostatně zakoupit jako náhradní díly. Každý joystick v sobě obsahuje dvě odporové dráhy a dva na sobě nezávislé jezdce. Je tedy možné na joystick napojit napájení a pak už jenom měřit jaké procento napájecího napětí vykazují jednotlivé dráhy. Joysticky nejsou úplně přesné, ani nikterak výrazně citlivé, ale vzhledem k rozměrům vysílače se jevili jako nejlepší řešení, všechny ostatní, běžně dostupné joysticky jsou i několikrát větší.

Dalšími mechanickými senzory jsou tlačítka. Tlačítek je na ovladači celkem 5:

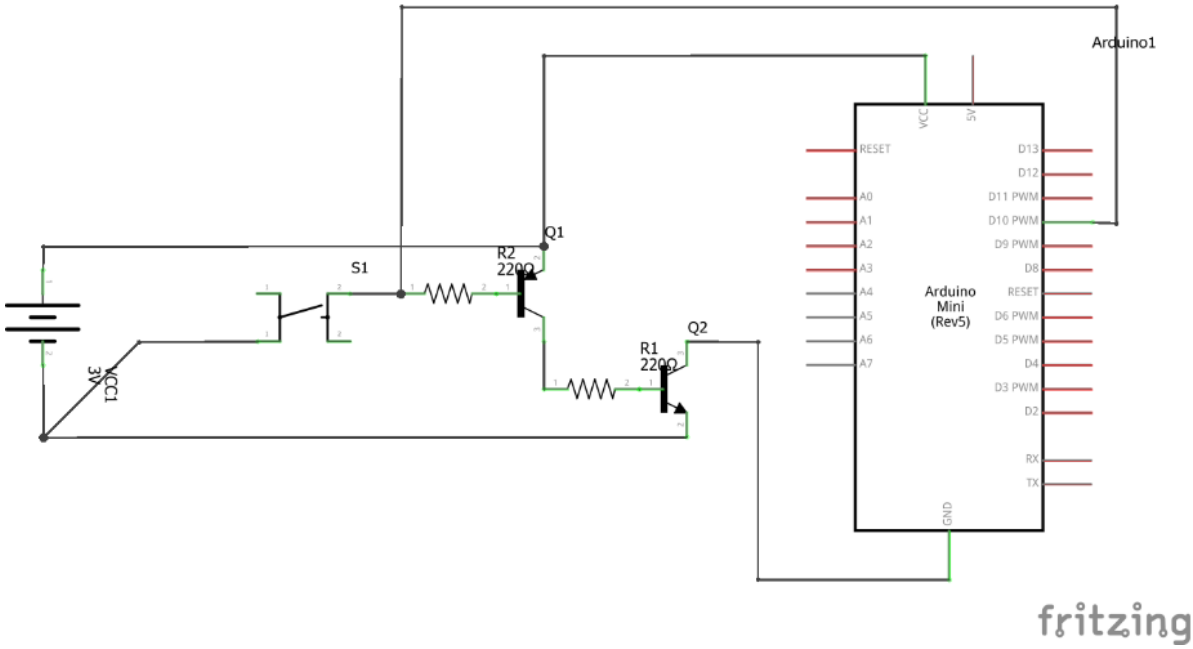
- Tlačítka pro pohyb nahoru a dolů
- Potvrzovací tlačítko
- Ukončovací tlačítko
- Zapínací tlačítko (skryto na spodní straně)

3.9. Výběr dalších obvodů vysílače

Pro vysílač bylo potřeba pořídit/vyrobit několik dalších obvodů:

1) Obvod pro řízení displeje pomocí I^2C sběrnice.
   Pro tuto aplikaci byl využit obvod PCF8574, který umožňuje pomocí I^2C sběrnice ovládat své výstupní piny. Obvod jsem zakoupil osazený na desce a připravený na přímou montáž připájením k desce displeje. Tento obvod bylo nutno využít z důvodu nedostatku pinů u procesoru pro přímé řízení displeje.

2) Zapínací obvod.
   Ten je vytvořen za pomoci dvou tranzistorů, jak je znázorněno na schématu.
   Po stisku tlačítka se uzemní první, PNP, tranzistor, který sepne a připojí kladné napájení na druhý, NPN, tranzistor, který také sepne a pustí napětí do obvodu mikroprocesoru. Mikroprocesor hned po zapnutí nastaví svůj pin 10 jako výstupní a nastaví se na nízkou hodnotu, čímž udrží sepnuté oba tranzistory. Pokud je třeba mikroprocesor vypnout, tak stačí programově nastavit pin 10 na vysokou hodnotu, čímž tranzistory rozepnou. Díky tomuto obvodu je odběr vysílače ve vypnutém stavu téměř nulový.



3) Stabilizátor napětí.
   Ten je potřeba pro stabilizaci napětí na 3,3V pro vysílací modul. Bylo nutné umístit na výstupní stranu kondenzátor s kapacitou jednotek uF, protože při vysílání vznikaly napěťové špičky, které nestíhal stabilizátor dorovnat a obvod se restartoval a vypadal jako nefunkční.

4) Obvod reálného času DS1307
   Slouží k zapamatování reálného času i po vypnutí vysílače, čas udržuje za pomocí své vlastní baterie.

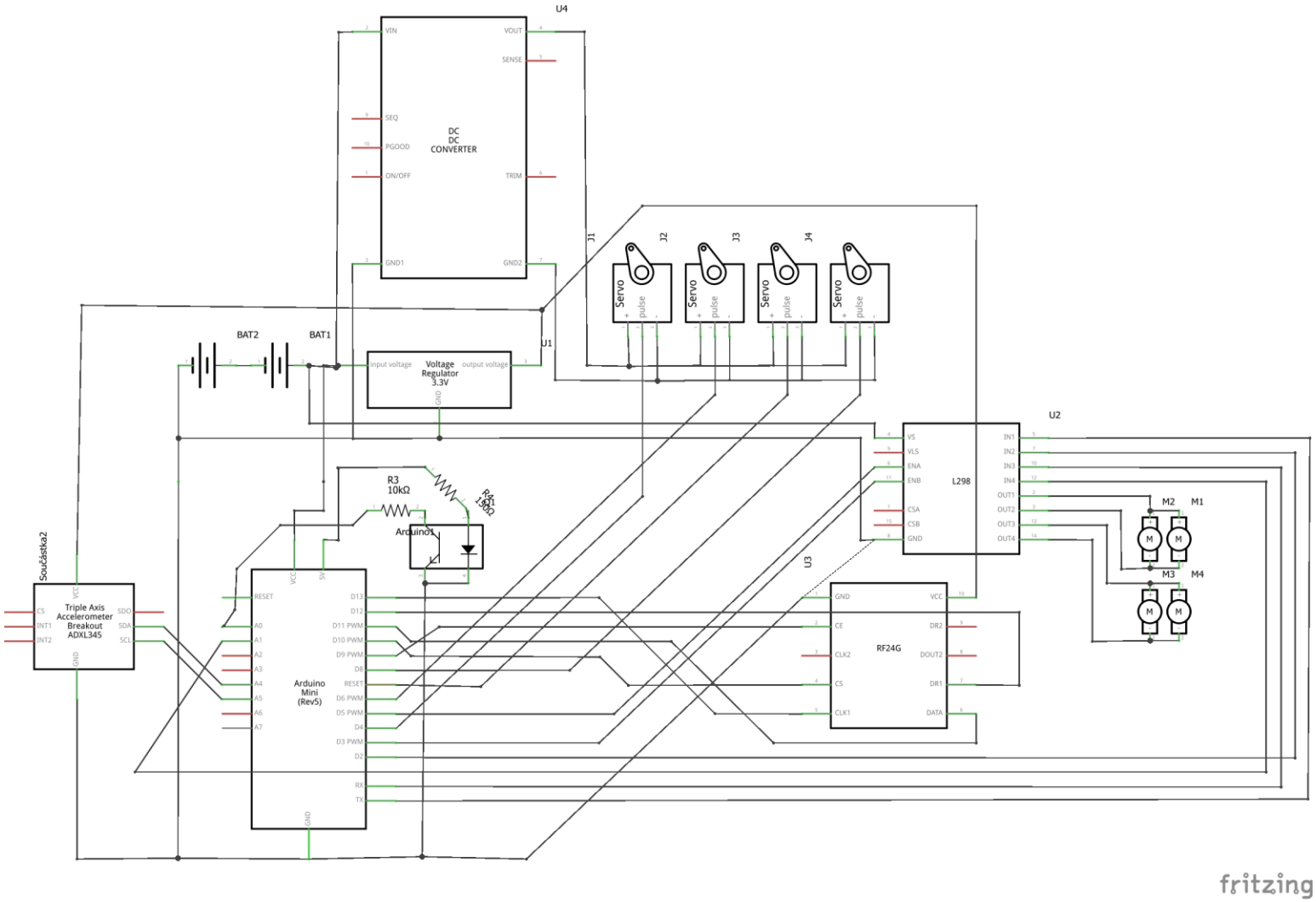
3.10. Výběr baterií

1) Výběr baterií pro vysílač
   U vysílače jsem od začátku počítal s využitím li-ion akumulátorů velikosti 18650, které se používají např. v bateriích notebooků. Ve vysílači jsou zapojeny dvě tyto baterie do série, tudíž výstupní napětí se pohybuje mezi 8,4-6V. Akumulátory jsou schovány uvnitř hliníkových rukojetí.

2) Výběr baterií pro vozidlo
   Původně jsem chtěl používat články 18650 i pro vozidlo, tento výběr se ale nakonec ukázal jako nevhodný, z důvodu velkého vnitřního odporu a nevhodných rozměrů.
   Pro vozidlo tedy používám 3 článkovou baterii li-pol, která má výrazně nižší vnitřní odpor a vyrábí se v hranatém provedení. Výstupní napětí se pohybuje mezi 12,6-9V.

4. Návrh zapojení, programu

4.1. Návrh zapojení jednotlivých komponent u vozidla



Základem celého zapojení je Arduino Pro mini s procesorem Atmega328p (na schématu vlevo dole).

Akcelerometr (na schématu vlevo) je připojen na piny A4, A5 a na stabilizátor napětí na 3,3V.

Na stabilizátor je dále připojen vysílací modul (na schématu vpravo dole), který je ještě připojen na piny 9-13 mikroprocesoru.

Na baterie je připojen spínaný měnič napětí (na schématu nahoře), jehož výstup je připojen ke všem servům.

Signálové vodiče serv jsou připojeny k pinům mikroprocesoru 4, 6, 7, 8.

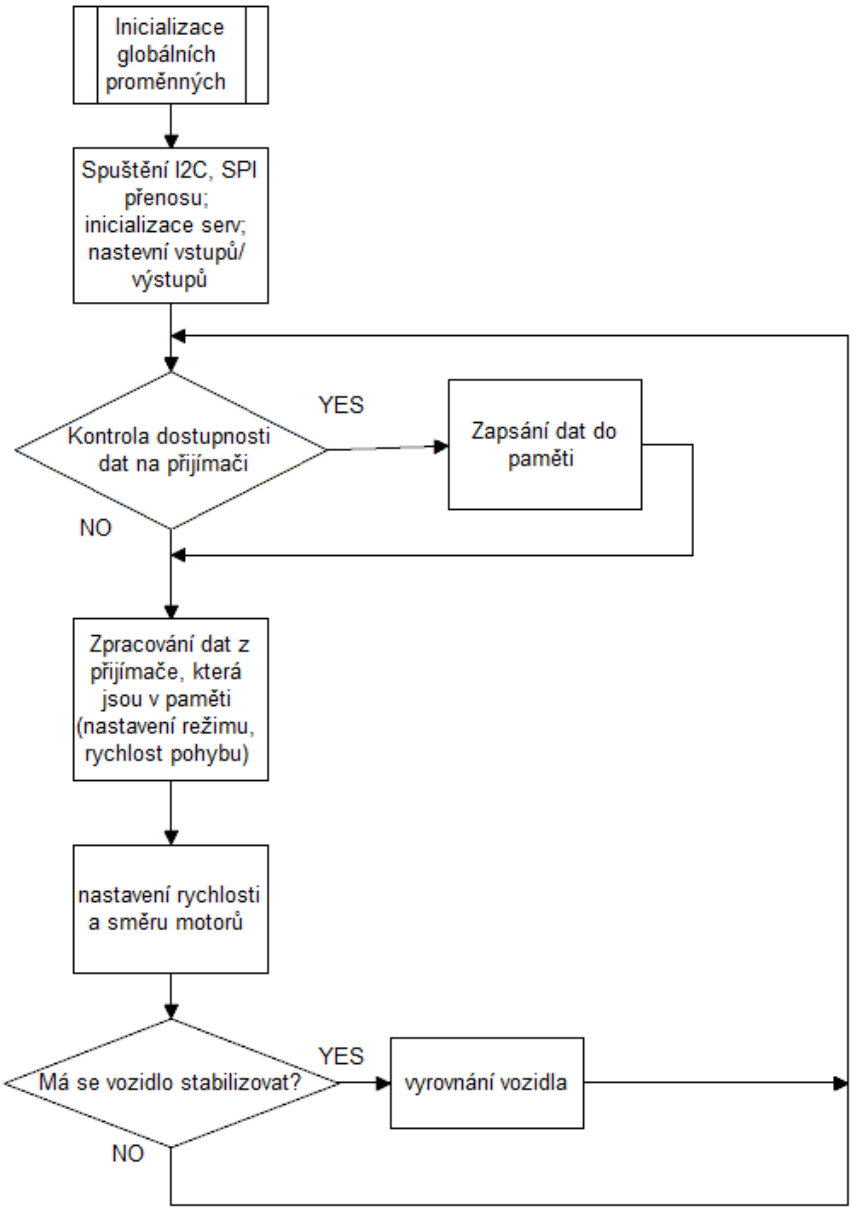
H – můstky (obvod L298, na schématu vpravo) jsou připojeny na piny A1, 0, 1, 2, 3, 5. Výstup H-můstku je připojen na paralelně spojené motory (každý H- můstek pohání jednu stranu vozidla). Napájení H-můstku je přímo z baterií.

Dále je na obrázku fototranzistor s diodou – tento senzor je implementovaný v přední části vozidla a je orientován dolů. Zatím nemá funkci, je zde kvůli dalšímu možnému rozšíření (např. pomalé sledování kontrastní čáry na zemi.

4.2. Návrh programu pro vozidlo

Celý program je pro jednoduchost rozdělen do dvou hlavních souborů. V jednom souboru je hlavní program, ve druhém souboru jsou funkce, které jsou volány z hlavního programu.

Základní princip programu pro vozidlo je patrný z diagramu níže.



4.2.1. Realizace programu pro vozidlo

Jako první jsou v programu uvedeny knihovny, které je zapotřebí zahrnout do programu.

Jedná se o tyto knihovny:

1) Wire.h, I2Cdev.h – knihovny sloužící pro komunikaci přes I2C sběrnici (akcelerometr)
2) Servo.h – knihovna pro ovládání serv pomocí vnitřních časovačů procesoru (ovládání běží „na pozadí“)
3) MPU6050.h – knihovna pro zjednodušení komunikace s akcelerometrem
4) SPI.h – knihovna sloužící pro komunikaci přes TWI sběrnici (vysílací modul)
5) Mirf.h, nRF24L01.h, MirfHardwareSpiDriver.h – knihovny pro jednoduchou komunikaci s vysílacím/přijímacím modulem

Dále jsou nastaveny veškeré proměnné, které jsou deklarované globálně – lze s nimi pracovat ve všech funkcí bez využití ukazatelů.

Zde uvádím nejzákladnější proměnné, jejich kompletní seznam je možné nalézt v programu v příloze.

`MPU6050 accelgyro;` – proměnná pro práci s akcelerometrem

`Servo LPS;` (a další 3) – slouží k označení serva, zkratka je složená z pozice ($L_{evé}P_{řední}S_{ervo}$)

`const int LPS_stred = 1370;` (a další 3) – udává středovou polohu serva v µs, kterou jsem zjistil pokusnou metodou. Jelikož se jedná o datový typ const int, tak se nejedná o klasickou proměnnou. Jedná se o tzv. statickou proměnnou, která je při kompilaci nahrazena konkrétním číslem.

Následuje velké množství pomocných proměnných, které slouží pro dočasná data v programu. Vypisovat zde všechny proměnné by nemělo smysl, většina proměnných by měla být pochopitelná z názvu proměnné.

Funkce **`setup`**`()` – tato funkce proběhne pouze jednou (po startu procesoru) a slouží k nastavení různých hodnot, které většinou zůstávají po dobu běhu programu neměnné.
Moje funkce setup() obsahuje jediný příkaz a to `initializovat();` - jedná se o podprogram, který je obsažen v souboru funkce.ino.

Zde je obsah funkce initializovat() s popisem činnosti:

```
Wire.begin();                          //započetí I2C komunikace pro
                                       komunikaci s akcelerometrem

 accelgyro.initialize();               //inicializace akcelerometru

 Mirf.spi = &MirfHardwareSpi;
 Mirf.cePin = 9;                       //nastavení ce a csn pinu modulu
 Mirf.csnPin = 10;
```

```
Mirf.init();                          //inicializace modulu
Mirf.setRADDR((byte *)"vysilac");     //nastavení adresy vysílače
Mirf.setTADDR((byte *)"robot");       //nastavení adresy přijímače
Mirf.payload = sizeof(unsigned long);      //nastavení datového typu
                                           přenášené proměnné
Mirf.channel = 22;                         //nastavení kanálu pro
                                           komunikaci
Mirf.config();                             //zapsání nastavených hodnot
                                           do bezdrátového modulu


pinMode(A0, INPUT);                         //nastavení vstupů/výstupu
pinMode(A1, OUTPUT);                        //A před číslem značí, že se
                                            jedná o analogový pin
pinMode(0, OUTPUT);
pinMode(1, OUTPUT);
pinMode(2, OUTPUT);
pinMode(3, OUTPUT);
pinMode(5, OUTPUT);

LPS.attach(6, 570, 2350);                   //inicializace serv,
PPS.attach(4, 630, 2650);                   //nastavení pinů, krajních
LZS.attach(7, 570, 2450);                   //poloh
PZS.attach(8, 620, 2550);
```

Po funkci `setup()` následuje funkce `loop()`, která se opakuje stále dokola, do vypnutí procesoru.

Funkce loop se skládá z několika základních částí:

1) Kontrola dostupnosti dat na přijímači a případný příjem a dat a jejich zapsání do paměti
2) Zpracování dat z přijímače a jejich rozdělení do několika proměnných
3) Zjištění aktuálního jízdního režimu a podle toho nastavení rychlostí motorů, poloh serv

Ad 1)

Kontrola dat probíhá pomocí příkazu `Mirf.dataReady()` který vrací hodnotu TRUE, nebo FALSE, podle toho, zda jsou data dostupná. Dále se ještě kontroluje, zda modul neodesílá data (příprava pro obousměrnou komunikaci robota s vysílačkou) pomocí příkazu `Mirf.isSending()`. Pokud jsou data dostupná a modul neodesílá data, tak je proveden zápis přijímaných dat do paměti pomocí příkazu `Mirf.getData(&unsigned long)`.

Celý příkaz poté vypadá takto:

```
if (!Mirf.isSending() && Mirf.dataReady()) {     //kontrola dostupnosti
                                                   dat na sběrnici
  Mirf.getData((byte *)&data);                   //získání dat z modulu
}
```

Ad 2)

Zpracování dat probíhá následujícím způsobem:

Přijímaná proměnná je typu unsigned long (32bitová kladná proměnná), který může nabývat hodnot od 0 do 4 294 967 295.

První dvě číslice (zleva) označují jízdní režim.

U jízdního režimu 01 (stabilizace) znamenají další číslice následující věci:

3. číslice udává, které osy se mají stabilizovat

    0 – stabilizace jak osy x, tak osy y

    1 – stabilizace osy x (předo - zadní náklon)

    2 – stabilizace osy y (boční náklon)

    3 – bez stabilizace – vozidlo se přepne do základní polohy serv

4. číslice udává posun v horizontálním směru

    0 – bez posunu

    1 – vozidlo se zvedne

    2 – vozidlo se sníží

5. – 7. číslice udává polohu levého joysticku na ovladači ve vertikálním směru (zatáčení). Číslo nabývá hodnot 45-555, kde 300 je střed.

8. – 10. číslice udává polohu pravého joysticku na ovladači v horizontálním směru (rychlost). Číslo nabývá hodnot 45-555, kde 300 je střed.

Jednotlivá čísla jsou z proměnné získávána pomocí funkce / (děleno) a pomocí funkce % (modulo – zbytek)

Prozatím vozidlo disponuje tímto jedním jízdním režimem, avšak v plánu je několik dalších (např. ježdění pod dvou kolech ve stylu vozidla Segway, testování senzorů, testování serv, sledování černé čáry na zemi apod.)

Ad 3)

Zjištění jízdního režimu je řešenou podmínkami (např. `if ( mode == 1)`).

Jízdní režim 01: nastavení serv je řešeno voláním funkce `vyrovnani()`, rychlost motorů je vypočítávána následujícím způsobem:

Od polohy pravého i levého joysticku je odečteno 300, pro dostání čísla 0 v nulové poloze.

Jako první je identifikován směr jízdy, zjištěním, zda je poloha pravého joysticku větší, nebo menší, než nula. Pokud je větší, pak je směr jízdy vpřed a naopak. Směr jízdy je zapsán do proměnné

(1 – dopředu, -1 – dozadu). Rychlost levých motorů je vypočítávána jako: poloha pravého (rychlostního) joysticku + poloha levého (zatáčecího) joysticku * směr jízdy, rychlost pravých motorů je vypočítávána jako: poloha pravého (rychlostního) joysticku - poloha levého (zatáčecího) joysticku * směr jízdy.

Přičítání (odečítání) polohy levého joysticku je zde z důvodu změny rychlosti kol na pravé a levé straně pro dosažení zatáčení rozdílnou rychlostí kol. Násobení směrem jízdy je zde proto, aby se vozidlo při zatáčení dopředu i dozadu chovalo, jako by mělo zatáčecí kola (při jízdě dopředu zatáčí na druhou stranu, než při jízdě dozadu).

Popis funkce vyrovnani():

Funkce vyrovnání se skládá z těchto částí:

1) Zjištění náklonu pomocí funkce getDeg() (popsána níže)
2) Kontrola, zda vozidlo není převrácené, případně, reakce na převrácení
3) Výpočet velikosti úhlu, o který se mají serva posunout
4) Přičtení/odečtení správného úhlu ke konkrétním servům, podle toho, kam je vozidlo nakloněné. Pokud je vozidlo v režimu Nestabilizovat, tak je tento krok přeskočen
5) Přičtení horizontálního posunutí k pozicím serv
6) Kontrola, zda nebyly překročeny maximální úhly serv, případná korekce
7) Zápis úhlu na serva

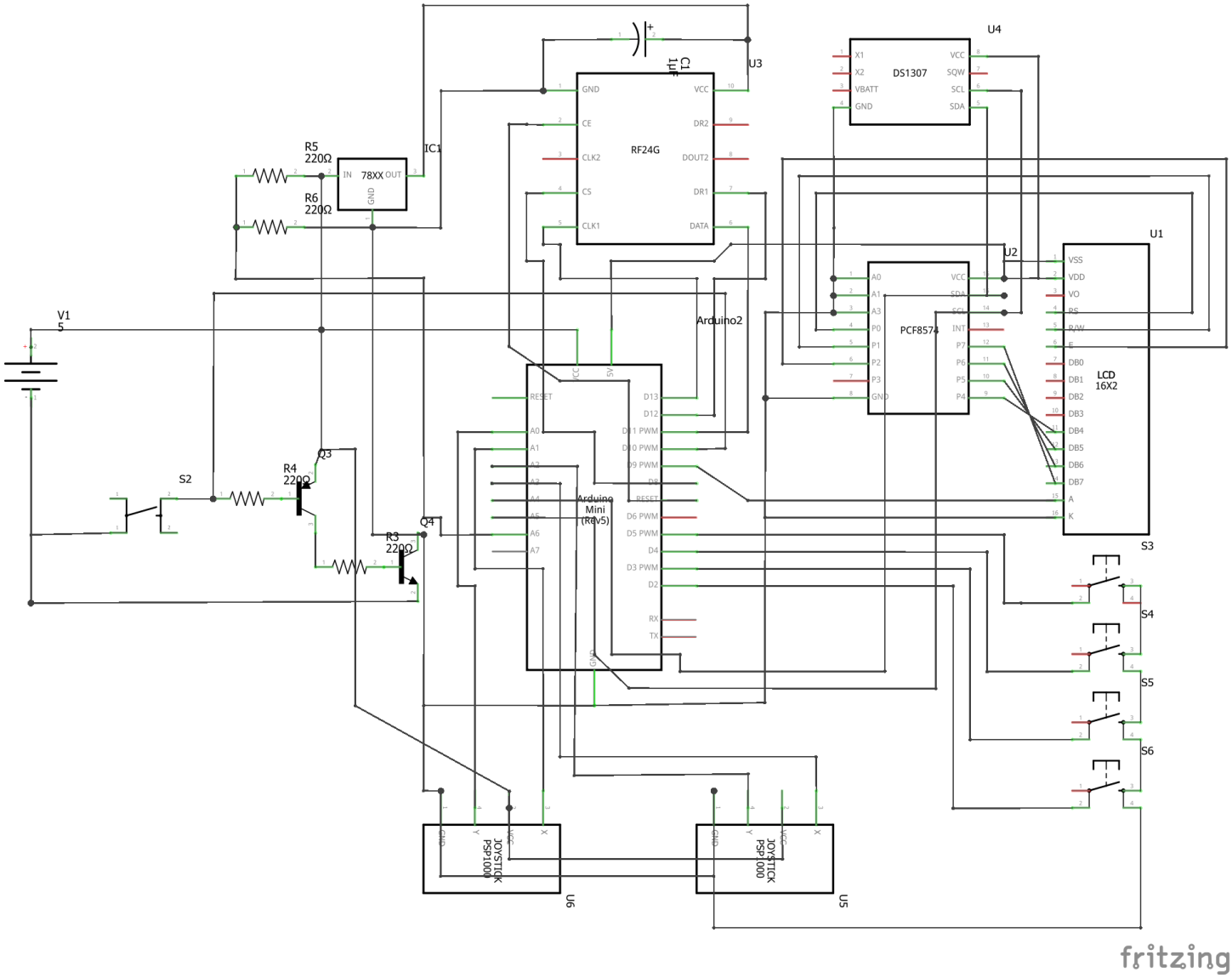
Celý tento proces je okomentován přímo v programu, který je přiložen.

Popis funkce getDeg():

Tato funkce se skládá z následujících kroků:

1) Zjištění surových dat z akcelerometru pomocí funkce `accelgyro.getMotion6()`
2) Průchod zjištěných hodnot přes číslicovou dolní propust kvůli odstranění rušení, které přichází především od mechanického pohybu serv a motorů.

4.3. Návrh zapojení jednotlivých komponent u vysílače



Řídící jednotkou je stejně jako u vozidla Arduino Pro Mini.

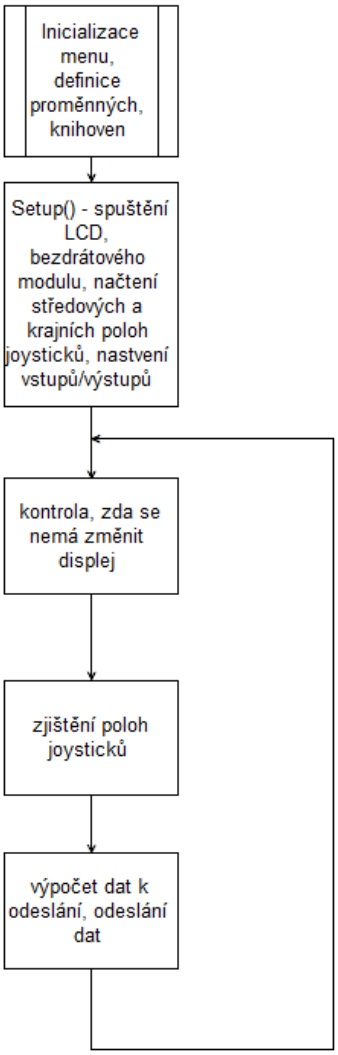
Dále vysílač obsahuje následující součásti a moduly:

- 2x joystick z konzole PSP
- 4x ovládací tlačítko
- Zapínací obvod
- Stabilizátor napětí na 3,3V
- Dělič vstupního napětí na polovinu pro možnost změření

- Vysílací/přijímací modul
- Obvod reálného času (DS1307)
- LCD displej
- Obvod pro řízení displeje pomocí I2C sběrnice

4.4. Návrh programu pro vysílač

Koncepce programu je stejná jako u vozidla, základní funkce jsou v hlavním souboru, pomocné funkce jsou umístěny ve druhém souboru. Na obrázku níže je základní princip programu.



4.4.1. Realizace programu pro vysílač

Syntaxe programu je zcela stejná, jako u programu pro vozidlo, proto zde uvedu pouze kratší popis programu.

Jako první jsou znovu uvedeny zahrnuté knihovny, zde se jedná o tyto knihovny:

1) Wire.h – pro komunikaci přes I2C sběrnici
2) LiquidCrystal_I2C.h – pro řízení LCD displeje
3) SPI.h, Mirf.h, nRF24L01.h, MirfHardwareSpiDriver.h – pro řízení vysílacího/přijímacího modulu
4) EEPROM.h – slouží k přístupu k EEPROM paměti mikroprocesoru, v mém programu konkrétně pro zapamatování středových, krajních poloh joysticků a úrovně podsvícení
5) LCDMenuLib.h – slouží k vytvoření grafického menu na LCD displeji
6) DS1307new.h – slouží k ovládání obvodu reálného času

Jako další jsou zde opět nadeklarované globální proměnné a dále jsou zde nadefinované položky menu, např.:

```
LCDMenuLib_element(0, Item, Item_1, "Jizdni rezimy", FUNC);
```

V programu následuje funkce setup(), která obsahuje následující věci:

- Nastavení pinu podsvícení displeje na uloženou hodnotu
- Nastavení vstupních/výstupních pinů
- Konfigurace bezdrátového modulu
- Načtení do paměti středových a krajních poloh joysticků
- Inicializace grafického menu
- Nastavení pinu 10 na nízkou úroveň a tím udržení sepnutých tranzistorů v zapínacím obvodu
- Smazání displeje

Následuje program loop():

Jako první dvojice příkazu je:

```
LCDMenuLib_control_digital();
LCDMenuLib_loop();
```

Která slouží k ovládání grafického menu.

Dále následuje přečtení polohy joysticků, zjištění jízdního režimu, výpočet dat pro odeslání a odeslání dat.

Další funkce ovladače, které je možno vyvolat z menu:

1) Funkce test() (test tlačítek)

Funkce pro otestování všech ovládacích tlačítek a joysticků.

2) Funkce kalibrace() (nastavení krajních a středových poloh joysticků)

Funkce vytiskne na displej instrukce pro kalibraci a poté i nastavené hodnoty. Na konci funkce se hodnoty zapíší do paměti EEPROM

3) Funkce jas() (nastavení a uložení jasu displeje)

Funkce tiskne na displej aktuální hodnotu podsvícení v procentech, je možné jí ovládat za pomoci šipek nahoru, dolů. Pomocí potvrzovacího tlačítka se uloží nastavená hodnota, pomocí stornovacího tlačítka se načte původní hodnota

4) Vypnout()

Funkce nastaví pin 10 na logickou 1, čímž rozepne PNP a následně i NPN tranzistor. Tím dojde k odpojení procesoru od napájení.

Před vypnutím dojde k postupnému snižování jasu displeje a tisku textu „Na shledanou“

5) Funkce zobrazit_cas()

Funkce vytiskne na displej čas, který přečte z obvodu reálného času. Zároveň upraví čas do takového formátu, aby vždy byl stejně dlouhý (před 1ciferné číslo přidá nulu). Dále vytiskne den v týdnu.

V této funkci je též zakomponován ukazatel napětí baterie, který ukazuje v 6 stupních, od plně nabité, po plně vybitou

Funkce se automaticky zobrazí po 30 sekundách nečinnosti.

6) Funkce ukazat_napeti()

Funkce přečte na analogovém pinu napětí vydělené analogově dvěma, opět jej vynásobí dvěma a vytiskne na displeji.

7) Funkce nastavenicasu()

Funkce pro nastavení času. Stále zobrazuje na displeji čas, který je možno nastavit šipkami nahoru a dolu, po potvrzení je čas zapsán do obvodu reálného času a je spuštěn oscilátor v tomto obvodu.

5. Problémy při návrhu a jejich řešení

Při výrobě a programování jak vozidla, tak vysílače nastalo několik komplikací. Zde popíšu ty nejhlavnější:

- Špatně zvolená mechanická konstrukce vozidla
    - Původní návrh vozidla počítal pouze s 2mm polykarbonátovou vrchní deskou a s žádnou spodní deskou. Tento návrh ale nevydržel moc dlouho, při prvním pádu se deska rozlomila, z toho důvodu jsem se rozhodl použít pro vrchní desku 4mm polykarbonát a vyztužit celé vozidlo zespodu 4mm duralovou deskou, kam jsem nakonec umístil velkou část elektroniky.

- Špatně zvolené H-můstky pro vozidlo
    - Jako první jsem počítal s využitím čtyř obvodů LG9110S, avšak jak se po krátké době ukázalo, tak tento obvod není dostatečně výkonný a tyto obvody začaly brzo odcházet. Proto jsem zvolil obvod L298, který má výrazně vyšší maximální povolený proud.

- Nevhodná baterie pro vozidlo
    - Vozidlo jsem chtěl napájet, stejně jako vysílač, pomocí dvou článků 18650. Vozidlo s těmito bateriemi bylo plně provozuschopné, ale bylo velice pomalé a kola neměla velkou sílu. Proto jsem se rozhodl použít tříčlánkovou baterii li-pol.

- Problém s komunikací bezdrátových modulů
    - Několik týdnů jsem měl problém zprovoznit bezdrátové moduly, nějakou dobu jsem si dokonce myslel, že jsem moduly neopatrným zacházením zničil. Avšak ani nové moduly nebyly funkční, tudíž jsem začal hledat příčinu jinde. Příčina byla ve stabilizátoru napětí na 3,3V, který nestíhal dostatečně rychle srovnávat výstupní napětí při odesílání dat (odběrová špička obvodu). Tento problém jsem vyřešil připojením kondenzátoru na výstup stabilizátoru.

- Nevhodný materiál pro ovladač
    - Při výrobě ovladače jsem prvotně využil 2mm desku z překližky. To se neukázalo jako nejlepší nápad, jelikož deska byla velmi ohebná a rychle se špinila. Desku jsem tedy nahradil 4mm polykarbonátem, který jsem ještě kvůli vzhledu potáhl samolepící fólií s vzhledem uhlíkových vláken.

6. Závěr

Během vytváření práce jsem prozkoumal spoustu možností dnešních elektronických systémů, různých integrovaných obvodů a mikroprocesorů. Naučil jsem se pracovat s novými prvky ovládanými mikroprocesorem (např. LCD displej, bezdrátový modul, senzor polohy, atd.).

Má práce pravděpodobně nebude mít žádný reálný přínos v oblasti automaticky řízených systémů, avšak mohla by mít praktické využití jako výuková pomůcka (celé vozidlo lze sestavit pod 2000Kč)., především díky rozmanitosti řešených problémů (mechanická konstrukce, elektronická konstrukce, bezdrátový přenos, návrh a napsání softwaru).

Vozidlo plánuji dále vylepšovat a rozšiřovat o další režimy, na které je místo jak v procesoru vozidla, tak v procesoru vysílače.

Sestrojené vozidlo a tato práce odpovídá všem bodům zadání, navíc obsahuje konstrukci vysílače, se kterým jsem původně nepočítal (plánoval jsem využít běžně prodávaný vysílač pro RC modely)

7. Použité materiály a zdroje

- Oficiální stránky projektu Arduino - www.arduino.cc, podstránky playground.arduino.cc a forum.arduino.cc

- Stránka uživatele maniacbug, který vytvořil knihovnu pro ovládání bezdrátových modulů nRF24L01 - https://github.com/maniacbug

- Stránka uživatele Jomelo, který vytvořil knihovnu pro menu na LCD displeji - https://github.com/Jomelo

- Stránka Olivera Krause, který vytvořil knihovnu pro práci s obvodem reálného času - http://code.google.com/p/ds1307new/

- Stránka uživatele jrowberg, který vytvořil knihovnu pro komunikaci s akcelerometrem MPU6050 - https://github.com/jrowberg

8. Přílohy

Jako první bych zde rád uvedl program, který operuje s robotem. Nejdříve uvedu jeho hlavní část, poté soubor s funkcemi.

```
#include <Wire.h>
#include <Servo.h>
#include <I2Cdev.h>
#include <MPU6050.h>
#include <SPI.h>
#include <Mirf.h>
#include <nRF24L01.h>
#include <MirfHardwareSpiDriver.h>
//#include <EEPROM.h>


MPU6050 accelgyro;                      //vytvoření proměnných, do
int16_t ax, ay, az;                     //kterých se budou zapisovat
int16_t gx, gy, gz;                     //data z akcelerometru (16-bit
                                                integer)




Servo LPS;                              //výstupy
Servo PPS;
Servo LZS;
Servo PZS;

const int LPS_stred = 1370;
const int PPS_stred = 1550;
const int LZS_stred = 1450;
const int PZS_stred = 1500;


int krok, posunuti;
int krok_k = 135;
int LPPp, PPPp, LZPp, PZPp;
int citac_prevraceni, citac, citac_stabilizace, az_pred = 1, zdrzeni;
long predchozidoba;
float x, y, z;
double LPSp = LPS_stred - 300 ;
double PPSp = PPS_stred + 300;
double LZSp = LZS_stred + 300;
double PZSp = PZS_stred - 300;
const float filtr = 0.5;
unsigned long data = 100300300, mode = 1, stabilizace = 0;
int counter = 0;
int smer1, smer2;
int poloha_joylx, poloha_joyly, poloha_joypx, poloha_joypy;


void setup() {
  initializovat();
}



void loop() {
```

```
 if (!Mirf.isSending() && Mirf.dataReady()) {     //kontrola dostupnosti
                                                       dat na sběrnici
    Mirf.getData((byte *)&data);                  //získání dat z modulu
    mode = (data / 100000000);
  }
  if ( mode == 1) {
    stabilizace = ((data / 10000000) % 10);      //výpočet módu stabilizace
    vyrovnani();
    poloha_joypy = (data % 1000) - 300;         //zjištění polohy joysticků
                                                       z příchozích dat
    poloha_joylx = ((data % 1000000) / 1000) - 300;

    switch ((data / 1000000) % 10) {           //zjišťování, zda se má robot
                                               zvednout/snížit/nedělat nic
      case 1:
        posunuti = 5;
        break;
      case 2:
        posunuti = -5;
        break;
      default:
        posunuti = 0;
    }
    smer2 = (poloha_joypy < 0) ? 1 : -1;     //zjištění směru vozidla, pro
                                               možnost správně určit, jak má
                                               vozidlo zatáčet
    motor(0, poloha_joypy + poloha_joylx * smer2);  //volání funkcí pro
                                                       motory s vypočítáním
                                                        správné rychlosti
    motor(1, poloha_joypy - poloha_joylx * smer2);
  }
  if (mode == 2) {
    motor(0, 0);
    motor(1, 0);
  }
  if (mode == 3) {
    motor(0, 0);
    motor(1, 0);
    zdrzeni = data % 10000;
  }
}
```

A nyní soubor s funkcemi:

```
void vyrovnani() {        //funkce pro vyrovnání robota
  getDeg();               //volání funkce pro zjištění náklonu
  if (az < -50){          //kontrola, zda je vozidlo převráceno, pokud ano,
                        převrácení hodnot náklonu, pro stále funkční
                                                          stabilizaci
    x=-x;
    y=-y;
  }
  if ((az < -50)&&az_pred==1) {     //kontrola převracení a kontrola, zda
                                      už tato funkce nebylo po převrácení
                                      provedena


 posunuti=-1000;              //pokud je vozidlo převráceno, tak se
```

```
                                odečte od serv 1000 pro zvednutí
                                serv, aby bylo možné pokračovat v jízdě
 az_pred=-1;                  //nastavení na -1, aby se už znova neprováděla
                                horní instrukce
 }
  if ((az > 50)&&az_pred==-1) { //stejné, jako /\ /\ /\ ale pro převrácení
                                                                    zpět
 posunuti=1000;
 az_pred=1;
  }

  if (((abs(x) > abs(y)) && stabilizace == 0) || stabilizace == 1)
//výpočet, o kolik se má servo posunout, hodnota, jakou dělit vstupní data
byla zjištěna experimentálně, díky rychlému běhu programu stačí posouvat o
relativně malé číslo
    krok = abs(x / 1000);    //výpočet posunutí se provádí z osy, která je
                                                          více nakloněná
  if ((((abs(x) <= abs(y)) && stabilizace == 0) || stabilizace == 2))
    krok = abs(y / 1000);
  krok = min(krok, 50);       //zajištění, že krok nebude větší, než 50us
  if (stabilizace != 3) {
    citac_stabilizace = 0;
    if ((abs(x) > abs(y))&&(stabilizace==0||stabilizace==1)) {
//výběr osy, která se bude stabilizovat (té s větší odchylkou)

      if (x > 0) { //provádění srovnání v závislosti na kterou stranu je
                                                        vozidlo nakloněno
        LPSp -= krok;
        PPSp += krok;
        LZSp -= krok;
        PZSp += krok;
      }
      else if (x < 0) {
        LPSp += krok;
        PPSp -= krok;
        LZSp += krok;
        PZSp -= krok;
      }
    }
    if ((abs(x) < abs(y)&&(stabilizace==0||stabilizace==2))) {
      if (y > 0) {
        LPSp -= krok;
        PPSp -= krok;
        LZSp += krok;
        PZSp += krok;
      }
      else if (y < 0) {
        LPSp += krok;
        PPSp += krok;
        LZSp -= krok;
        PZSp -= krok;
      }
    }
  }
  else if (citac_stabilizace == 0) {  //pokud je v modu nestabilizovat, tak
                                poprvé nastaví serva do středové polohy
    LPSp=LPS_stred;
    PPSp=PPS_stred;
    LZSp=LZS_stred;
    PZSp=PZS_stred;
    citac_stabilizace++;
```

```
  }
  LPSp -= posunuti;             //vypocet o kolik se mají serva posunout
  PPSp += posunuti;
  LZSp += posunuti;
  PZSp -= posunuti;

  LPSp = constrain(LPSp, 570, 2350);  //omezeni hodnot posílaných do serv
  PPSp = constrain(PPSp, 630, 2650);
  LZSp = constrain(LZSp, 570, 2450);
  PZSp = constrain(PZSp, 620, 2550);
  LPS.writeMicroseconds(LPSp);     //odeslání hodnot do serv
  PPS.writeMicroseconds(PPSp);
  LZS.writeMicroseconds(LZSp);
  PZS.writeMicroseconds(PZSp);
}



void getDeg() {               //funkce pro zjištění náklonu
  accelgyro.getMotion6(&ax, &ay, &az, &gx, &gy, &gz);   //přečtení
                                    výstupních dat z akcelerometru
  x = (ax * (1 - filtr)) + (x * filtr);        //softwarové dolní propusti,
  y = (ay * (1 - filtr)) + (y * filtr);       //hodnotou filtr je nastavena
  z = (az * (1 - filtr)) + (z * filtr);       //časová konstanta

void initializovat() {
  Wire.begin();    //započetí I2C komunikace pro komunikaci s
akcelerometrem
  accelgyro.initialize();                     //inicializace akcelerometru

  Mirf.spi = &MirfHardwareSpi;          //inicializace bezdrátového modulu
  Mirf.cePin = 9;
  Mirf.csnPin = 10;
  Mirf.init();
  Mirf.setRADDR((byte *)"vysilac");
  Mirf.setTADDR((byte *)"robot");
  Mirf.payload = sizeof(unsigned long);
  Mirf.channel = 22;
  Mirf.config();

  pinMode(A0, INPUT);                      //nastavení vstupů/výstupu
  pinMode(A1, OUTPUT);
  pinMode(0, OUTPUT);
  pinMode(1, OUTPUT);
  pinMode(2, OUTPUT);
  pinMode(3, OUTPUT);
  pinMode(5, OUTPUT);

   LPS.attach(6, 570, 2350);           //inicializace serv, nastavení
   PPS.attach(4, 630, 2650);           //pinů, krajních poloh
  LZS.attach(7, 570, 2450);
  PZS.attach(8, 620, 2550);
}

void motor(int strana, int rychlost) {   //funkce pro ovládání motoru
  rychlost = constrain(rychlost, -255, 255);  //ujištění se, zda je vstupní
                                          proměnná v mezích, pokud ne,
                                    tak se omezí na maximální hodnotu
  bool smer = ((rychlost > 1) ? 1 : 0);   // zjištění směru otáčení motorů
  rychlost = abs(rychlost);    //úprava vstupní hodnoty na kladnou hodnotu
```

```
  if (strana == 0) {                                    //levá strana
    digitalWrite(0, smer);
    digitalWrite(2, !smer);
    analogWrite(5, rychlost);
  }
  else if (strana == 1) {                               //pravá strana
    digitalWrite(1, smer);
    digitalWrite(A1,!smer);
    analogWrite(3, rychlost);
  }
}

void serva_domu() {                     //testovací funkce, zavolá všechna
                                                serva do středové polohy
  LPS.writeMicroseconds(LPS_stred);
  PPS.writeMicroseconds(PPS_stred);
  LZS.writeMicroseconds(LZS_stred);
  PZS.writeMicroseconds(PZS_stred);
}
```

Dále bych uvedl program, který běží ve vysílači.

```
#include <Wire.h>
#include <LiquidCrystal_I2C.h>
#include <SPI.h>
#include <Mirf.h>
#include <nRF24L01.h>
#include <MirfHardwareSpiDriver.h>
#include <EEPROM.h>
#include <LCDMenuLib.h>
#include <DS1307new.h>

#define    podsviceni    9
#define    joylx         A3
#define    joyly         A2
#define    joypx         A1
#define    joypy         A0
#define    tlnah         5
#define    tldol         4
#define    tlano         3
#define    tlne          2
#define    vyp           10
#define    deadzone      40

#define _LCDMenuLib_LCD_cols              16
#define _LCDMenuLib_LCD_rows              2
#define _LCDMenuLib_LCD_addr              0x20
#define _LCDMenuLib_LCD_e                 2
#define _LCDMenuLib_LCD_rw                1
#define _LCDMenuLib_LCD_rs                0
#define _LCDMenuLib_LCD_backlight         3
#define _LCDMenuLib_LCD_backlight_pol     POSITIVE
#define _LCDMenuLib_LCD_dat4              4
#define _LCDMenuLib_LCD_dat5              5
#define _LCDMenuLib_LCD_dat6              6
#define _LCDMenuLib_LCD_dat7              7
#define _LCDMenuLib_cfg_initscreen        1
#define _LCDMenuLib_cfg_initscreen_time   30000
#define _LCDMenuLib_cfg_scrollbar       1
#define _LCDMenuLib_cfg_lcd_standard    0
#define _LCDMenuLib_cfg_press_time        400
#define _LCDMenuLib_cnt                   19

LCDMenuLib_init(_LCDMenuLib_cnt);


LCDMenuLib_element(0   , Item          , Item_1         , "Jizdni rezimy"
, FUNC);
LCDMenuLib_element(1   , Item_1        , Item_1_1       , "StabilizaceX,Y"
, stabilizaceXY);
LCDMenuLib_element(2   , Item_1        , Item_1_2       , "Stabilizace X"
, stabilizaceX);
LCDMenuLib_element(3   , Item_1        , Item_1_3       , "Stabilizace Y"
, stabilizaceY);
LCDMenuLib_element(4   , Item_1        , Item_1_4       , "Nestabilizovat"
, stabilizaceN);
LCDMenuLib_element(5   , Item_1        , Item_1_5       , "Zamavat"
, FUNC);
LCDMenuLib_element(6   , Item_1_5      , Item_1_5_1     , "Pravou predni"
, FUNC);
```

```
LCDMenuLib_element(7  , Item_1_5     , Item_1_5_2     , "Levou predni"
, FUNC);
LCDMenuLib_element(8  , Item_1_5     , Item_1_5_3     , "Pravou zadni"
, FUNC);
LCDMenuLib_element(9  , Item_1_5     , Item_1_5_4     , "Levou zadni"
, FUNC);
LCDMenuLib_element(10 , Item_1       , Item_1_6       , "Pejsek"
, pes);
LCDMenuLib_element(11 , Item         , Item_2         , "Nastaveni"
, FUNC);
LCDMenuLib_element(12 , Item_2       , Item_2_1       , "Test tlacitek"
, test);
LCDMenuLib_element(13 , Item_2       , Item_2_2       , "Kalibrace"
, kalibrace);
LCDMenuLib_element(14 , Item_2       , Item_2_3       , "Nastaveni jasu"
, jas);
LCDMenuLib_element(15 , Item_2       , Item_2_4       , "Nastaveni casu"
, nastaveni_casu);
LCDMenuLib_element(16 , Item_2       , Item_2_5       , "Ukazat napeti"
, ukazat_napeti);
LCDMenuLib_element(17 , Item         , Item_3         , "Vypnout"
, FUNC);
LCDMenuLib_element(18 , Item_3       , Item_3_1       , "Ano    Vypnout?"
, vypnout);
LCDMenuLib_element(19 , Item_3       , Item_3_2       , "Ne"
, zpet);
LCDMenuLib_createMenu(_LCDMenuLib_cnt);

//LiquidCrystal_I2C lcd(0x20, 2, 1, 0, 4, 5, 6, 7, 3, POSITIVE);

int sloupec, radek;
int i = 0;
long poloha_joylx, poloha_joyly, poloha_joypx, poloha_joypy;
int stred_joylx, stred_joyly, stred_joypx, stred_joypy;
int min_joylx, min_joyly, min_joypx, min_joypy;
int max_joylx, max_joyly, max_joypx, max_joypy;
int jasint,napeti,napeti_ukazatel;
int stabilizace;
unsigned long nahoru, dolu, mode = 1, stabilizace1, zdrzeni = 400;
unsigned long data = 0;
byte bat0[8]={
  B01110,  B11111,  B10001,  B10001,  B10001,  B10001,  B10001,  B11111};
byte bat1[8]={
  B01110,  B11111,  B10001,  B10001,  B10001,  B10001,  B11111,  B11111};
byte bat2[8]={
  B01110,  B11111,  B10001,  B10001,  B10001,  B11111,  B11111,  B11111};
byte bat3[8]={
  B01110,  B11111,  B10001,  B10001,  B11111,  B11111,  B11111,  B11111};
byte bat4[8]={
  B01110,  B11111,  B10001,  B11111,  B11111,  B11111,  B11111,  B11111};
byte bat5[8]={
  B01110,  B11111,  B11111,  B11111,  B11111,  B11111,  B11111,  B11111};



void setup() {

  initializovat();
  nastaveni();
  pinMode(10, OUTPUT);
  digitalWrite(10, LOW);
```

```
  LCDMenuLib_setup(_LCDMenuLib_cnt);
  lcd.clear();
  zpet();

}
```

```
void loop() {
  LCDMenuLib_control_digital();
  LCDMenuLib_loop();


  readJoylx();
  readJoypy();
  poloha_joylx += 300;
  poloha_joylx *= 1000;
  poloha_joypy += 300;
  stabilizace1 = stabilizace * 10000000;
  if (mode == 1)
    data = poloha_joylx + poloha_joypy + nahoru + dolu + (mode * 100000000)
 + stabilizace1;
  Mirf.send((byte *)&data);
  while (Mirf.isSending());


  Serial.println(data);


}
```

A pomocné funkce

```
#define    podsviceni    9
#define    joylx         A3
#define    joyly         A2
#define    joypx         A1
#define    joypy         A0
#define    tlnah         5
#define    tldol         4
#define    tlano         3
#define    tlne          2
#define    baterka       A6
```

```
void test() {
  int testing = 0;
  if (!LCDML.FuncInit()) {
    while (testing < 10) {
      testing = 0;
      lcd.clear();
      radek = (analogRead(joypy) > stred_joypy) ? 0 : 1;
      sloupec = (analogRead(joypx) > stred_joypx) ? 10 : 11;
      lcd.setCursor(sloupec, radek);
      lcd.print("X");
      radek = (analogRead(joyly) > stred_joyly) ? 0 : 1;
      sloupec = (analogRead(joylx) > stred_joylx) ? 3 : 4;
      lcd.setCursor(sloupec, radek);
      lcd.print("X");
      lcd.setCursor(0, 0);
      (digitalRead(tlnah)) ? lcd.print(" ") : lcd.print("O");
      lcd.setCursor(0, 1);
      (digitalRead(tldol)) ? lcd.print(" ") : lcd.print("O");
      lcd.setCursor(15, 0);
      lcd.print(" ");
      lcd.setCursor(15, 1);
      (digitalRead(tlano)) ? lcd.print(" ") : lcd.print("O");
      delay(100);
      while (!digitalRead(tlne)) {
        lcd.setCursor(15, 0);
        testing++;
        lcd.print(10 - testing);
        delay(100);
      }
    }
  }
  if ( LCDML.FuncEnd(1, 0, 0, 0, 0, 0)) {
  }


}


void initializovat() {
  Serial.begin(9600);
  pinMode(podsviceni, OUTPUT);
  jasint = EEPROM.read(12);
  map(jasint, 0, 100, 0, 255);
  analogWrite(podsviceni, map(jasint, 0, 100, 0, 255));
  pinMode(joylx, INPUT);
  pinMode(joyly, INPUT);
  pinMode(joypx, INPUT);
  pinMode(joypy, INPUT);
  pinMode(tlnah, INPUT);
  pinMode(baterka,INPUT);
  digitalWrite(tlnah, HIGH);
  pinMode(tldol, INPUT);
  digitalWrite(tldol, HIGH);
  pinMode(tlano, INPUT);
  digitalWrite(tlano, HIGH);
  pinMode(tlne, INPUT);
  digitalWrite(tlne, HIGH);


  Mirf.spi = &MirfHardwareSpi;
  Mirf.init();
```

```
  Mirf.setRADDR((byte *)"robot");
  Mirf.setTADDR((byte *)"vysilac");
  Mirf.payload = sizeof(unsigned long);                          //delka
posilanych dat
  Mirf.channel = 22;
  Mirf.config();
  //  Mirf.configRegister(RF_SETUP, 0x06);        //nastaveni kmitoctu na
1MHz pro zvyseni dosahu

}

void nastaveni() {
  stred_joylx = EEPROM.read(0) + 300;
  stred_joyly = EEPROM.read(1) + 300;
  stred_joypx = EEPROM.read(2) + 300;
  stred_joypy = EEPROM.read(3) + 300;
  min_joylx = EEPROM.read(4);
  min_joyly = EEPROM.read(5);
  min_joypx = EEPROM.read(6);
  min_joypy = EEPROM.read(7);
  max_joylx = EEPROM.read(8) + 700;
  max_joyly = EEPROM.read(9) + 700;
  max_joypx = EEPROM.read(10) + 700;
  max_joypy = EEPROM.read(11) + 700;

}

void kalibrace() {
  if (!LCDML.FuncInit()) {
    while (!digitalRead(tlano));
    while (digitalRead(tlano)) {
      lcd.clear();
      lcd.setCursor(0, 0);
      lcd.print("Nastavte packy do stredove polohy");
      if (!digitalRead(tlano))
        break;
      delay(1000);
      for (int scroll = 0; scroll < 18; scroll++) {
        lcd.scrollDisplayLeft();
        if (!digitalRead(tlano))
          break;
        delay(300);
      }
      lcd.setCursor(18, 1);
      lcd.print("a stisknete OK");
      if (!digitalRead(tlano))
        break;
      delay(2000);
    }
    lcd.clear();
    int stred_joylx = analogRead(joylx) - 300;
    lcd.setCursor(0, 0);
    lcd.print(stred_joylx + 300);
    int stred_joyly = analogRead(joyly) - 300;
    lcd.setCursor(0, 1);
    lcd.print(stred_joyly + 300);
    int stred_joypx = analogRead(joypx) - 300;
    lcd.setCursor(10, 0);
    lcd.print(stred_joypx + 300);
    int stred_joypy = analogRead(joypy) - 300;
    lcd.setCursor(10, 1);
```

```
lcd.print(stred_joypy + 300);
while (!digitalRead(tlano));
while (digitalRead(tlano));
EEPROM.write(0, stred_joylx);
EEPROM.write(1, stred_joyly);
EEPROM.write(2, stred_joypx);
EEPROM.write(3, stred_joypy);

lcd.clear();
lcd.print("Ulozeno!");
delay(1000);
min_joylx = stred_joylx;
min_joyly = stred_joyly;
min_joypx = stred_joypx;
min_joypy = stred_joypy;
max_joylx = stred_joylx;
max_joyly = stred_joyly;
max_joypx = stred_joypx;
max_joypy = stred_joypy;
lcd.clear();
lcd.setCursor(0, 0);
lcd.print("Nastavte extremn");
lcd.setCursor(0, 1);
lcd.print("i hodnoty a OK");

while (digitalRead(tlano)) {
  int minjoylx = analogRead(joylx);
  if (minjoylx < min_joylx)
    min_joylx = minjoylx;
  int minjoyly = analogRead(joyly);
  if (minjoyly < min_joyly)
    min_joyly = minjoyly;
  int minjoypx = analogRead(joypx);
  if (minjoypx < min_joypx)
    min_joypx = minjoypx;
  int minjoypy = analogRead(joypy);
  if (minjoypy < min_joypy)
    min_joypy = minjoypy;
  int maxjoylx = analogRead(joylx);
  if (maxjoylx > max_joylx)
    max_joylx = maxjoylx;
  int maxjoyly = analogRead(joyly);
  if (maxjoyly > max_joyly)
    max_joyly = maxjoyly;
  int maxjoypx = analogRead(joypx);
  if (maxjoypx > max_joypx)
    max_joypx = maxjoypx;
  int maxjoypy = analogRead(joypy);
  if (maxjoypy > max_joypy)
    max_joypy = maxjoypy;
}

lcd.clear();
lcd.print(min_joylx);
lcd.print("-");
lcd.print(max_joylx);
lcd.print("  ");
lcd.print(min_joypx);
lcd.print("-");
lcd.print(max_joypx);
lcd.setCursor(0, 1);
```

```
    lcd.print(min_joyly);
    lcd.print("-");
    lcd.print(max_joyly);
    lcd.print("  ");
    lcd.print(min_joypy);
    lcd.print("-");
    lcd.print(max_joypy);
    while (!digitalRead(tlano));
    while (digitalRead(tlano));
    while (!digitalRead(tlano));
    EEPROM.write(4, min_joylx);
    EEPROM.write(5, min_joyly);
    EEPROM.write(6, min_joypx);
    EEPROM.write(7, min_joypy);
    EEPROM.write(8, max_joylx - 700);
    EEPROM.write(9, max_joyly - 700);
    EEPROM.write(10, max_joypx - 700);
    EEPROM.write(11, max_joypy - 700);
    lcd.clear();
    lcd.print("Ulozeno!");
    delay(1000);
  }
  if ( LCDML.FuncEnd(1, 0, 0, 0, 0, 0)) {
  }
}



void LCDMenuLib_control_digital()
{
#define _BUTTON_digital_enter         3
  pinMode(_BUTTON_digital_enter, INPUT);
  digitalWrite(_BUTTON_digital_enter, HIGH);
#define _BUTTON_digital_up            5
  pinMode(_BUTTON_digital_up, INPUT);
  digitalWrite(_BUTTON_digital_up, HIGH);
#define _BUTTON_digital_down          4
  pinMode(_BUTTON_digital_down, INPUT);
  digitalWrite(_BUTTON_digital_down, HIGH);
  //optional
#define _BUTTON_digital_enable_quit   1
#define _BUTTON_digital_quit          2
  pinMode(_BUTTON_digital_quit, INPUT);
  digitalWrite(_BUTTON_digital_quit, HIGH);
  //optional
#define _BUTTON_digital_enable_lr     0
#define _BUTTON_digital_left          0
#define _BUTTON_digital_right         0

  if (!digitalRead(_BUTTON_digital_enter) ||
!digitalRead(_BUTTON_digital_up) || !digitalRead(_BUTTON_digital_down)
    || (!digitalRead(_BUTTON_digital_quit) && _BUTTON_digital_enable_quit
== 1)
    || ((!digitalRead(_BUTTON_digital_left) ||
!digitalRead(_BUTTON_digital_right)) && _BUTTON_digital_enable_lr == 1))
  {
    if (LCDML.Timer(g_LCDMenuLib_press_time, _LCDMenuLib_cfg_press_time))
    {
      if (!digitalRead(_BUTTON_digital_enter)) {
        LCDML.Button_enter();
```

```
      }
      else if (!digitalRead(_BUTTON_digital_up)) {
        LCDML.Button_up_down_left_right(_LCDMenuLib_button_up);
      }
      else if (!digitalRead(_BUTTON_digital_down)) {
        LCDML.Button_up_down_left_right(_LCDMenuLib_button_down);
      }
      else if (!digitalRead(_BUTTON_digital_quit) &&
_BUTTON_digital_enable_quit == 1) {
        LCDML.Button_quit();
      }
      else if (!digitalRead(_BUTTON_digital_left) &&
_BUTTON_digital_enable_lr == 1) {
        LCDML.Button_up_down_left_right(_LCDMenuLib_button_left);
      }
      else if (!digitalRead(_BUTTON_digital_right) &&
_BUTTON_digital_enable_lr == 1) {
        LCDML.Button_up_down_left_right(_LCDMenuLib_button_right);
      }
    }
  }
}



void FUNC_init_screen(void) {

  if (!LCDML.FuncInit())
    lcd.clear();
  zobrazit_cas();
  if (!digitalRead(tlnah))
    nahoru = 1000000;
  else
    nahoru = 0;
  if (!digitalRead(tldol))
    dolu = 2000000;
  else
    dolu = 0;
  if (LCDML.FuncEnd(0, 0, 0, 0, 0, 1)) /* (direct, up, down, left, right)
*/
  {
  }
}




void jas() {
  if (!LCDML.FuncInit()) {
    lcd.clear();
    int jaspred = jasint;
    while (!digitalRead(tlano));
    while (digitalRead(tlano) && digitalRead(tlne)) {
      jasint = constrain(jasint, 0, 100);
      lcd.setCursor(0, 0);
      lcd.print("Jas");
      lcd.setCursor(12, 1);
      lcd.print("   ");
```

```
      lcd.setCursor(12, 1);
      lcd.print(jasint);
      lcd.setCursor(15, 1);
      lcd.print("%");
      if (!digitalRead(tldol))
        jasint--;
      if (!digitalRead(tlnah))
        jasint++;
      jasint = constrain(jasint, 0, 100);
      analogWrite(podsviceni, map(jasint, 0, 100, 0, 255));
      delay(100);
    }
    if (!digitalRead(tlne)) {
      jasint = map(jaspred, 0, 100, 0, 255);
      analogWrite(podsviceni, jasint);
    }
  }
  if (LCDML.FuncEnd(1, 0, 0, 0, 0, 0)) { /* (direct, up, down, left, right)
*/
    EEPROM.write(12, jasint);
    while (!digitalRead(tlano) || !digitalRead(tlne));
  }
}



void vypnout() {
  lcd.clear();
  lcd.setCursor(2, 0);
  lcd.print("Nashledanou!");
  for (int vyp_counter = map(jasint, 0, 100, 0, 255); vyp_counter > 0;
vyp_counter--) {
    analogWrite(podsviceni, vyp_counter);
    delay(1000 / jasint);
  }
  if (jasint == 0)
    delay(1000);
  digitalWrite(vyp, HIGH);
}


void zpet() {
  LCDML.Button_quit();
}



void zobrazit_cas() {
  RTC.getTime();
  lcd.setCursor(4, 0);


  if (RTC.hour < 10) {
    lcd.print("0");
    lcd.print(RTC.hour, DEC);
  }
  else
    lcd.print(RTC.hour, DEC);
  lcd.print(":");
  if (RTC.minute < 10) {
    lcd.print("0");
```

```
  lcd.print(RTC.minute, DEC);
}
else
  lcd.print(RTC.minute, DEC);
lcd.print(":");
if (RTC.second < 10) {
  lcd.print("0");
  lcd.print(RTC.second, DEC);
}
else
  lcd.print(RTC.second, DEC);
lcd.setCursor(0, 1);
lcd.print(RTC.day, DEC);
lcd.print(".");
lcd.print(RTC.month, DEC);
lcd.print(".");
lcd.print(RTC.year % 1000, DEC);
lcd.print(" ");


switch (RTC.dow)
{
case 0:
  lcd.print("Nedele");
  break;
case 1:
  lcd.print("Pondeli");
  break;
case 2:
  lcd.print("Utery");
  break;
case 3:
  lcd.print("Streda");
  break;
case 4:
  lcd.print("Ctvrtek");
  break;
case 5:
  lcd.print("Patek");
  break;
case 6:
  lcd.print("Sobota");
  break;
case 7:
  lcd.print("Nedele");
  break;
}


napeti=map(analogRead(baterka),0,1023,0,10000);
napeti_ukazatel=map(constrain(napeti,5800,8400),6000,8200,0,5);
switch (napeti_ukazatel){
case 0:
  lcd.createChar(5, bat0);
  break;
case 1:
  lcd.createChar(5, bat1);
  break;
case 2:
  lcd.createChar(5, bat2);
  break;
```

```
  case 3:
    lcd.createChar(5, bat3);
    break;
  case 4:
    lcd.createChar(5, bat4);
    break;
  case 5:
    lcd.createChar(5, bat5);
    break;
  }
  lcd.setCursor(0, 0);
  lcd.write(5);
  Serial.println(napeti);
  Serial.println(napeti_ukazatel);
}

void ukazat_napeti() {
  if (!LCDML.FuncInit()) {
    lcd.clear();
    lcd.setCursor(0,0);
    lcd.print("Napeti baterii");
  }

  lcd.setCursor(11,1);
  lcd.print(map(analogRead(baterka),0,1023,0,10));
  lcd.print(",");
  lcd.print(map(analogRead(baterka),0,1023,0,1000)%100);
  lcd.print("V");

  if (LCDML.FuncEnd(0, 0, 0, 0, 1, 1)){}
  }



  void readJoylx() {
    poloha_joylx = map(analogRead(joylx), min_joylx, max_joylx, -270 -
deadzone, 270 + deadzone);
    if (abs(poloha_joylx) > deadzone) {
      if ( poloha_joylx > 0)
        poloha_joylx -= deadzone;
      else
        poloha_joylx += deadzone;
    }
    else
      poloha_joylx = 0;
  }


void readJoyly() {
  poloha_joyly = map(analogRead(joyly), min_joyly, max_joyly, -270 -
deadzone, 270 - deadzone);
  if (abs(poloha_joylx) > deadzone) {
    if ( poloha_joyly > 0)
      poloha_joyly -= deadzone;
    else
      poloha_joyly += deadzone;
  }
  else
    poloha_joyly = 0;
}
```

```
void readJoypx() {
  poloha_joypx = map(analogRead(joypx), min_joypx, max_joypx, -270 -
deadzone, 270 - deadzone);
  if (abs(poloha_joypx) > deadzone) {
    if ( poloha_joypx > 0)
      poloha_joypx -= deadzone;
    else
      poloha_joypx += deadzone;
  }
  else
    poloha_joypx = 0;
}

void readJoypy() {
  poloha_joypy = map(analogRead(joypy), min_joypy, max_joypy, -270 -
deadzone, 270 - deadzone);
  if (abs(poloha_joypy) > deadzone) {
    if ( poloha_joypy > 0)
      poloha_joypy -= deadzone;
    else
      poloha_joypy += deadzone;
  }
  else
    poloha_joypy = 0;
}


void stabilizaceXY() {
  //  LCDML.FuncInit();
  mode=1;
  stabilizace = 0;
  //  LCDML.FuncEnd(1,0,0,0,0,0);
}

void stabilizaceX() {
  // LCDML.FuncInit();
  mode=1;
  stabilizace = 1;
  // LCDML.FuncEnd(1,0,0,0,0,0);
}

void stabilizaceY() {
  //  LCDML.FuncInit();
  mode=1;
  stabilizace = 2;
  //  LCDML.FuncEnd(1,0,0,0,0,0);
}

void stabilizaceN() {
  //  LCDML.FuncInit();
  mode=1;
  stabilizace = 3;
  // LCDML.FuncEnd(1,0,0,0,0,0);
}
void pes() {
  if (!LCDML.FuncInit()) {
    lcd.clear();
    lcd.print("         Zdrzeni");
    mode=3;
  }
```

```
  if (!(digitalRead(tlnah))) {
    zdrzeni += 1;
  }
  if (!(digitalRead(tldol))) {
    zdrzeni -= 1;
  }
  zdrzeni = constrain(zdrzeni, 100, 9999);
  lcd.setCursor(0, 1);
  lcd.print(zdrzeni);
  data = (mode * 100000000) + zdrzeni;

  Mirf.send((byte *)&data);
  while (Mirf.isSending());
  if (LCDML.FuncEnd(0, 0, 0, 0, 1, 0))
    mode = 1;
}

void nastaveni_casu() {
  LCDML.FuncInit();
  nastavenicasu();
  if (LCDML.FuncEnd(1, 0, 0, 0, 0, 0))
    while (!digitalRead(tlano));
}

void nastavenicasu() {
  int nastaveni_casu_cislo = 0;
  int hodina, minuta, sekunda, den, mesic, rok;
  lcd.clear();
  RTC.getTime();
  hodina = RTC.hour;

  for (bool konec = 0; konec == 0;) {
    if (!digitalRead(tlne)) {
      while (!digitalRead(tlne));
      lcd.noCursor();
      return;
    }
    RTC.getTime();
    if (nastaveni_casu_cislo < 1) {
      minuta = RTC.minute;
    }
    if (nastaveni_casu_cislo < 2) {
      sekunda = RTC.second;
    }
    if (nastaveni_casu_cislo < 3) {
      den = RTC.day;
    }
    if (nastaveni_casu_cislo < 4) {
      mesic = RTC.month;
    }
    if (nastaveni_casu_cislo < 5) {
      rok = RTC.year;
    }

    if (nastaveni_casu_cislo == 0) {
      lcd.setCursor(1, 0);
      lcd.cursor();
      delay(100);
      if (!digitalRead(tlnah))
        hodina++;
      while (!digitalRead(tlnah));
```

```
  if (!digitalRead(tldol))
    hodina--;
  while (!digitalRead(tldol));
}

if (nastaveni_casu_cislo == 1) {
  lcd.setCursor(4, 0);
  lcd.cursor();
  delay(100);
  if (!digitalRead(tlnah))
    minuta++;
  while (!digitalRead(tlnah));
  if (!digitalRead(tldol))
    minuta--;
  while (!digitalRead(tldol));
}

if (nastaveni_casu_cislo == 2) {
  lcd.setCursor(7, 0);
  lcd.cursor();
  delay(100);
  if (!digitalRead(tlnah))
    sekunda++;
  while (!digitalRead(tlnah));
  if (!digitalRead(tldol))
    sekunda--;
  while (!digitalRead(tldol));
}

if (nastaveni_casu_cislo == 3) {
  lcd.setCursor(1, 1);
  lcd.cursor();
  delay(100);
  if (!digitalRead(tlnah))
    den++;
  while (!digitalRead(tlnah));
  if (!digitalRead(tldol))
    den--;
  while (!digitalRead(tldol));
}

if (nastaveni_casu_cislo == 4) {
  lcd.setCursor(4, 1);
  lcd.cursor();
  delay(100);
  if (!digitalRead(tlnah))
    mesic++;
  while (!digitalRead(tlnah));
  if (!digitalRead(tldol))
    mesic--;
  while (!digitalRead(tldol));
}

if (nastaveni_casu_cislo == 5) {
  lcd.setCursor(9, 1);
  lcd.cursor();
  delay(100);
  if (!digitalRead(tlnah))
    rok++;
  while (!digitalRead(tlnah));
  if (!digitalRead(tldol))
```

```
    rok--;
  while (!digitalRead(tldol));
}
if (hodina > 23) {
  hodina = 0;
  //        den++;
}
if (hodina < 0) {
  hodina = 23;
  //        den--;
}

if (minuta > 59) {
  minuta = 0;
  //        hodina++;
}
if (minuta < 0) {
  minuta = 59;
  //        hodina--;
}

if (sekunda > 59) {
  sekunda = 0;
  //        minuta++;
}
if (sekunda < 0) {
  sekunda = 59;
  //        minuta--;
}

if (den > 31) {
  den = 0;
  //        mesic++;
}
if (den < 0) {
  den = 31;
  //        mesic--;
}

if (mesic > 12) {
  mesic = 0;
  //        rok++;
}
if (mesic < 0) {
  mesic = 12;
  //        rok--;
}

if (rok > 2200) {
  rok = 2000;
}
if (rok < 2000) {
  rok = 22000;
}


lcd.setCursor(0, 0);
if (hodina < 10)
  lcd.print("0");
lcd.print(hodina, DEC);
lcd.print(":");
```

```
    if (minuta < 10)
      lcd.print("0");
    lcd.print(minuta, DEC);
    lcd.print(":");
    if (sekunda < 10)
      lcd.print("0");
    lcd.print(sekunda, DEC);
    lcd.setCursor(0, 1);
    if (den < 10)
      lcd.print("0");
    lcd.print(den, DEC);
    lcd.print(".");
    if (mesic < 10)
      lcd.print("0");
    lcd.print(mesic, DEC);
    lcd.print(".");
    lcd.print(rok, DEC);

    if (!digitalRead(tlano))
      nastaveni_casu_cislo++;
    if (nastaveni_casu_cislo > 5) {
      RTC.fillByYMD(rok, mesic, den);
      RTC.fillByHMS(hodina, minuta, sekunda);
      RTC.setTime();
      RTC.startClock();
      lcd.noCursor();
      return;
    }
    while (!digitalRead(tlano))
      lcd.noCursor();
  }
}
```