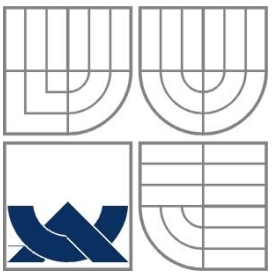


VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA STROJNÍHO INŽENÝRSTVÍ
ÚSTAV MECHANIKY TĚLES, MECHATRONIKY A BIOMECHANIKY

FACULTY OF MECHANICAL ENGINEERING
INSTITUTE OF SOLID MECHANICS, MECHATRONICS AND BIOMECHANICS

REALIZACE ÚLOHY „SLEDOVÁNÍ ČÁRY“ NA PLATFORMĚ EYEBOT

TITLE

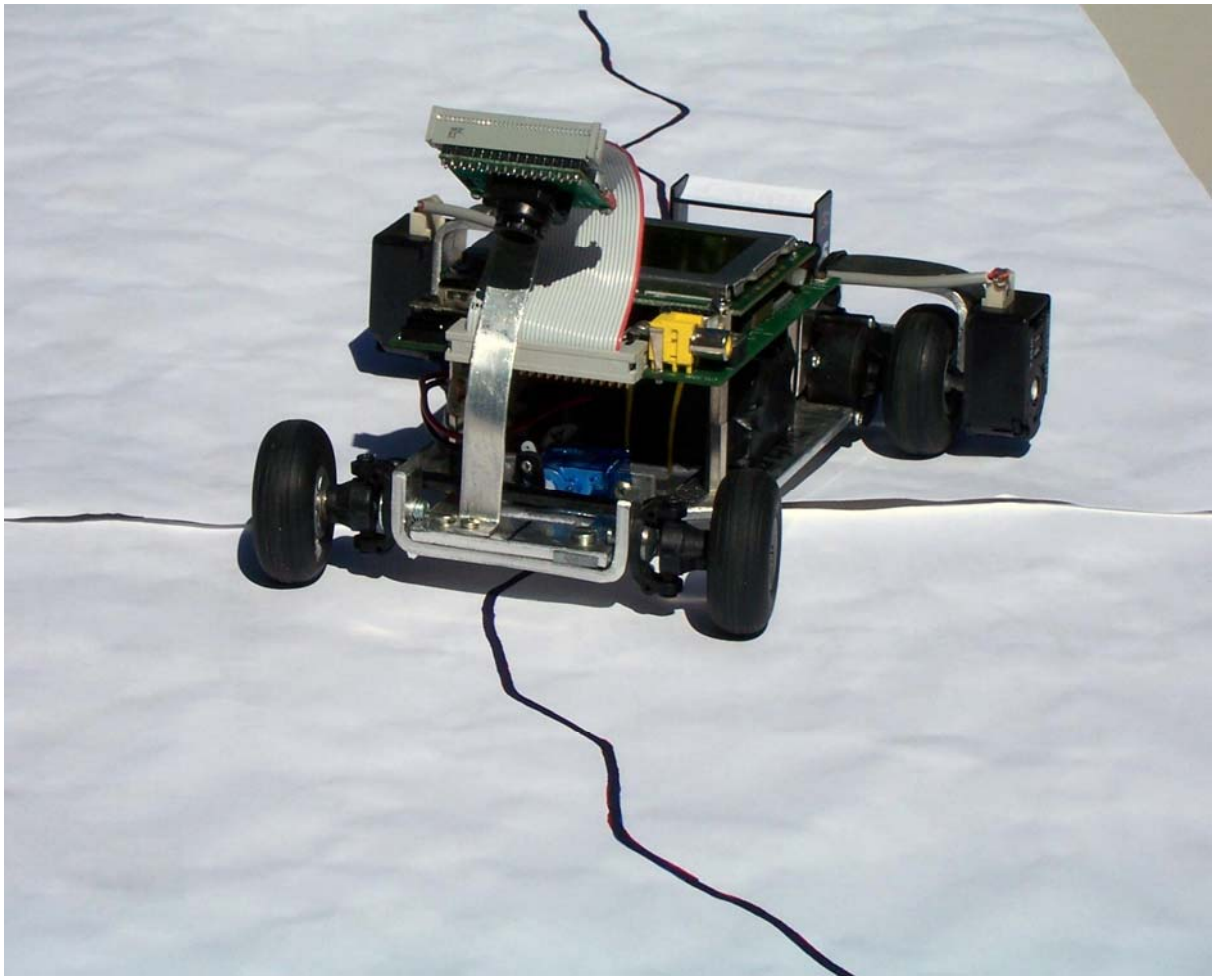
BAKALÁŘSKÁ PRÁCE
BACHELOR THESIS

AUTOR PRÁCE — ONDŘEJ VOKOUN
AUTHOR

VEDOUCÍ PRÁCE — ING. ROBERT GREPL, PH.D.
SUPERVISOR

BRNO 2007

BAKALÁŘSKÁ PRÁCE



REALIZACE ÚLOHY „SLEDOVÁNÍ ČÁRY“ NA PLATFORMĚ EYEBOT

vypracoval: Ondřej Vokoun
vedoucí práce: Ing. Robert Grepl, Ph.D.
obor: Aplikované vědy v inženýrství
specializace: Mechatronika
2008

Vysoké učení technické v Brně, Fakulta strojního inženýrství

Ústav mechaniky těles, mechatroniky a biomechaniky
Akademický rok: 2008/09

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

student(ka): Vokoun Ondřej

který/která studuje v **bakalářském studijním programu**

obor: **Mechatronika (3906R001)**

Ředitel ústavu Vám v souladu se zákonem č.111/1998 o vysokých školách a se Studijním a zkušebním řádem VUT v Brně určuje následující téma bakalářské práce:

Realizace úlohy "sledování čáry" na platformě Eyebot

v anglickém jazyce:

Implementation of "Line Following" task using platform Eyebot

Stručná charakteristika problematiky úkolu:

Práce se bude zabývat řízením jednoduchého modelu vozidla s řízenou nápravou při úloze "sledování čáry". Vozidlo se bude orientovat s využitím kamery integrované v platformě Eyebot.

Cíle bakalářské práce:

1) Navrhněte konstrukční řešení testovacího vozidla s přední řízenou a zadní hnanou nápravou.
2) Zajistěte výrobu testovacího vozidla.
3) Seznamte se s platformou Eyebot, především s algoritmy analýzy obrazu implementovaným v RoBIOS.
4) Implementujte algoritmus řízení pohybu vozidla po čáře a prakticky ověřte jeho funkčnost.

Seznam odborné literatury:

[1] DP Miroslav Světlík, 2008
[2] web stránky výrobce platformy Eyebot
[3] Valášek, M.: Mechatronika, Vydavatelství ČVUT 1995
[4] Dušek, F.: Matlab a Simulink, skriptum ČVUT
[5] Herout, P.: Učebnice jazyka C
[6] Noskievič: Modelování a identifikace systémů

Vedoucí bakalářské práce: Ing. Robert Grepl, Ph.D.

Termín odevzdání bakalářské práce je stanoven časovým plánem akademického roku 2008/09.

V Brně, dne 2.11.2008

L.S.

prof. Ing. Jindřich Petruška, CSc.
Ředitel ústavu

doc. RNDr. Miroslav Doupovec, CSc.
Děkan fakulty

ABSTRAKT

Tato bakalářská práce pojednává o využití platformy EyeBot v realizaci čtyřkolového vozidla, jež pomocí kamery sleduje tmavou linii, podle níž je zamýšlena jeho jízda. Eyebot je učební pomůcka, obsahující procesor, řadu vstupů a výstupů a to jak analogových tak digitálních a spoustu dalších komponent, která nám přibližuje postup moderní techniky a dává nám možnost seznámit se s funkcí a principy ovládání mechatronických soustav.

ABSTRACT

This bachelor work is focused on using Eyebot platform in construction of four-wheeled vehicle, that has objective to follow the dark line with a camera. Eyebot is a kind of educational robotic device, which helps us to understand closely modern technology and programmable systems with CPU, learn more about controlling and working with the mechatronical systems.

Prohlášení:

Prohlašuji, že jsem bakalářskou práci zpracoval samostatně na základě rad a pokynů vedoucího diplomové práce, svých znalostí, odborných konzultací a literárních zdrojů.

V Brně dne:

……
Ondřej Vokoun

Poděkování:

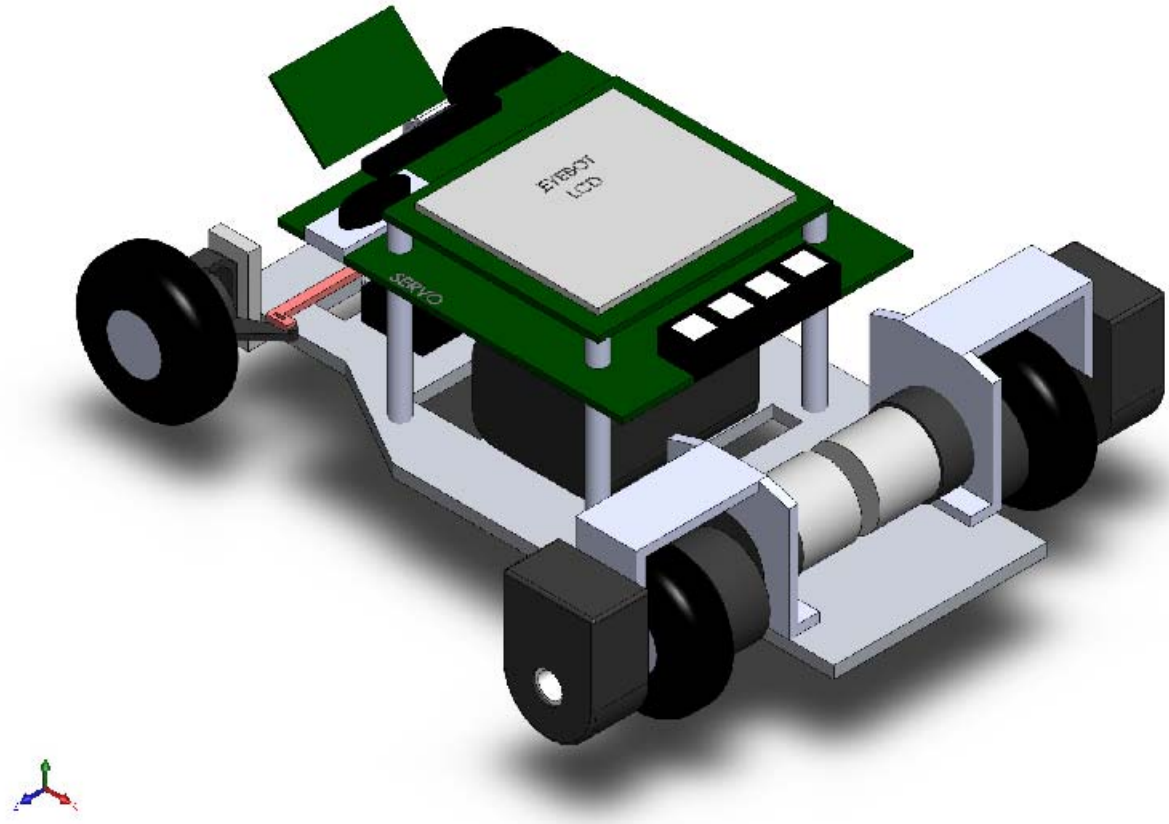
Na tomto místě bych rád poděkoval všem, kteří mi byli nápomocni při tvorbě bakalářské práce a během studií. Především bych rád poděkoval vedoucímu práce Ing. Robertu Greplovi, Ph.D. za odbornou pomoc a cenné rady. A taktéž rodině, jež mi umožnila bezstarostně studovat a věnovat svou energii na rozšiřování svých vědomostí.

Obsah:

1 ÚVOD

Technika, věda a poznání pronásleduje člověka již od prvopočátku existence. Od dob, kdy si náš prapředek uvědomil, že je schopen dokázat něco více než okolní živočichové a pomocí své rozvinutější nervové soustavy, důvtipu a vynalézavosti si může usnadnit život a dostat se na vývojový vrchol, kde představují největší hrozbu jen další lidé či přírodní živly. Vědomí této skutečnosti nás, jakožto moderního člověka, doposud neopustilo a i nadále se snažíme být na špici vývoje, ulehčit hlavou našim rukám, pochopit fyzikální i ostatní zákonitosti světa, pevně se jich chopit a dokázat využít v náš prospěch.

Robotika a roboti všeobecně jsou záležitostí pár posledních desetiletí. Největšího rozmachu bylo zaznamenáno s příchodem čipů, procesorů a pokročilejší elektroniky, které z pouze pohybujících se strojů, ve spojení s další nedílnou součástí-a to softwarem, tvoří vskutku „inteligentní zařízení“, jež dokáží téměř vše, když je tomu naučíme. Roboti byli revoluční vynález a v současné době jsou již nepostradatelné zařízení, která člověku značně ulehčují práci, dokáží podávat výkon i v prostředích, kam lidé nemohou, chrání naše životy při pomoci s likvidací nebezpečných látek, zvyšují produktivitu práce atd. Z těchto a dalších důvodů, domnívám se, je dobré pochopit, jak vlastně tato zařízení fungují, jak můžeme ovlivnit jejich chování, jak je přizpůsobit našim potřebám a účelně využít jejich potenciálu. Doufám tedy, že se mi do problematiky řízení a ovládání robotů podaří dostatečně proniknout ke zdárnému splnění úkolu.



Obr. 1.: Vizualizace vozidla

1.1 OBECNÁ CHARAKTERISTIKA PROBLEMATIKY

Problém řízení vozítka po předem stanovené cestě lze řešit vícero způsoby.Od jednodušších jako je přesné naprogramování cesty do paměti zařízení, za současného snímání potřebných údajů z postupu, k složitějším, jako je například satelitní navigace s problematičtějším počítáním a zjišťováním polohy. Mnou aplikované navádění pomocí kamery patří do skupiny nesložitých a dosti adaptabilních řešení, jelikož se předem nic neskriptuje a robot postupuje tak, jak „vede cesta“ a jak je naučen zvládat překážky či vybočení od standardu. Použití kamery má své výhody, jak již bylo zmíněno, má však také nevýhody jako je například potřeba údržby a čistoty zorné části kamery, potřeba jistých světelných podmínek, citlivost na nárazy, chvění a dalších, což do značné míry omezuje její použití v praxi.

Další otázkou bylo konstrukční provedení vozítka, každý způsob má své výhody oproti ostatním, každý také svá úskalí a ovládání každého je odlišné. V práci byl aplikován čtyřkolový podvozek, tak jak známe od aut se zadní hnanou a přední řiditelnou nápravou. Řízení tudíž probíhalo za pomoci serva, jež vykonávalo akční zásahy do korekce směru tak, jak rozhodoval software po získání informací z kamery. Bylo tedy nutné napsat program, který analýzou obrazu rozhodoval, zda má servo zasáhnout, jak a kam. To vše za současného ovládání také zadní nápravy, kde byly použity dva motory jako elektrický diferenciál, nutný ke zpomalení kola opisujícího při zatáčení menší poloměr kružnice, aby byla konstrukce s oběmi hnanými koly vůbec schopná zatočit.

O navigaci se starala již zmíněná procesorová platforma Eyebot, která umístěná přímo na pevném podvozku vozítka, napájená z baterií a s informacemi z kamery a enkodérů, po nahrání příslušného programu rozhodovala o dalším postupu po náhodně navržené a nakreslené trase z jedné linie.

2 PRÁCE S EYEBOTEM

V této kapitole se budeme zabývat charakteristikou EyeBota, vysvětlením způsobu práce s ním a s informačními zdroji, jež nám pomáhají uskutečnit zamýšlenou funkci.

2.1 NEŽ ZAČNEME

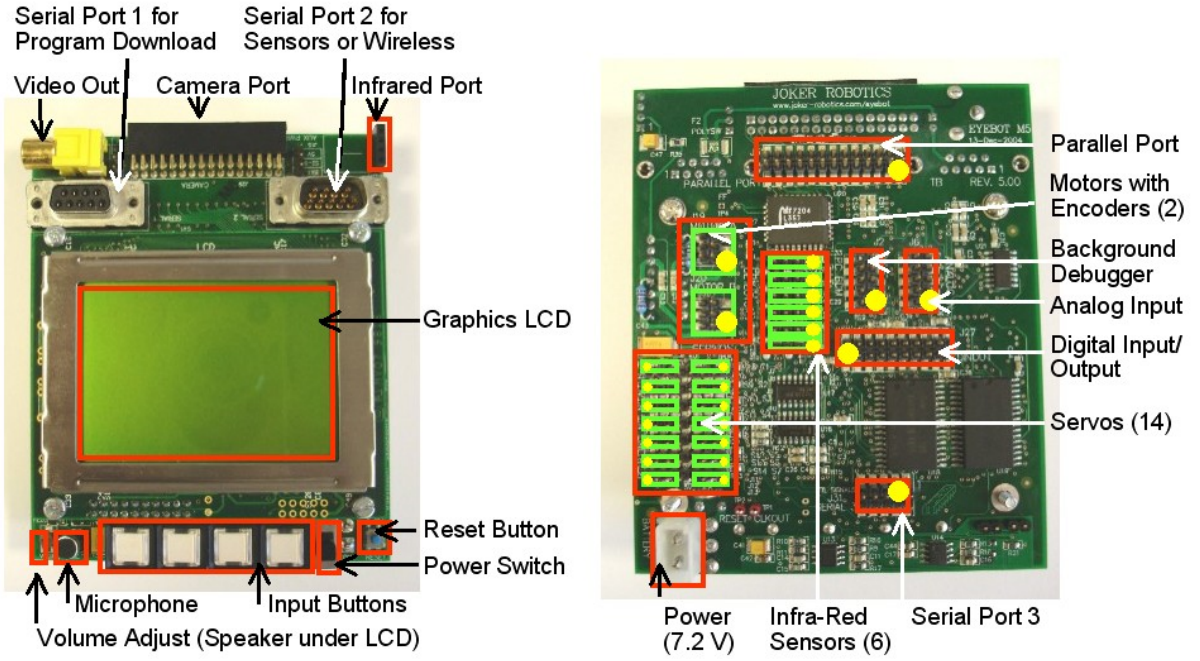
Podkapitola pojednávající o všem co je nutné udělat, nainstalovat a prostudovat ke zvládnutí práce s platformou EyeBot.

2.1.1 OBECNÉ INFORMACE

Výrobou EyeBota se zabývá firma Joker robotics, jejíž domovská stránka se nachází na www:
http://www.joker-robotics.com/eyebot/index.html
Na této adrese je obsažen také detailní popis Eyebota, včetně odkazů ke stažení potřebného softwaru a odkazy na další a významnější stránku, za kterou stojí strůjce a hlavní osoba zainteresovaná do tohoto projektu- prof.dr. Thomas Bräunl z University of Western Australia. Zde se nachází odkaz na stránku profesora Bräunla (po instalaci software rob65win.exe se vytvoří offline zástupce s většinou obsahu této stránky přímo na ploše vašeho PC):
http://robotics.ee.uwa.edu.au/eyebot/
Na stránce je přístupná charakteristika Eyebota, popis periferií, způsob práce s Eyebotem, často kladené dotazy, taktéž velmi užitečný FTP server se značným množstvím ukázkových programů a další náležitosti. Jedny z nejdůležitějších záložek jsou RoBIOS a HDT, což jsou knihovny předprogramovaných funkcí pro platformu- k bližšímu vysvětlení se dostaneme v následujících kapitolách. Téměř vše nutné pro práci s Eyebotem můžeme nalézt přímo zde, popřípadě v pracích jiných studentů, jež lze ze stránky také stáhnout a jež se povětšinou zabývají již praktickou aplikací v řešení určitého problému.

2.1.2 EYEBOT HARDWAROVĚ

EyeBot je kontroler pro mobilní roboty - jak chodící, tak kolové a dokonce i létající či plovoucí. Srdcem, které se stará o běh a správu všech připojených či připojitelných periferií, o převedení naprogramovaných funkcí v realitu je 32bitový, 25Mhz procesor Motorola 68332 (pozn.: ve verzi M5, jež je zde rozebírána). Na první pohled zaujme grafický LCD display (128x64 pixelů), pod nímž se nacházejí čtyři tlačítka k práci a ovládáni bota. Deska je dále osazena řadou vstupů a výstupů- digitálních i analogových. Mezi ně například patří tři sériové porty, jeden paralelní port, osm digitálních vstupů a výstupů, výstup pro dva motory s enkodéry. Dále je možno připojit čtrnáct serv, infračervené snímače polohy, barevnou kameru, bluetooth modul k bezdrátové komunikaci a další. Následuje obrázek bota s fyzickým rozložením důležitých částí:



Obr. 2.1.2: Pohled na EyeBota s popisem (verze M5)

Kompletní soupis a funkce pinů je dostupný na již zmíněné webové stránce prof. Bräunla v sekci: controller/hardware, pod pojmenováním- pinouts, tedy na přiloženém odkazu na tuto stránku, kde jsou dále k nalezení detailní datasheety, diagramy a schémata EyeBota:

http://robotics.ee.uwa.edu.au/eyebot/doc/hardware.html

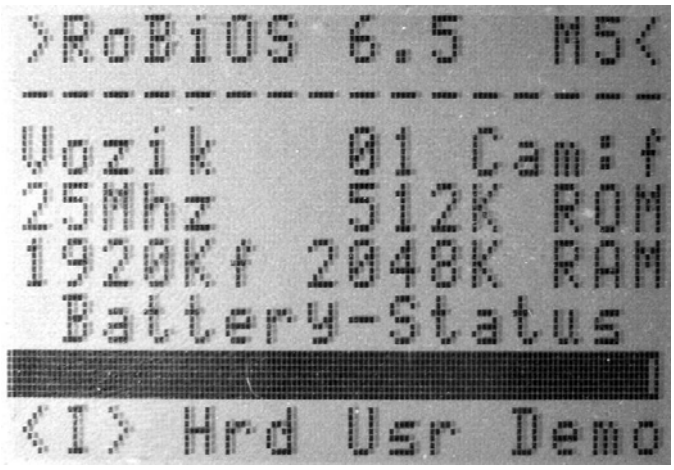
Použitý procesor v nynější době nepatří mezi výkonnější a vystačuje spíše na jednodušší aplikace, jež nekladou takové požadavky na výpočetní sílu kontroleru. Problémem tak může být kupříkladu připojení více komponent a jejich řízení zároveň, kdy dochází k nadměrnému zatížení a systém je tak náchylný na pád či zamrznutí. Také u použité CMOS kamery a následné potřeby zpracování obrazu dochází k problému s nedostatečnou snímkovou frekvencí. Kamera sice pro většinu aplikací nevyužívá svého plného rozlišení, nýbrž jen výsek o velikosti 82x62 pixelů, ale i tak při současném řízení, například dvou motorů jakožto elektrického diferenciálu, snímá rychlostí pouze tři snímky za sekundu (pozn.: doporučeno výrobcem pro aplikaci systému vω-řízení). Tyto důvody jsou vcelku limitující a nutí nás popřemýšlet, co vše je vůbec tento systém schopný zvládnout dříve než se do realizace daného problému pustíme.

2.1.3 PRVNÍ KROKY

Před započetím samotné práce s EyeBotem je doporučeno prostudovat přiloženou dokumentaci a následně se tak vyvarovat chyb, jež by mohly vést ke zničení přístroje.

K činnosti EyeBota je nutné napájení. Výrobcem udaná možná hodnota je mezi 6 až 9 volty stejnosměrného napětí, doporučeno je 7,2VDC a maximální proud je 3A. Při překročení prahové hodnoty proudu dojde ke shození napájení pojistkou, která má za úkol chránit hardware a tato je samoobnovitelná a není ji tedy nutné po zásahu měnit, stačí počkat pár minut, než se vrátí do svého původního stavu. Důležité je však dát pozor na správnou polaritu napájení, sic je proti tomuto bot chráněn diodou, ale i tak by se mohlo stát, že dojde k poruše - raději před připojením zkontrolujeme multimetrem správnou velikost a polaritu napájení!

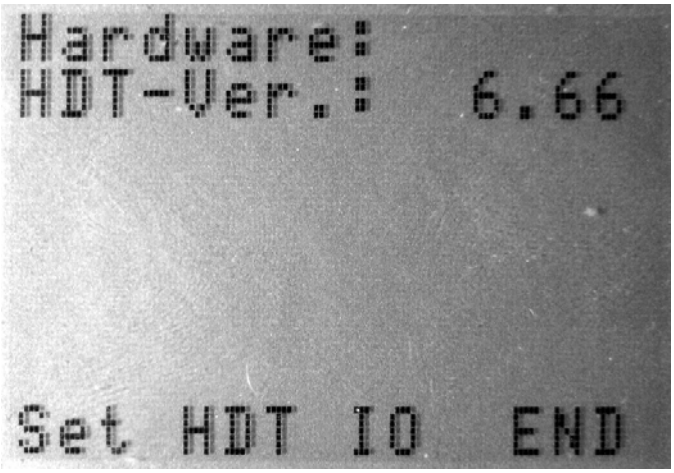
Dále můžeme přesunout startovací přepínač (pozn.: černý, dvoupolohový přepínač v pravém dolním rohu) do polohy "**ON**" a po zaznění startovacího zvuku a zobrazení loga na displeji nás program přivede na hlavní obrazovku, kde jsou zobrazeny informace o EyeBotovi a stavu baterie. Z úvodní obrazovky máme možnost pomocí čtyř ovládacích tlačítek k přístupu k hlavním sekcím EyeBota a ovládat jej tak (pozn.: tlačítko "**END**" ukončuje aktuální činnost a vrací nás o úroveň zpět):



obr. 2.1.3a: Pohled na úvodní obrazovku

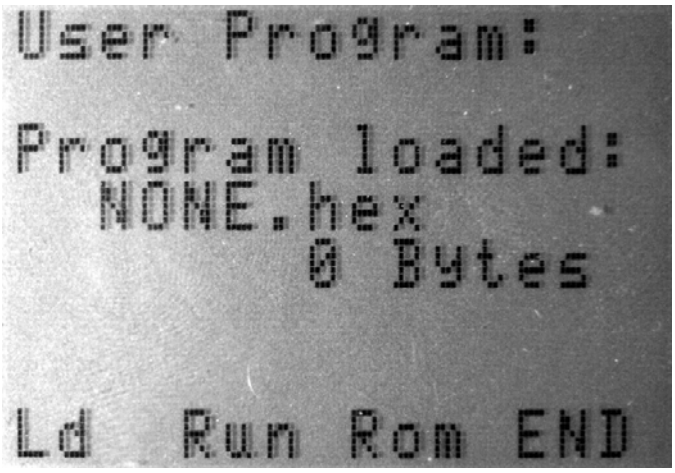
- **< I >**
  -zobrazí informace o autorech platformy
  -obsahuje podsekce: "**More**", jež slouží k zobrazení dalších informací a "**REG**", které zřejmě vypíše sériové číslo bezdrátového ovladače

- **Hrd**
  -přivede nás do sekce nastavení hardware či testování připojených komponent
  -**Set** – slouží k nastavení sériového a bezdrátového propojení s PC ("**Ser**" a "**Rmt**")
  -**HDT** – oddíl jež obsahuje testovací makra připojených periferií, nadefinovaných v hdt souboru, což nám ulehčuje práci s programováním dané součásti, při pouhé snaze o otestování (k problematice HDT se dostaneme podrobněji dále v kapitole 2.3)
  -**IO** –v této sekci můžeme nastavit digitální a paralelní port a také A/D převodník



obr. 2.1.3b: Sekce Hrd

- **Usr**
  -zřejmě nejdůležitější oddíl ve struktuře, slouží ke správě uživatelských programů
  -**Ld -** funkce pro nahrávání programů z počítače, zmáčkneme "**Ld"** a v počítači příslušný program odešleme (viz dále v sekci programování, kapitola 2.2.3)
  -**Run** – tlačítko ke spuštění programů načtených v paměti RAM, která slouží k práci se spouštěnými programy a která vzhledem ke konstrukci nedokáže po vypnutí bota a odpojení od napájení udržet data, narozdíl od paměti ROM
  -**Rom** – přístup do sekce jakési „pevné paměti", do které můžeme uložit až tři programy, ze které jdou následně nahrát do paměti RAM a poté spustit, tato paměť si vzhledem ke konstrukci drží svá data i bez napájení



obr. 2.1.3c: Sekce Usr

- **Demo**
  -obsahuje demonstrační programy fungování EyeBota a periferií (pozn.: v EyeBotovi může být tato sekce prázdná, dema jsou však přítomna ve složce, která se vytvořila v počítači po nainstalování programu rob65win.exe)

2.2 TVORBA A KOMPILACE PROGRAMŮ

Programování bota je hlavní úlohou práce s tímto zařízením a dává nám řadu možností např. již ve volbě programovacího jazyka. Můžeme pracovat v jazykovém prostředí ANSI C, ANSI C++ či v ASSEMBLERU, též je možné jazyky kombinovat (např. ASM a C) a v důsledku se liší pouze způsobem kompilace. Nicméně nejjednodušší variantou je jazyk C, pro nějž jsou v systému RoBIOS předprogramované funkce ovládání.

Nejprve je důležité stáhnout ze stránek prof. Bräunla kompilátor s přidruženým softwarem a nainstalovat do počítače (software běží jak pod Windows tak pod Linuxem, mnou zkoušená aktuální verze a použitelná pro systém Windows - rob65win.exe, dále bude práce popisovat pouze činnost v prostředí Windows), software je umístěn na stránce:

http://robotics.ee.uwa.edu.au/eyebot/ftp/

Po nainstalování se na ploše vytvoří zástupce offline verze stránek prof. Bräunla s názvem- EyeDoc a také dialogové okno s příkazovým řádkem- RoBIOS Prompt, ve kterém poté kompilujeme námi vytvořené programy pro potřeby EyeBota. Výběr programového prostředí k vytváření vlastních programů je pouze na uživateli samotném a je doporučeno zvolit

program, na nějž jste zvyklí a jenž je přehledný (pozn.: mnou používané Dev-C++ a Borland Turbo C++, což jsou oba zdarma přístupné programy). Po nainstalování kompilátoru a programu pro tvorbu je taktéž doporučeno podniknout jeden důležitý krok a tím je překopírování "include" souborů do složky editačního programu.
Tedy: otevřeme složku kompilátoru a najdeme složku– "include" (v mém případě: F:/EyeBot/robios/include) a obsah složky poté zkopírujeme do "include" složky editoru. Taktéž je možnost dané postahovat z FTP serveru stránky:

http://robotics.ee.uwa.edu.au/eyebot/ftp/ROBIOS/include/

Tento krok je dobré provést z toho důvodu, že veškeré utvořené programy musí obsahovat direktivu:

```
#include "eyebot.h"
```

která stejně jako další direktivy v programovacím jazyce odkazuje na hlavičkové soubory (tzv. header soubory s příponou- .h, jež obsahují předdefinované funkce či další často používané datové typy atd.). Složka obsahuje také další hlavičkové soubory, na které je odkaz v již zmíněném eyebot.h a tyto obsahují předdefinované funkce RoBIOSu, utvořené pro EyeBota a při snaze kompilace právě přímo v editačním programu by byly neznámé a kompilace by se tím pádem nezdařila. Daný hlavičkový soubor si, jako všechny ostatní přítomné, můžeme prohlédnout a v případě potřeby i poupravit.

2.2.1 TVORBA PROGRAMŮ V JAZYCE C

Po provedení všech potřebných instalačních úkonů můžeme začít programovat. Otevřeme stránku- RoBIOS Library Functions, jež obsahuje předprogramované rutiny systému RoBIOS (což je operační systém EyeBota) a to na adrese:

http://robotics.ee.uwa.edu.au/eyebot/doc/API/library.html

-také jsou obsaženy v offline EyeDoc souboru pod kolonkou RoBIOS. Předprogramované funkce jsou vcelku přehledně seřazeny do skupin podle příslušnosti k dané periferii a u každé je stručný popis, co daná funkce umí, jaké potřebuje vstupy a jaký má výstup. Například sekce: Camera, sekce v níž se nacházejí funkce potřebné pro práci s kamerou a obsahuje funkci pro načtení obrazu z kamery, vše vypadá takto:

```
int CAMGetFrame (image *buf);
    Input:      (buf) a pointer to a grey scale image
    Output:     NONE
    Semantics:  Read an image size 62x82 from grey scale camera.
                Return 8 bit gray values 0 (black) .. 255 (white)
```

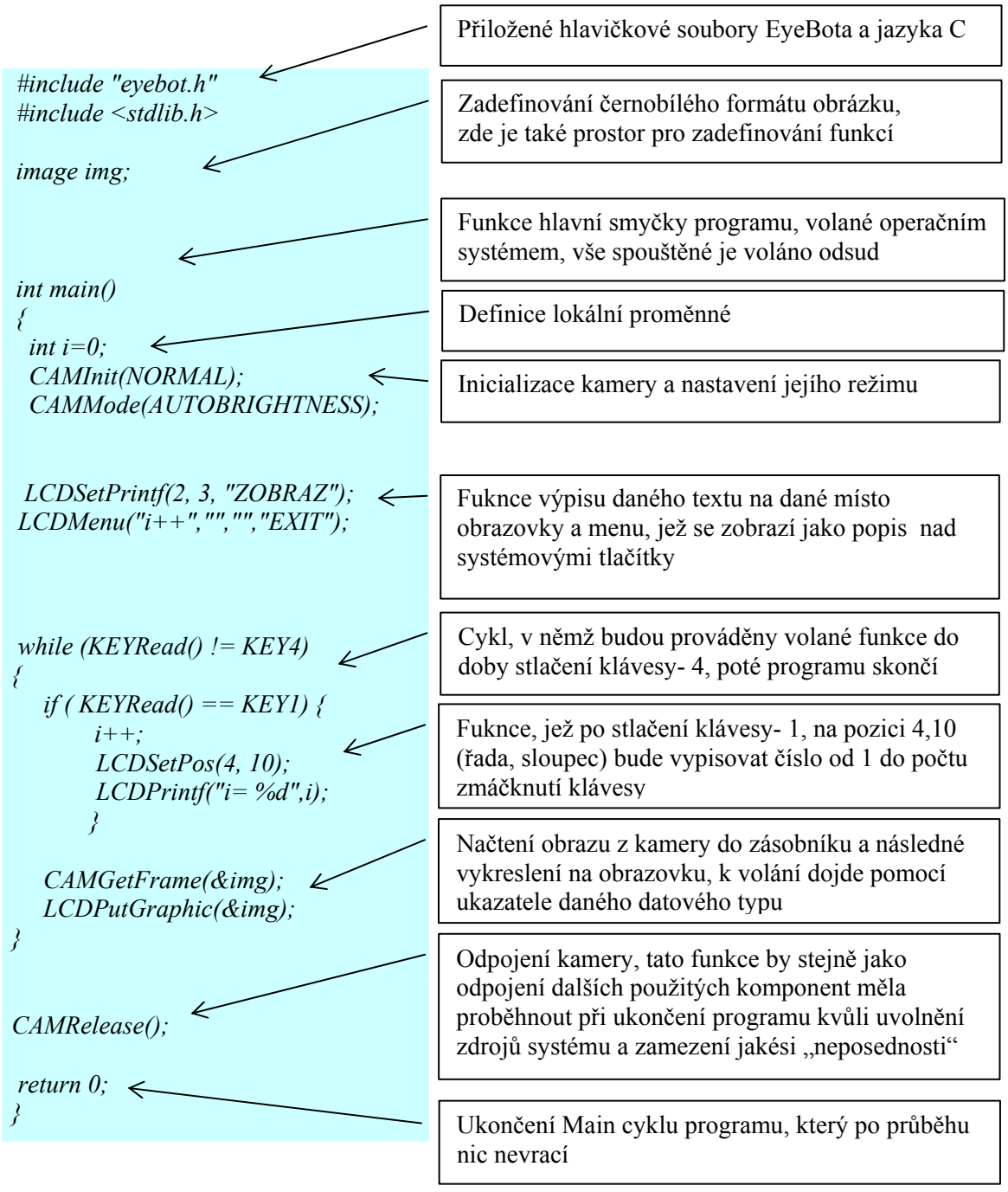
Funkce:

- *int*- zadefinování proměnné
- *CAMGetFrame*- samotný název funkce
- *image *buf*- pointer (ukazatel) na datový typ- image (v eyebot.h nadefinován jako černo-bílý formát o rozměru 62x82 pixelů s 8mi bitovými hodnotami)

Popis Funkce:

- *input*- vstupu do funkce
- *output*- výstup z funkce
- *semantics*- samotný popis, také příklad implementace v programu

Program musí také dodržovat jistou strukturu, pro ilustraci přiložený jednoduchý program, na kterém bude demonstrováno, jak vše asi vypadá v praxi:



Struktura je tedy stejná jako běžně v jazyce C. Doporučuje se nadefinovat funkce do prostoru mimo hlavní smyčku- kvůli přehlednosti a volání funkcí. Také se vyhnout používání globálních proměnných- hlavně v případě kdy jim přidělujeme danou hodnotu a to z důvodu, že by jim tato hodnota mohla být přidělena pouze při startu programu a pak při běhu dané funkce již ne. Proto je dobré proměnné spíše definovat lokálně a v případě potřeby globálního využívaní proměnné její hodnotu odkazovat jako návratovou hodnotu z dané funkce (př.: pomocí příkazu- return, na konci funkce). Také je dobré vyhnout se použití slova „start“ u

definice globální proměnné, jelikož je tento výraz rezervován pro potřeby systému a v programu by tedy nemusela daná proměnná fungovat.

2.2.2 KOMPILACE VYTVOŘENÝCH PROGRAMŮ

Zkompilování programu pro potřeby bota obstarává již zmíněný RoBIOS Prompt. Tento program je obdobou DOS-ovské příkazové řádky a tedy i pohyb v něm a některé funkce jsou totožné. Záleží na uživateli, kam si ukládá své vytvořené programy, ale kvůli kompilaci musí v Prompt cestu navést právě do této své složky. Implicitně startuje Prompt ve složce: Vas_disk:\EyeBot\examples (pozn.: Vas_disk označuje disk, kam jste program RoBIOS nainstalovali).

Pohyb po složkách je, jak jsem se již zmínil, totožný s pohybem v příkazové řádce, tím pádem například: ve složce- examples mám svou složku s názvem- moje. Po spuštění Prompt napíši příkaz: **cd moje**
příkaz otevře tuto složku a nastaví ji za aktuální, pro cestu zpět použiji: **cd ..**
když jsem již v nějaké složce a chci pouze zkontrolovat obsah, mohu dát příkaz, jenž mi vypíše, co se zde nachází: **dir**, popřípadě stromově: **tree**
(pozn.: vše ostatní jako např. tvorba podložek je výhodnější obstarat ze struktury Windows)

Ke kompilaci vytvořeného programu v jazyce C (zde označen jako mujsoubor.c) slouží následující příkaz:

gcc68 mujsoubor.c –o mujsoubor.hex

Při takto použité syntaxi je volán překladač gcc68 pro jazyk C, načte "**mujsoubor.c"** a vytvoří "**mujsoubor.hex**" (pozn.: jedná se o S-record soubor, má speciální formát řetězce v ASCII znacích a je určen pro procesory Motorola). Parametr "**–o**" je použit z důvodu, aby zkompiloval "**.c**" soubor spolu s připojenými hlavičkovými soubory do jednoho "**.hex**" souboru. Můžeme také zkompilovat pomocí parametru "**–c**" v jednotlivé objekty a ty později pomocí příkazu "**Makefile**" spojit v jeden soubor pro přenos do EyeBota (pozn.: tímto způsobem jsem se nezabýval a je popsán na www stránkách EyeBota). Další možné řešení je zkompilovat pouze příkazem:

gcc68 mujsoubor.c

Při takto použitém příkazu se vytvoří soubor- a.out, jež je také možné poté nahrát do EyeBota.

Programy vytvořené v jazyce C++ (přípona ".cpp") a Assembleru (přípona ".s") se kompilují obdobně, až na rozdíl volání jiného překladače. V případě C++ kompilujeme pomocí volání: **g++68**, v případě ASM souboru voláme: **gas68**.

Namísto používání nainstalovaného kompilátoru je taktéž možnost použít online kompilátor na adrese: http://roboti.cs.siue.edu/eyebot/68compiler/.

2.2.3 ODESÍLÁNÍ PROGRAMŮ DO EYEBOTA

Jakmile jsou programy vytvořeny a zkompilovány, můžeme přistoupit k nahrávání do EyeBota, což je proces zahrnující více fází:

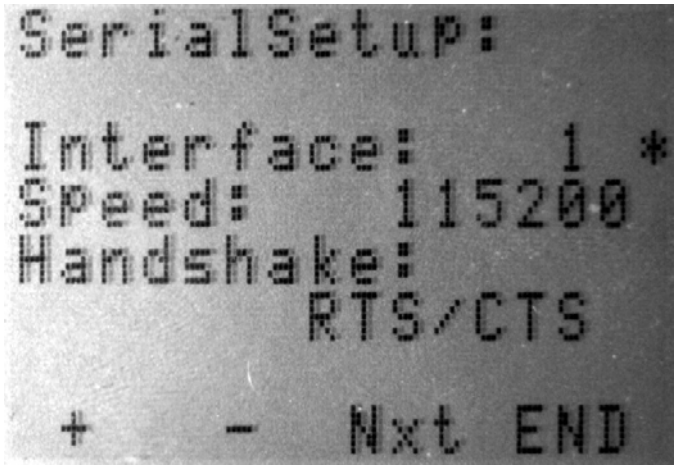
- Program lze nahrát propojením bota se sériovým portem vašeho počítače (tzv. COM port). V případě užití novějšího počítače kde již COM port není přítomen lze použít redukci, například USB to RS232, u níž však může být obsluha nepatrně odlišná.
- Po připojení nastavíme v systému Windows ve správci zařízení u položky COM a LPT u daného COM portu následující hodnoty:

  baud=115200, datové bity=8, parita=žádná, počet stop bitů=1, řízení toku=off

  Stejné hodnoty je potřeba nastavit také v botovi, kde dané nalezneme v oddíle- "**Hrd**", dále "**Set**", "**Ser**" a v něm:

  interface=1 (pro připojení do levé horní zdířky- serial port 1),
  speed=115200,
  handshake=RTS/CTS

  (pozn.: u aktuálně vybrané se zobrazuje symbol "*", pohyb se uskuteční stisknutím tlačítka "**Nxt**", "+" a "– " slouží ke zvyšování, respektive k snižování hodnoty)



Obr. 2.2.3a: Nastavení sériového portu

- Dále se již můžeme pustit do samotného přesunu souborů- v programu RoBIOS Prompt použijeme (samozřejmě po nastavení složky se zkompilovaným programem, za aktuální):

  dl a.out
  dl mujsoubor.hex
  (nebo také: **dl transhex a.out**, v případě že bot nebude brát daný nehex soubor)

V případě že počítač disponuje pouze replikovaným COM portem, může dojít k problému s přenosem, a to tak že při odesílání je kontaktován COM1 port a ten nemusí být nalezen. Existuje však další postup, který je možný použít také v případě správné funkce, ale pouze k ulehčení přenosu a to následující:
- otevřeme poznámkový blok a zkopírujeme do něj následující příkaz:

```
@echo off
mode com7: baud=115200 parity=n data=8 stop=1 to=off xon=off odsr=off
octs=on dtr=off rts=hs idsr=off > nul
copy /b %1 com7: > nul
```

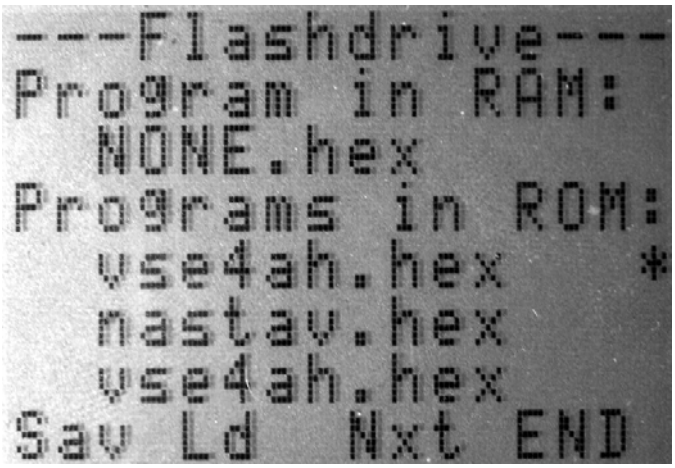
- com7 je zde jen pro příklad, na místo čísla 7 musí být napsáno číslo, na jakém se COM hlásí ve správci zařízení v použitém počítači
- soubor uložíme pod libovolným názvem s příponou- "**.bat**"

Nyní již máme vytvořenou funkci k odeslání, pomocí které stačí pouze přenesením a puštěním zkompilovaného "**.hex**" souboru na tuto ikonu zahájit přenos.

- Kliknutím na příkaz "**Ld**" v EyeBotovi dojde k přenosu programu. Příkazem "**Run**" jej můžeme spustit a nechat konat naprogramované.

  Také můžeme program uložit do pevné paměti ROM:

  - po přijetí programu klikneme na "**ROM**", dále tlačítkem "**Nxt**" vybereme jaký již uložený program chceme přepsat a dáme "**Sav**", přičemž musíme potvrdit tlačítkem "**Yes**". Při příštím spuštění EyeBota můžeme daný program opět pomocí "**Nxt**" v paměti ROM vybrat, zmáčkneme "**Ld**", ze sekce ROM vyskočíme pomocí tlačítka "**End**" a pomocí již zmíněného "**Run**" program opět spustíme.



obr. 2.2.3b: Sekce ROM

2.3 ROBIOS A HARDWARE DEFINITION TABLE (HDT)

RoBIOS a HDT jsou stavební pilíře této platformy. RoBIOS- alias Robot Basic I/O System je operační systém, jež se po softwarové stránce stará o součinnost všech systémů pod vedením procesoru. Stará se také o vizualizaci a převedení signálů od zařízení tak, aby byly viditelné pro nás a naopak naše programy a jejich povely uvádí v praxi a činnost v daných zařízeních. HDT je naopak „pouze" program, běžící pod RoBIOSem, obsahuje informace o připojených zařízeních (senzory, serva, motory, ale i startovací znělku a zobrazené informace), jejich seznam a informace jak je ovládat. Při spuštění tohoto programu máme také přístupné jakési testovací jednoduché programy pro ověření činnosti připojených zařízení.

Jak RoBIOS, tak HDT je možné uživatelem upravit a poté nahrát do EyeBota. V případě RoBIOSu se tato záležitost spíše nedoporučuje a náleží osobám znalým programování v assembleru, znalým mikroprocesorové techniky a tím pádem je dobré program nahrávat pouze v případě poškození v paměti, či při uvedení novější verze. Nahrávání probíhá stejně jako nahrávání uživatelských programů a systém sám rozpozná co mu vlastně posíláte (pozn.: soubor se však nutně musí jmenovat: robios.hex). Další způsob nahrávání je pomocí „background debuggeru" (tzv. BDM) pomocí kabelu k tomu určenému- tohoto způsobu se užívá v případě prvního nahrávání či při poškození stávající instalace, kdy není možnost

spustit systém a program nahrát standardně. Uvedená procedura je poté spuštěna přímo z uživatelského počítače. Postup této procedury je popsána na stránkách EyeBota a RoBIOS je možné stáhnout ze FTP serveru stránek nebo je uložen na vašem disku ve složce EyeBota, aktuální verze se nazývá- **rob65f.hex** (aktuální v roce 2009, F- pro daný typ kamery):

http://robotics.ee.uwa.edu.au/eyebot/ftp/ROBIOS/hex/

Vas_disk:\EyeBot\robios\hex

Ve věci HDT je již situace jiná. Několik verzí je uloženo ve složce, či na www adrese:

http://robotics.ee.uwa.edu.au/eyebot/ftp/ROBIOS/hdtdata/

Vas_disk:\EyeBot\robios\hdtdata

přičemž přítomná "**hdt-M5.c**" je určená právě pro verzi M5, která byla použita v této práci. Jelikož se jedná o soubor v jazyce C, je jej též možné i stejným způsobem upravovat. Stejně jako jsme vytvářeli programy za pomoci rutin RoBIOS, tak i zde můžeme vytvářet za pomoci rutin, jenž jsou vcelku podrobně popsané a vysvětlené na webové i offline stránce EyeBota, pod záložkou HDT. Odkaz na webovou stránku je zde:

http://robotics.ee.uwa.edu.au/eyebot/doc/API/HDT.txt

Programování poté probíhá obdobně jako u uživatelského programu, na rozdíl od něj však zde voláme direktivy:

```
#include "hdt.h"
#include "hdt_sem.h"
```

Oba dva soubory se nacházejí již ve zmiňované složce include, kde máme možnost nahlédnout dovnitř, v případě nejasností struktury nějaké funkce, či v případě potřeby úpravy (pozn.: hdt.h obsahuje vcelku podrobný popis struktury funkcí). Pro ilustraci ukázka s popsanou syntaxí, jak je v HDT zadefinován enkoder:

```
quad_type decoder0 = {0, 3, 2, MOTOR_LEFT, 1234, 2.34};
```

- *quad_type decoder0* - zadefinování typu a názvu daného prvku
- *0* - hodnota, jež udává maximální verzi ovladače, se kterou je tento zápis kompatibilní (novější verze by kupříkladu chtěla více informací, ty by avšak nebyly dostupné)
- *3, 2*- dvě hodnoty, udávající ke kterému TPU (= timer procesor unit- užívá se v některých mikrokontrolerech k ušetření výpočtového času procesoru v jednodušších aplikacích jako je generování časového průběhu signálu) kanálu je daný dvoukanálový enkoder připojen (kanál, jež čte PWM signál přicházející z enkoderu, potřeba dvou kanálů je za účelem zjištění směru pohybu- zda vpřed, či vzad), jejich vzájemnou výměnou dojde k čítání hodnot opačným směrem- například když se vozidlo pohybuje vpřed ale hodnoty jdou do záporna
- *MOTOR_LEFT* - informuje nás k jakému motoru enkoder přísluší
- *1234*- hodnota udávající počet „kliků" enkoderu na 1 metr uražené vzdálenosti. Enkoder je vyroben s určitým počtem „kliků" na otočku, tím pádem závisí na jakých kolech, pásech či podobném se náš robot pohybuje a je nutné spočítat, či pomoci programu změřit, kolik skutečně udává „kliků" na jeden metr dráhy. Tato hodnota je důležitá pro funkci VW-driving.
- *2.34*- maximální rychlost v [m/s], jež jsme si definovali, kterou se může náš robot pohybovat

Vytvořený program uložíme stejně jako každý jiný program v jazyce c, při kompilaci je však nutné tento HDT soubor zkompilovat se jménem začínajícím na- "**hdt**" a s příponou- "**.hex**" (např.: hdt_vuz.hex). Kompilace probíhá obdobným způsobem jako kompilace vlastních programů, ale s rozdílem volání odlišného překladače a to- **gcchdt**, celé např.:

```
gcchdt hdt_vuz.c –o hdt_vuz.hex
```

Následně jej již můžeme odeslat EyeBotovi: stejně jako vlastní program myší uchopíme zkompilovaný "**.hex**" soubor a přeneseme na ikonu, jež jsme vytvořili pro odesílání programů. Hdt soubor přijmeme a po rozpoznání botem nám je nabídnuto přepsání stávajícího, což musíme potvrdit stisknutím klávesy "**YES"**. Dále po vyzvání zmačkneme tlačítko- "**reset**" a poté můžeme v sekci **HDT** zkontrolovat že se jedná o námi upravený.

3 ÚLOHA ŘÍZENÉHO VOZIDLA

V této kapitole se budeme zabývat samotným technickým a programovým provedením úlohy bakalářské práce. Budou uvedeny části zdrojových kódů, potřebných pro řízení a funkci systému, spolu s vysvětlením, proč je dané uspořádání tak, jak prezentováno. Na závěr dojde k rozboru vozidla i po stránce konstrukční, jelikož i tato část je nedílnou součástí práce.

3.1 ÚPRAVA HDT

HDT jakožto soupis všech zařízení je většinou nutné upravit „na míru“ každé aplikace. Z důvodu připojení motoru, serva a enkodérů a jejich osobitých nastavení byla úprava nezbytná. Programování HDT vycházelo z předprogramovaných funkcí, jež jsou popsány v záložce HDT na již zmíněné internetové stránce EyeBota, či v EyeDoc souboru. V této sekci dojde k podrobnějšímu rozboru pro tuto úlohu upraveného souboru- **hdt_vuz.c**, jež se nachází na přiloženém cd v sekci- programy/vozidlo/hdt, upravená verze vychází z originální **hdt-M5.c**.

o Servo:
Servo je součást, jež celkem s vysokou přesností provádí rotační pohyb o zadaný úhel. V zásadě bychom jej mohli přirovnat ke krokovému motoru, avšak servo většinou není schopné rotace stále dokola, ale jen v určitém úhlovém rozsahu. V dalším pohybu jej omezují „dorazy“ a vše je kontrolováno pomocí uvnitř zabudovaného potenciometru a čipu. Vyrábí se v rozličném množství velikostí a rozsahů otáčení a právě rozsah otáčení je důležitá věc, kterou je třeba brát na zřetel. Ve zpracované úloze byl, z důvodu omezeného rozsahu tyče řízení, kladen požadavek maximálního úhlu natočení +/- 30˚, což je nutné nastavit jelikož možnosti serva jsou značně větší. Při zapnutí EyeBota totiž dochází k rázu a skoku serva do maximální krajní polohy, v případě většího rozsahu seva, nežli je pohybová možnost tyče řízení, dojde k poničení tyče či serva. EyeBotem je servo ovládáno pomocí TPU kanálu, přes nějž je posílán pulzně modulovaný signál (PWM), jehož délkou v Hertzech (PWM pracuje s určitou ovládací frekvencí) určujeme otáčení na jednu či druhou stranu o daný počet kroků. Následující část zdrojového kódu je použita z již zmíněného souboru **hdt_vuz.c** a týká se objasnění daného natavení serv. Z důvodu použití motorů s enkodéry, které sdílejí TPU kanály se servy 0-5, je nutné užití serva 6 a výše!

```
 * SERVOs */
servo_type servo7  = {2,  7, TIMER2, 20000, 1100, 1900};    // mnou použité a upravené
servo_type servo8  = {2,  8, TIMER2, 20000, 900, 2100};
```

- *servo_type servo7*- zadefinování typu a název daného prvku
- *2*- hodnota, jež udává maximální verzi ovladače, se kterou je tento zápis kompatibilní (novější verze by kupříkladu chtěla víc informací, ty by avšak nebyly dostupné)
- *7*- pořadové číslo TPU kanálu, kanály 0-5 jsou již využity motory (EyeBot ovládá celkem 16 TPU kanálů, z čehož serva obsazují 14)
- *TIMER2*- serva jsou připojena k časovači 2, taktéž se nabízí možnost připojení k časovači 1 (*TIMER1*), ale ten má menší frekvenční rozsah, tím pádem méně diskrétních kroků k polohování a ovládání by bylo méně přesné, nežli s doporučeným časovačem 2

- *20000*- doporučená hodnota, označující délku periody v mikrosekundách (μs), kterou potřebuje servo k přesnému ovládání, při kratší či delší hodnotě bude docházet například ke zpoždění odezvy serva na řídící signál
- *1100, 1900*- start a stop hodnota PWM signálu. Jejich rozdíl je procesorem rozdělen na 256 kroků (hodnoty 0-255), jež nám dovolují změnu pozice od jednoho kraje do druhého s úhlovou změnou o 256x jeden krok. V mém případě jsem u serva 7 zvýšil start hodnotu a zároveň snížil stop hodnotu, za účelem snížení úhlového rozsahu, který poté naprosto vyhovoval možnému rozsahu tyče řízení. Středová poloha serva je na hodnotě 1500 μs, zvětšováním nebo zmenšováním této hodnoty dosáhneme otáčení serva ze střední polohy na jednu nebo druhou stranu. Serva je u EyeBota možné řídit signálem od 740-2140 μs, v závislosti na provedení.

Jak již bylo řečeno, u serv je nutné dbát na správné připojení v zájmu vyhnutí se konfliktů s dalšími zařízeními a také na rozsah. Snížení úhlového rozsahu lze provést zmenšením rozsahu hodnot PWM, zvýšení je však závislé na konstrukci použitého serva. Zmenšením rozsahu taktéž docílíme zvýšení přesnosti, jelikož zmenšenou oblast procesor stále rozděluje na stejný počet kroků.

o DC motory:

V konstrukci bylo použito dvou stejnosměrných motorů, zapojených ve funkci elektrického diferenciálu a řízených pomocí vω-driving (pozn.: diferenciál je nutný z důvodu různých poloměrů oblouků, jež vozidlo při zatáčení opisuje a v případě stejných rychlostí obou kol by nebylo schopné zatočit – kolo na větším poloměru se musí otáčet rychleji). Jedná se o ovládání pomocí nastavené přímé a obvodové rychlosti, za současné kontroly správného otáčení pomocí enkodérů a následných akčních zásahů kontroleru. Eyebot motorům poskytuje stejnosměrné napětí s amplitudou okolo 6,4V, rychlost otáčení je stejně jako u serv ovládána pomocí PWM signálu, jehož frekvence (rychlost zapínání a vypínání) určuje poté střední hodnotu napětí a tím i rychlost otáčení motorů. Následující část zdrojového kódu prezentuje nastavení motorů v upraveném hdt souboru.

```
/* DC motors */
/* Motor A = left, Motor B = right */
motor_type motorA = {3, 0, TIMER1, 8191, (void*)sim_porte, 0, 0, (BYTE*)0, 0};
motor_type motorB = {3, 1, TIMER1, 8191, (void*)sim_porte, 1, 1, (BYTE*)0, 1};
```

- *motor_type motorA* - zadefinování typu a názvu daného prvku- motor A nebo B
- *3*- hodnota, jež udává maximální verzi ovladače, se kterou je tento zápis kompatibilní (novější verze by kupříkladu chtěla víc informací, ty by avšak nebyly dostupné)
- *0*- pořadové číslo TPU kanálu
- *TIMER1*- kvůli generování PWM signálu i motory musí náležet jistému časovači, nabízí se dvě volby- Timer1 a 2, frekvence časovačů jsou závislé na aktuální frekvenci procesoru a pro dané časovače- Timer1: 4MHz – 8MHz, Timer2: 512kHz – 1MHz.
- *8191*- číslovka, jež udává délku PWM periody v Hz, taktéž jsou doporučeny intervaly hodnot nejdelších period, aby nebyly závislé na aktuálním kmitočtu procesoru- pro Timer1: 256Hz - 40kHz, pro Timer2: 32Hz - 5kHz. Nastavení příliš pomalé frekvence vede ke „škubání“ motoru, vysoká na druhou stranu zatěžuje procesor a taktéž roste ztrátový výkon.

- *(void*)sim_porte, 0, 0*- označení s příslušnými dvěmi čísly odkazuje adresy pinů k ovladači, v EyeBotovi je užit H-můstek (vysvětlení viz internet), první pin je pro jízdu vpřed a druhý pro jízdu vzad - ovladači tedy udávají směr otáčení (zda se v odezvě na kladný signál mají točit na jednu či druhou stranu), při zanechání stejných hodnot u obou motorů a použití podvozku se dvěmi koly by se totiž vozidlo místo rozjetí vpřed točilo dokola, při udání ovladače 3 by měly být hodnoty obou bitů u jednoho motorku stejné a směr rotace udává až poslední číslo v celé závorce nastavení motorů- 0 označuje jednu stranu a naproti tomu jednička směr invertuje
- *BYTE* conv_table*- odkaz k bitové převodní tabulce pro motory- motory se totiž mohou lišit a tím pádem je v některých aplikacích nutná synchronizace pomocí této převodní tabulky, za účelem stejné rychlosti rotace

Nastavení velikosti PWM periody k zajištění správné funkce je kapitola sama pro sebe a podle rad odborníků bylo k nastavení použito metody pokus-omyl. Při chybné hodnotě motory téměř nejevily snahu o pohyb, přičemž proudový odběr motorů převyšoval povolené limity, v čehož důsledku EyeBot „zamrzal“ a hrozilo zvýšení vstupního proudu nad pojistkou hlídané 3A.

o Enkodéry a funkce elektrického diferenciálu:
Elektrický diferenciál potřebuje dostávat k správné činnosti zpětnou odezvu o aktuálním natáčení kol. Tuto funkci zde zajišťují optické enkodéry, jako odpověď na rotaci vysílají „pilovitý“ signál s vlastnostmi závislými na konstrukci a to především na počtu „kliků“ na otočku. Vysvětlení struktury nastavení enkodéru bylo již popsáno v kapitole- 2.3 ROBIOS A HARDWARE DEFINITION TABLE (HDT), tím pádem bude přiloženo pouze nastavení bez dalšího vysvětlení funkce jednotlivých členů:

```
/* Enkodery */
quad_type  encoderA = {0, 3, 2, MOTOR_LEFT,  14000, 0.45};
quad_type  encoderB = {0, 4, 5, MOTOR_RIGHT, 14000, 0.45};
```

Nastavení diferenciálu je vcelku podstatná záležitost ke správné funkci vozidla, zvláště v případě, kdy je vozidlo pomocí diferenciálu nejen „tlačeno“ vpřed, ale kdy je jím také, bez závislosti na jiném aktivním prvku k zatáčení, ovládáno směrově. Z konstrukčního hlediska bylo pro aplikaci na, vpředu řiditelném a vzadu hnaném, vozidle vhodnější Ackermannské diferenciální řízení. Nicméně tato funkce, ač prezentována, ve strukturách EyeBota dosud nefunguje, a proto došlo k použití obyčejného dvoukolového diferenciálu. Následuje nastavení:

```
/* Diferencial */
vw_type drive = {0, DIFFERENTIAL_DRIVE, {QUAD_LEFT, QUAD_RIGHT, 0.135}};
```

- *vw_type drive*- definice struktury
- *0* - hodnota odkazující na verzi kompatibilního ovladače
- *DIFFERENTIAL_DRIVE* - způsob funkce daného diferenčního řízení (další možné jsou omni, tricycle, či ackermann- který ale doposud nefunguje a jedná se pouze o ukázku implementace)
- *QUAD_LEFT, QUAD_RIGHT* - připojené snímače otáček (enkodér levý a pravý)

- *0.135*- informační hodnota o rozchodu kol diferenciálu a to za účelem možnosti spočtení středu otáčení, tím pádem i rozdílu úhlových rychlostí kol při zatáčení

3.2 PROGRAM ŘÍZENÍ

Tato celá podkapitola bude zaměřena na rozbor a vysvětlení funkce programu, sepsaného k ovládání sestrojeného vozidla, podle podmínek zadaných k realizaci této bakalářské práce. Vše zde rozebrané je zkopírováno ze zdrojového kódu - **vse5a.c**, který se nachází na přiloženém CD v sekci: programy/vozidlo/funkcni.

3.2.1 OVLÁDÁNÍ KAMERY

K EyeBotovi se kamera připojí přímo, nebo lze použít 32pinový nástavec, jež bylo nutno pro aplikaci na vozidle vyrobit (plochý kabel a příslušné dvouřadé konektory- je však nutné dohlédnout aby pin1 na EyeBotovi doléhal na pin1 kamery a dále).

Kamera snímá obraz oblasti před jedoucím vozidlem a hlavním cílem bylo provést rozbor obrazu, na kterém bylo třeba pomocí matematických úprav a funkcí zpracování obrazu detekovat tmavou naváděcí linii. Z přijatého barevného obrazu byl vytvořen binární bitový obraz, obsahující pouze černou a bílou. Poté dojde k rozboru jedné řady z kamery (1D úloha), kolmé ke směru pohybu, nalezení vodící linie, spočítání těžiště linie a odchylky od středové polohy. K objasnění je přiložena část zdrojového kódu funkce zpracování obrazu, dalšími nutnostmi jako je definice datových typů obrazu či inicializace kamery a spuštění funkce rozboru se zde nebudeme zabývat. Zdrojový kód je zde zapsán kurzívou a stojí před komentářem, jež jej osvětluje a je psán obyčejným písmem a umístěn v závorkách pro komentáře- /* */. Algoritmus zpracování obrazu je inspirován jinými programy volně šířenými na internetu.

```
// funkce rozboru obrazu
void BinImg(colimage img)
{
  int row, col;
  int sum = 0, cogrow = 40, x_spos = 0, x_lpos = 0, i=0;
 // nadefinování proměnných, vynulování hodnot a určení linie rozboru na řadu 40

  for(row = 1; row < imagerows-1; row++)
       for(col = 1; col < imagecolumns-1; col++)
        {
         sum = (img[row][col][0] + img[row][col][1] + img[row][col][2]) / 3;
```

/*
Přijatý obraz si můžeme představit jako matici hodnot o row=počtu řad a col=počtu sloupců (v použité funkci zpracovává kamera výsek o velikosti 62*82pixelů). Předchozí dva cykly tedy postupně načítají všechny hodnoty z matice obrazu. Snímač kamery je založen na RGB barevném modelu s filtrem typu Bayer, kde se nachází na jeden snímaný čtverec po jednom detektoru od modré a červené barvy a dva detektory zelené, každý dosahuje hodnot od 0 do 255 (min - max). Následně všechny hodnoty sečteme, vydělíme třemi a podle předem

nastavené prahové hodnoty je výsledek přiřazen buď černé nebo bílé barvě (hodnotě 0 nebo 255) a tento výsledek program zapíše do proměnné- "sum".
*/

```
// vytvoření binárního obrazu
if(sum < 80)
```

/*
Podmínka udávající od jaké hodnoty bude přiřazena hodnota proměnné "sum" - černé či bílé barvě. Tato dělící hodnota je rovna 80 a při splnění podmínky budou všechny barevné čtverce s hodnotou sum pod 80 převedeny na černé.
*/

```
    {
     grey[row][col] = 0;

          if(row == cogrow && i == 0)
          {
               x_spos = col;
               i++;
          }

          else if(row == cogrow  && i != 0)
          {
               x_lpos = col;
               i++;
          }
    }
     else
          grey[row][col] = 255;
}
```

/*
Předchozí algoritmus provádí následující: přiřazení odpovídajících čtverců černé barvě, vyhodnocení podmínky že se jedná o hodnoty řady, kterou bereme v potaz při počítání těžiště obrazce (pojmenována jako cogrow), zároveň vyhodnocení podmínky: i=0, která informuje že se jedná o první černý čtverec ve vyhodnocované linii. Program poté do proměnné- "x_spos" zapíše souřadnici umístění tohoto prvního čtverce a zvýší hodnotu proměnné- "i", tím pádem udá systému informaci, že již byl při průchodu cyklu v této řadě první černý čtverec nalezen. Poté vyhodnocuje stále danou řadu, avšak hledá souřadnici posledního černého čtverce, umístěného z pohledu kamery nejvíce vpravo a při nalezení každého dalšího zvyšuje hodnotu proměnné- "i". Hodnota posledního detekovaného čtverce zůstane, až do dalšího rozboru a opětovného průchodu cyklu celou maticí, uložen pod hodnotou proměnné- "x_lpos". Ostatní hodnoty, které neprošli počáteční podmínkou pro dělící hodnotu, jsou přiděleny bílé barvě a tím pádem tedy máme obraz, skládající se pouze z bílých a černých čtverců a v dané vyhodnocované řadě známe umístění pravého a levého kraje tmavé vodící linie.
*/

```
// Zobrazeni binárního obrazu na LCD obrazovce EyeBota
LCDPutGraphic(&grey);
```

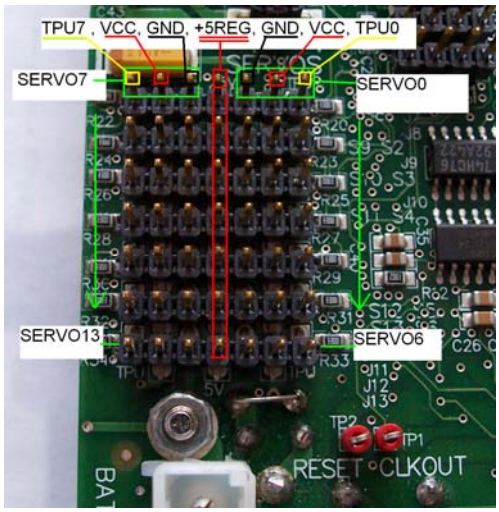
```
// Podmínka pro nalezení středového bodu
 if(i >= 2)
{
      c_point = (x_spos + x_lpos)/2;
      nic=0;
 }
else
 {
      OSWait(140);
      nic++;
 }
```

/*
Nalezení středového bodu, podmínka- i >= 2 je pro případ nalezení více než jednoho tmavého bodu. Poté dojde ke spočtení proměnné "c_point", což je střední pozice mezi pravým a levým krajem nalezené linie a bereme ji poté jako střed dané linie (= těžiště linie). Proměnná- "nic" je pouze informativní proměnná, která je zde vynulována a značí nalezení linie. Pokud je proměnná menší nežli dvě, je zřejmé že v rozboru obrazu nebyl nalezen žádný, podmínkám postačující obrazec. Systém tedy počká do následující akce 1,4 sekundy a poté zvýší informační proměnnou- "nic" o jedničku, což systém informuje o ztracení či „nevidění" vodící linie. Časová prodleva je nutná z důvodu nízké snímkovací frekvence a na druhou stranu několikaterého projetí cyklů rozboru, kdy by si při nulové prodlevě systém myslel že se zde linie střídavě nachází a nenachází. Informace o nenalezení je důležitá z hlediska navádění.
*/

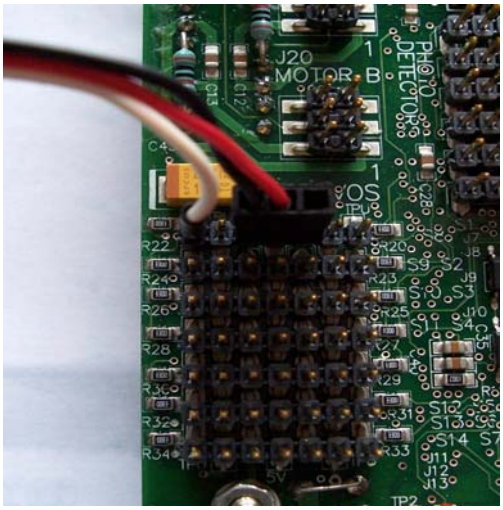
```
   // zaměřovač, jež na obrazovce označuje aktuální pozici středu vodící linie
   LCDArea(c_point-2, cogrow-2, c_point+2, cogrow+2, 2);
}
```

3.2.2 OVLÁDÁNÍ SERVA

K EyeBotovi se serva připojují pomocí standardizované koncovky, vyvstává však problém a to takový, že na kladném kolíku napájení serv EyeBota je jmenovitá hodnotu napájení (tedy tolik čím je Bot sám napájen, doporučeno 7,2VDC). Serva jsou však konstruována na 4,8-6VDC a tím pádem v mém případě došlo k poškození serva. Řešením, zdá se, je použití středního kolíku mezi dvěmi řadami pro připojení serv a na tomto kolíku by dle příručky mělo být regulovaných 5VDC, čili přesně hodnota jakou servo potřebuje. Dále bylo nutno servo připojit k pozici 6 a výše a to z již zmíněného důvodu použití motorů s enkodéry. Pro představu fotografie zapojení a popis pinů:



Obr. 3.2.2a: Rozložení pinů serv



Obr. 3.2.2b: Připojení serva k TPU, GND a +5V reg

Servo se stará o směrové natáčení předních kol a v programu pro ovládání jde v první řadě o snižování aktuální odchylky na nulu. To znamená, že v případě, kdy se vodící linie z pohledu kamery dostává doprava, kola se musí též natočit vpravo a aktivně tak odchylku vyregulovat. Následuje přiložená část funkce zatáčení z již zmíněného zdrojového souboru a stejně jako v případě kamery se budeme zabírat pouze stěžejní částí s vysvětlením:

```
// funkce řízení
void Steer(void)
{
  int ss=120;       //středová poloha na hodnotě 120

  //definice odchylky
  odch = stred - c_point;
```

```
/*
Hlavní úlohou je počítání odchylky od středového bodu (pixelu) kamery, definovaného na střed=41. Kamera totiž snímá 82 pixelů, tím pádem na hodnotě 41 se nachází střed snímané oblasti (2 krajní body jsou okraj, který nebereme v rozboru v potaz, tedy poté musí nastat omezení na 40 uvažovaných bodů). Program počítá odchylku jako definovaný střed – střed těžiště obrazce z funkce zpracování obrazu. Výpočet tedy např.: odch= 41-50 =>-9. V použité konvenci, kdy pixely kamery jdou zleva doprava od 0 do 82, získáme při záporné odchylce informaci že střed těžiště obrazce se nachází v tomto případě o 9 pixelů vpravo od středu obrazovky. Z toho důvodu potřebujeme natočení kol taktéž vpravo.
*/
```

```
  //Výpis informace na LCD obrazovku jaká je aktuální odchylka
  LCDSetPos(3, 6);
  LCDPrintf("od1: %d", odch);

  //Omezení hodnot odchylky na reálné hodnoty, kvůli omezení maximální výchylky serva
  if(odch >= 40)
  odch = 40;
  if(odch <= -40)
  odch = -40;
```

```
//Hlavní funkce zatáčení a řízení vozu, začíná couváním
if(nic!=0)
  {
  SERVOSet(serv, ss);                  //zatočí rovně
  VWSetSpeed(vwdrive, -0.025, 0);      //couvne rychlostí 0,025 m/s
  AUTone(12000, 50);                   //po dobu 50ms pouští zvuk o 12000Hz
  }
```

/*
Při splnění podmínky- proměnná "nic" je nenulová, nebyla linie nalezena a po uplyntí 1,4 sekundy od tohoto zjištění, aniž by se stalo jinak. Dojde ke spuštění funkce couvání, kdy je servo nastaveno na středovou polohu (hodnota ss=120) a vůz začne pomalu couvat. Ve chvíli opětovného nalezení linie je tato funkce přerušena, jelikož proměnná nic=0 je v nadřazeném cyklu a vozidlo se začne hýbat zamýšleným způsobem vpřed.
*/

```
// jinak vpravo či vlevo
else if(abs(odch) > 3)
        {
          servo_value=ss - odch*3;          //min: 120-odch*3=0, max: 120+odch*3=240
          SERVOSet(serv, servo_value);
          VWSetSpeed(vwdrive, 0.032, odch*0.003);       //w=v*tanfi/l,  fimax=0,523rad
        }
```

/*
V případě nalezení linie dochází k dvěma případům a to rovná jízda bez změn či zatáčení. Tato funkce je definována pro absolutní hodnotu odchylky větší než 3pixely a pokud je splněna, servo dostane povel zatáčet. K zatáčení dochází změnou hodnot ovládání serva od středu do krajní polohy a to: pro zatočení vlevo- 120→0, vpravo- 120→240. Tím pádem je nutné hodnotu odchylky od středové odečítat, abychom splnili požadavek kdy, při hodnotě odchylky rostoucí do záporného maxima potřebujeme hodnoty serva zvyšovat do kladného maxima. Hodnota odchylky je taktéž násobena, jelikož na rozdíl 40pixelů z kamery připadá rozdíl 120 hodnot pro servo. (pozn.: využitý rozsah serva 0-240 je pouze z důvodu vhodnějšího nastavení v této úloze, standardně je ovládáno v rozsahu 0-255)
*/

```
        // rovně
        else{
              SERVOSet(serv, ss);
              VWSetSpeed(vwdrive, 0.037, 0.00);
            }
}
```

/*
Tato poslední sekce je provedena při nalezení linie, stejně jako předchozí funkce, avšak absolutní hodnota odchylky je menší nežli 3 pixely. V tom případě je poté servu nastavena středová poloha a vozítko jede přímo rovně bez jakýchkoliv zásahů. Tuto funkci by bylo možné vynechat, avšak požadovaná přesnost je takto dostatečná a navíc je tímto způsobem šetřena životnost baterie a také serva. Následně je ukončen cyklus řízení a celá funkce.
*/

Stálé aktivní zásahy řízení poskytují systému tu výhodu, že řízení kol, přesnost umístění serva a celá část řízení může být o „pár řádů“ horší, než by se na první pohled zdálo. Při odchýlení od směru linie je totiž vozidlo ihned aktivně směřováno zpět a i zde tedy platí- čím horší jsou vlastnosti systému, tím lepší musí být řízení.

3.2.3 OVLÁDÁNÍ MOTORŮ S ENKODÉRY

Vzhledem k požadavku přesného navádění, užitím elektrického diferenciálu a tím i nutnosti dvou motorů s enkodéry ke kontrole polohy, došlo k větší složitosti konstrukce. Na druhou stranu však k zjednodušení ovládání a zvýšení přesnosti ovládání. Použití způsobu vω-řízení, což je řízení pohybu pomocí přímé rychlosti v a úhlové rychlosti ω, nám umožňuje nastavit požadovanou rychlost pro jízdu vpřed (kladná hodnota v), či vzad (záporná hodnota v). K tomuto účelu však musí být nastaveny správně enkodéry. Funkce ovládání jsou součástí funkce zatáčení zmíněné v kapitole ovládání serva, vložena tedy bude pouze samotná funkce s popisem. Enkodéry a motory jsou volány pomocí funkce současně, tím pádem s nimi a jejich hodnotami nemusíme vůbec pracovat. Při použití motorů dochází k situaci, jež pokládám za důležité zmínit: spuštěním programu, ve kterém se pracuje s motory a následným ukončením, taktéž i nutným „release“ příkazem (příkaz nutný k odinicializaci motorů), dochází i přesto při spuštění dalšího programu (a to i bez ovládání motorů) k otáčení kol. K této situaci dochází zřejmě kvůli nevyprázdnění systémového zásobníku a proto je doporučeno po skončení programu použít tlačítko- “**reset**”, poté lze již bez předchozích zmíněných problémů pracovat dále.

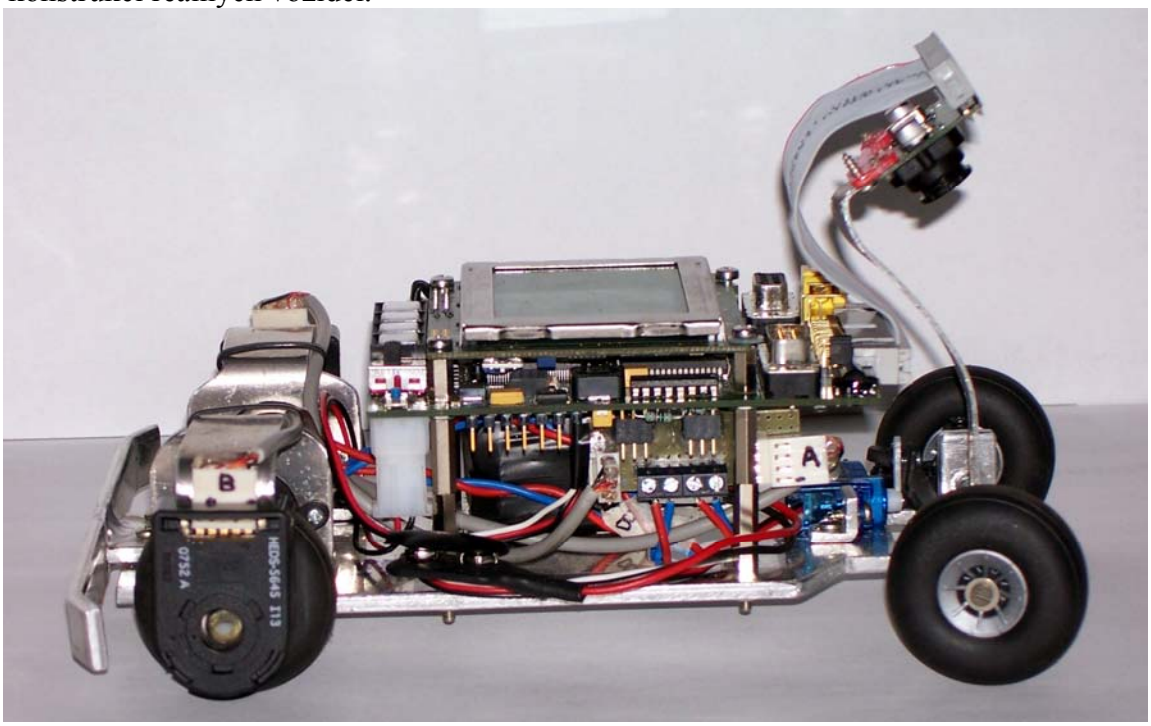
```
VWSetSpeed(vwdrive, 0.032, odch*0.003);
/*
Funkce nastavuje rychlost volané funkci vwdrive, což je funkce ovladače vω-řízení. Další hodnotou se nastaví přímá rychlost vpřed na hodnotu v=0,03m/s a jelikož je tato funkce vytažená přímo z funkce zatáčení, obsahuje také hodnotu úhlové rychlosti - ω. Stejně jako u serv potřebujeme zadní nápravou zatáčet, násobíme tedy hodnotu odchylky spočteným koeficientem. Při kladné hodnotě úhlové rychlosti zatáčí zadní náprava vlevo a při záporné vpravo, což je znaménkově přesně naopak nežli u serva a tím pádem bylo nutné pro daný směr i opačné znaménko.
*/
```

Při užití funkce vω-řízení je taktéž doporučeno nastavení přítomného PI regulátoru pro regulaci a udržování konstantní nastavené rychlosti. Při odpojení regulátoru za běhu pokračuje vůz poslední nastavenou rychlostí. Jednotlivé složky regulátoru je možné spočítat, ale ve většině případů plně postačuje doporučené nastavení. Dané doporučené nastavení funkce z programu vozidla:

```
VWStartControl(vwdrive, 7.0, 0.3, 7.0, 0.1);
/*
První dvě hodnoty udávají proporcionální a integrální konstantu pro regulaci přímé rychlosti, další dvě hodnoty jsou proporcionální a integrální složka pro regulaci obvodové rychlosti. Proporcionální bývá kolem 7 a integrální mezi 0 a 1
*/
```

3.3 TECHNICKÉ PROVEDENÍ

V zpracované úloze řízeného vozidla bylo nejen nutné zabývat se programováním kontroleru, ale taktéž sestavit vozidlo, které bude kontrolerem ovládáno. Vozidlo je postaveno na čtyřkolovém podvozku vlastní konstrukce, vývoj vycházel z vlastních zkušeností a s ohledem na konstrukci reálných vozidel.

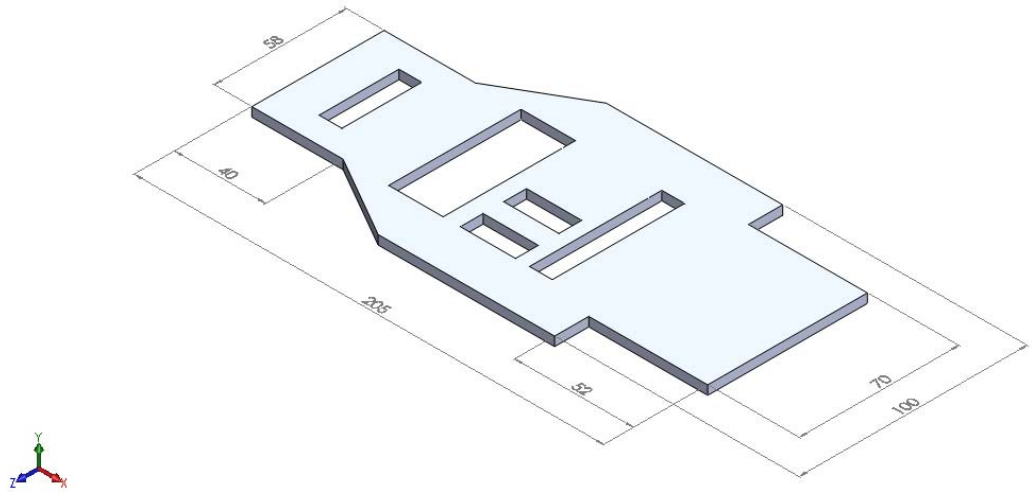


Obr. 3.3: Pohled na Vozidlo

3.3.1 KONSTRUKCE A UMÍSTĚNÍ JEDNOTLIVÝCH ČÁSTÍ

- Rám:

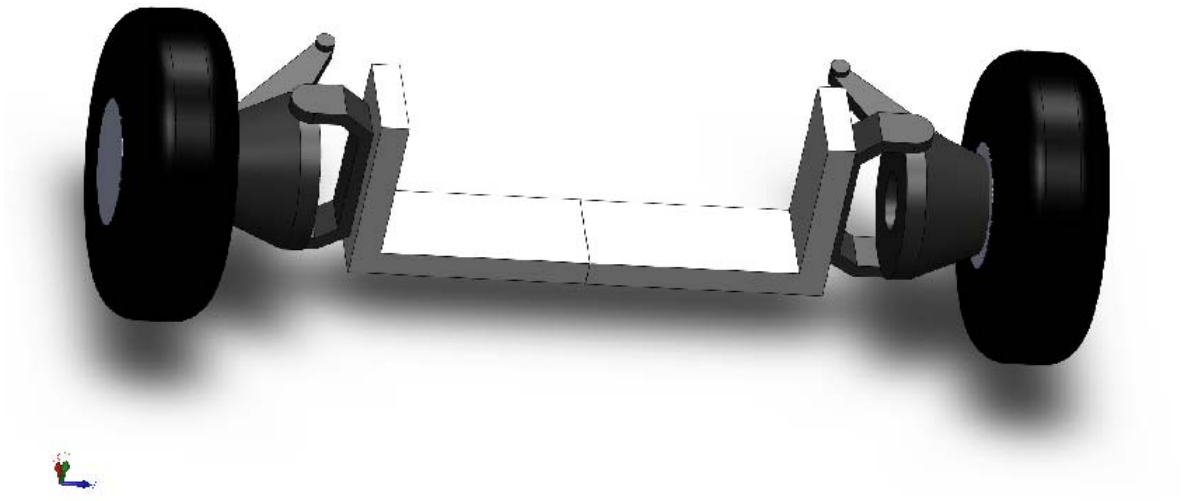
Vozidlo je postaveno na pevném rámu z jednoho 4mm tlustého hliníkového plechu, u nějž z důvodu odlehčení došlo na několika místech k odebrání materiálu. Hliník je použit z důvodu nižší hmotnosti a snadné opracovatelnosti oproti ostatním kovovým materiálům, avšak díky použité tloušťce je i dostatečně robustní a pevný.



Obr. 3.3.1a: Narýsovaná část rámu

- Přední náprava:

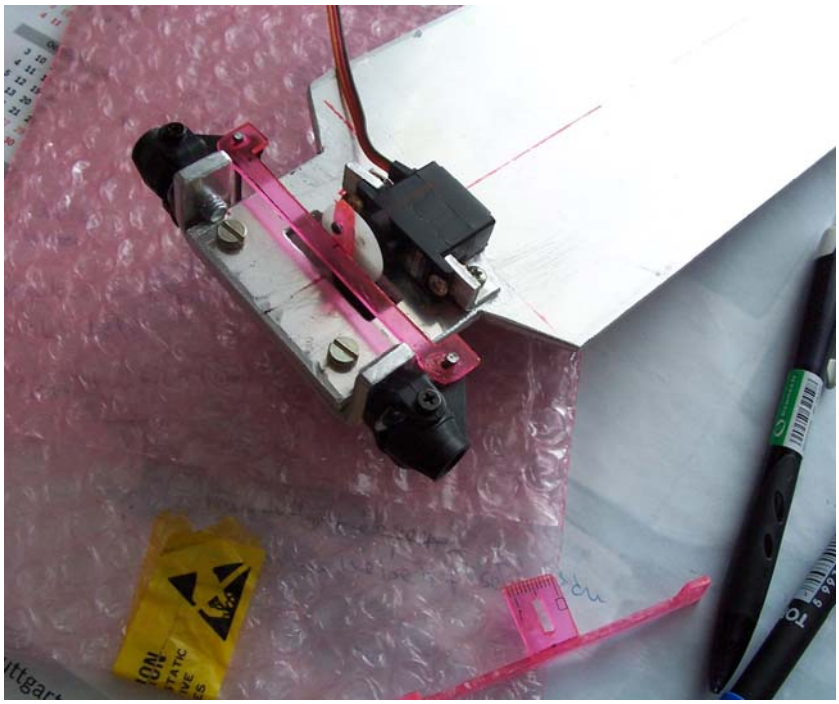
Základem přední řiditelné nápravy je neodpružené zavěšení kol, které bylo použito produkční, určené pro RC modely. Z důvodu potřeby větší přesnosti než dosažitelné ruční výrobou jako u ostatních dílů a vzhledem k použití lehkého, ale pevného plastu. Díly zavěšení kol, neboli těhlice, jsou poté přišroubovány k hliníkovému U-profilu a ten přišroubován k rámu. Tento způsob uchycení dovoluje posléze „doladit“ přesnou polohu kol.



Obr. 3.3.1b: Přední náprava

- Servo:

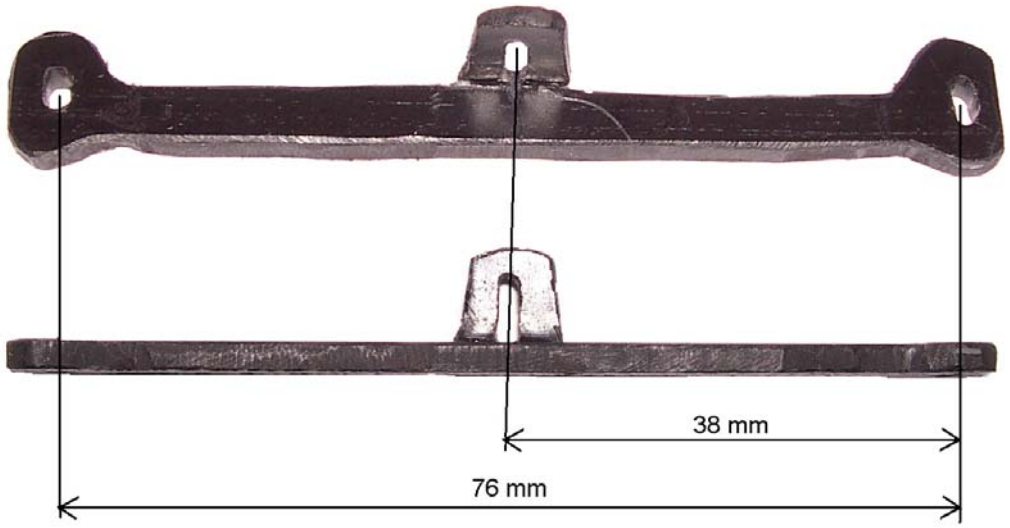
Servo je nutné uchytit blízko přední nápravy a s ohledem na posuv a rozměry tyče řízení. Přičemž je nutné dbát na umístění osy rotace výstupní hřídelky serva na středu rámu a tím pádem i středu mezi předními koly (v případě použití symetrické konstrukce tyče řízení).



Obr. 3.3.1c: Umístění serva na rámu vozidla (spolu s testovací verzí tyče řízení)

- Tyč řízení:

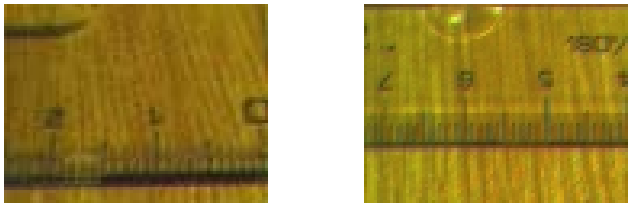
Poměrně důležitá součást, která převádí rotační pohyb serva v posuvný, nutný k natáčení předních kol kolem rejdové osy. Konstrukce dílu klade požadavek na co nejvyšší přesnost bez vůlí, které zhoršují chování podvozku jak při rovné jízdě, tak při zatáčení. Díl byl vyroben z pevného plastu ABS, s ohledem na kinematický rozbor ovládání natáčených kol. Výsledný maximální rozsah rotace serva byl upraven přímo podle pohybových možností tyče řízení, v závislosti na pohybu řízených kol.



Obr. 3.3.1d: Tyč řízení, verze V3

- Kamera:

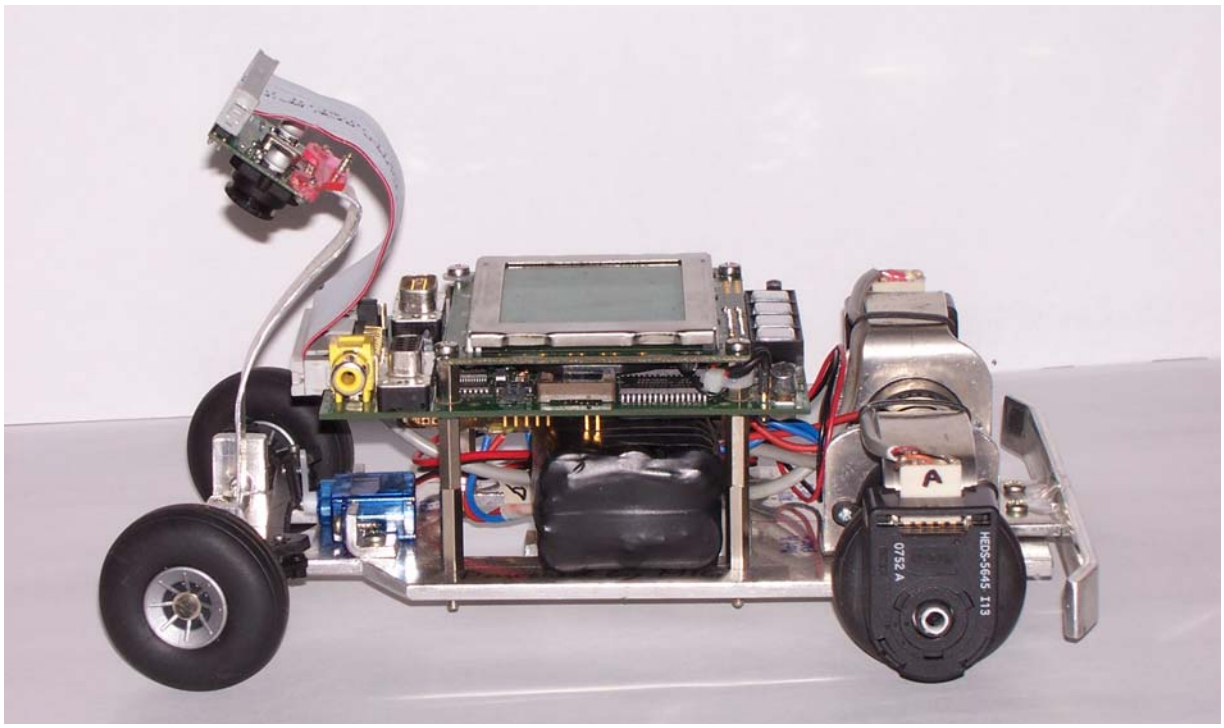
Kamera musí být umístěna s volným výhledem vpřed s co největším zorným úhlem. Vyhodnocovací oblast byla vzhledem k poloměru zatáčení nastavena zhruba 5cm před vozidlo. Při velmi blízkém bodu navádění by, kvůli nevelkému rozsahu zatáčení, vozidlo nebylo schopno pevně se držet bodu linie, při velmi vzdáleném opět vozidlo v podstatě nenásleduje linii. Další problém byl způsoben použitím čočky kamery s velkým přiblížením a při konstrukčně vhodnějším umístění kamery byla zorná část široká pouze 2,5cm, což je nedostatečné. Kamera tedy musela být umístěna „výše“ a dosáhlo se tak zvýšení zorné šíře na celých 3,2cm. Dále jsou pro porovnání přiloženy snímky z kamery EyeBota z obou poloh umístění (pozn.: snímky z EyeBota byly pořízeny pomocí programu EyeView, dostupného na FTP serveru stránek prof. Bräunla). Taktéž bylo nutno zaostřit čočku kamery na vyhodnocovanou vzdálenost- zaostřování je možné také provést za pomoci již zmíněného programu, který umožňuje téměř real-time náhled na aktuální průběh. K zaostření se nabízí použití více metod, tato je však nejnázornější a nejpřesnější (např.: lze zaostřovat pomocí analogového TV výstupu v součinnosti s propojením s televizí).



Obr. 3.3.1e: Vlevo se nachází snímek z původního umístění kamery, vpravo z konečného (snímky byly vzhledem k malé velikosti a nízkému rozlišení zvětšeny)

- Eyebot:

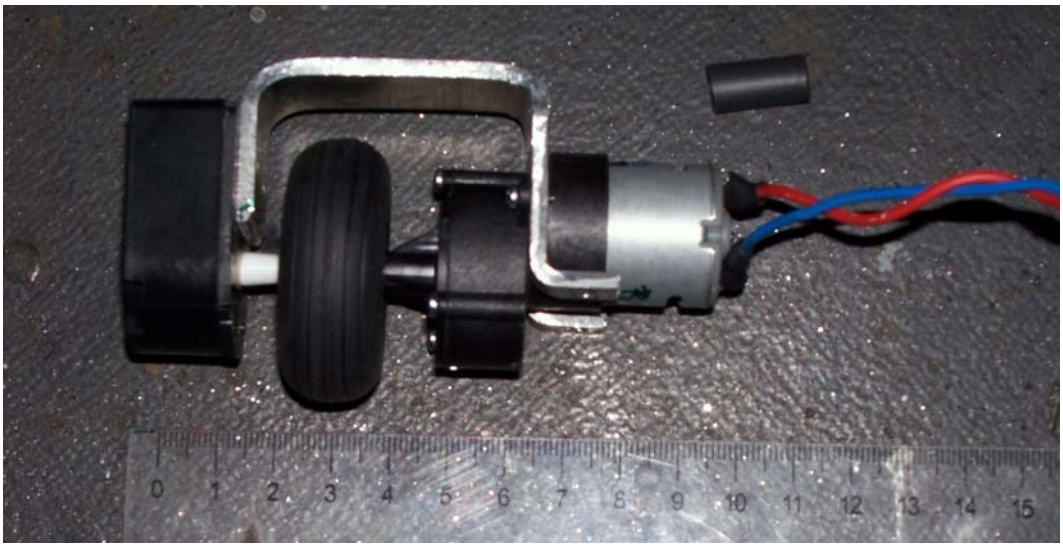
Rozměry vozidla jsou přizpůsobeny právě velikosti EyeBota. Vzhledem k připojení všech komponent kabely, je možné umístit EyeBot libovolně, nejvhodnější se však zdá umístění mezi motory a servem. Ležaté uchycení je z důvodu snadné manipulace s botem a přístupnosti k potřebným portům. Toto umístění taktéž dovoluje „schovat“ napájecí baterii do volného prostoru pod bota a tím pádem umístit největší zátěž do těžiště vozu.



Obr. 3.3.1f: Pohled na vozidlo s EyeBotem a baterií

- Motory a enkodéry:

Motory jsou umístěny na zadní nápravě. Jedná se o 2x MIG280, což jsou stejnosměrné kartáčové motory. Každý motor je trvale připojen na hnané kolo přes převodovku se stálým převodovým poměrem 5:1. Motory s převodovkami jsou k rámu vozidla uchyceny pomocí zkonstruovaných hliníkových držáků. K držákům jsou taktéž připevněny enkodéry, s přímým připojením na hnaná kola, ve snaze snímání skutečných otáček kol a tedy i přesné představy o pohybu vozidla. Pro připojení kabelů od motorů s enkodéry k EyeBotovi jsem použil nástavec od ing. M. Světlíka a to z důvodu že EyeBot disponuje standardizovanými konektory pro jiný druh motorů a bylo by nutné vyrábět redukci.

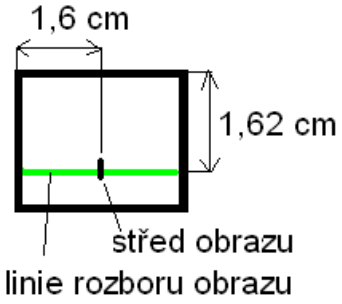


Obr. 3.3.1g: Motorová zástavba s převodovkou, kolem a enkodérem v držáku

3.3.2 VÝPOČTY A PARAMETRY VOZIDLA

- Snímková frekvence a maximální rychlost pohybu:
  V případě snímkové frekvence vyvstal problém s nedostatečným výkonem kontroleru. Při použití kamery se současným vω-řízením bylo výrobcem k funkčnosti doporučeno snímání 3 snímky/sekundu, což se po testech i s vyšší snímkovou frekvencí ukázalo opravdu vhodnější. Tím pádem bylo nutno určit maximální možnou rychlost, při které je soustava řiditelná:

  ➔ snímání max = 3 snímky/s



Obr. 3.3.2: Snímaná oblast

  ➔ maximální posun ke schopnosti řiditelnosti na jeden snímek = 1,62cm
  ➔ maximální rychlost: $v_{max} = 3 * 1{,}62 = 4{,}86\ cm/s = 0{,}0486\ m/s$ (1)

- Otáčky motoru:
  Při konstrukci je dobré vzít v potaz maximální otáčky motoru, maximální požadovanou rychlost vozidla a tomu úměrně vybrat rozměry kol a převodový poměr převodovky za účelem optimálních provozních otáček motoru. Pro použité motory jsou však k dispozici převodovky o maximálním převodovém poměru pouze 5:1, z vypočtených nízkých otáček však vychází, že by bylo vhodné použít převodovky alespoň s poměrem 20:1.

➔ maximální otáčky motoru na prázdno = 14000 s^{-1}
➔ maximální otáčky při požadavku z rovnice (1) :
- průměr kola: $\phi = 4{,}4\ cm$
- dráha kola na jednu otočku: $s = \frac{2 * \pi * \phi}{2} = 13{,}82\ cm = 0{,}1382\ m$ (2)

➔ počet otáček za sec, při maximální rychlostí z (1) vychází:
- otáčky kola: $o_k = \frac{v_{\max}}{s} = 0{,}3517\ s^{-1}$ (3)
- otáčky motoru: $o_m = o_k * 5 = 1{,}7583\ s^{-1}$ (4)
→ z výsledných velmi nízkých otáček motoru vyplývá potřeba většího převodového poměru, jak bylo zmíněno již v úvodu

- Rozlišení enkodéru:
Struktura HDT požaduje určení počtu „kliků“ enkodéru na uražení dráhy o délce 1 metr (systém potřebuje tuto informaci k výpočtu rychlosti pohybu, či uražené dráhy v závislosti na jemu známému počtu „kliků“). Tato konstanta je dána konstrukcí enkodéru (počet „kliků“ na otočku) a vlastnostmi převodu pohybu vozidla na otáčky enkodéru.

➔ enkodér je vyroben s konstantou 512 „kliků“ na otočku
➔ uchycení je přímo na kolech, tím pádem počítáme převod z rotace kola na enkodér:
- počet otáček kola = počtu otáček enkodéru, dráhu známe z rovnice (2)
- počet otáček na 1metr dráhy: $o = \frac{1}{s} = 7{,}2359\ ot$ (5)

➔ výsledný počet „kliků“ enkodéru: $k = o * 512 = 3704\ klik/m$ (6)

- výsledný počet kliků na uražený metr vzdálenosti lze poté ověřit experimentálně pomocí čítače v testovacích programech struktury HDT, naměřený výsledek se však liší od spočteného a to z důvodu sčítání obou hodnot dvoukanálového enkodéru, nebo z odlišného čítání signálu EyeBotem (například čítáním délky periody časově proměnného signálu z enkodéru), objasnění jsem však nikde nenalezl a do struktury je vhodnější zadat přímo tuto naměřenou konstantu.

- Nastavení rozdílu změny úhlové rychlosti
Pro potřeby diferenciálu a vω-řízení bylo nutno spočítat změnu úhlové rychlosti, při potřebě zatočení o požadovaný úhel, v součinnosti s natočením předních kol. Výpočtem jsem vycházel z konstrukce Ackermannova podvozku a použil jsem vzorec z knihy-Mechatronika [5].

➔ odchylka je definována pro: $abs(\varphi) = 0 - 30° = 0 - 0{,}523\ rad$ (7)
➔ výpočet úhlové rychlosti pro φmax:
$$\omega_{\max} = \frac{v * tg\varphi}{l} = \frac{0{,}03 * tg\,0{,}523}{0{,}16} = 0{,}1081\ rad/s \quad (8)$$
➔ tuto hodnotu však potřebujeme měnit stejně se změnou polohy serva, tím pádem maximální úhlovou rychlost zatáčení rozdělíme na 40 kroků změn odchylky:

$$\Delta\omega = \frac{\omega}{40} = 0{,}0027\ rad/s \tag{9}$$

- Parametry vozidla:

hmotnost bez baterie: 0,750 kg
rozměry: výška: 0,13 m
šířka: 0,20 m
délka: 0,22 m
rozvor náprav: 0,16 m
rozchod kol: vpředu: 0,125 m
vzadu: 0,135 m
pohon: 2x MIG 280
řízení: Robbe servo
napájení: 7,2 V
příkon: 21,6 W
výkon motorů: 2x6,4 W

4 ZÁVĚR

Ve své bakalářské práci jsem zpracoval téma: Realizace úlohy „sledování čáry“ na platformě EyeBot.
Seznámil jsem se se zásadami práce s touto platformou, programováním a použitím v praxi.
Sestrojil jsem 4kolový podvozek s přední řiditelnou a zadní hnanou nápravou, na kterém byla platforma umístěna. V jazyce C jsem napsal program, rozpoznávající tmavou linii na světlejším pozadí a kterou EyeBot následuje. V případě ztracení směru linie se EyeBot zastaví a couvá do doby, než linii opět nalezne.
Při práci na úloze jsem narazil na limitní možnosti EyeBota, také jako na nestandardní chování a spolu s výhodami a nedostatky se snažím tyto záležitosti ve své práci zmínit a popsat.

Seznam jednotek a zkratek:

A - jednotka SI proudu (Ampér)
A/D - analog to digital
abs(φ) - absolutní hodnota rozsahu definované odchylky
ANSI C/C++, C - programovací jazyková norma
ASM - programovací jazyk (Assembler)
BDM- background debugger
cm - jednotka vzdálenosti (cm=1/10m)
CPU - central processing unit, známý také jako procesor
Δω - změna úhlové rychlosti na 1/40 hodnoty odchylky
ϕ - průměr kola vozidla
FTP server- File Transfer Protocol server
HDT- hardware definition table -seznam definovaného hardwaru
k - výsledný počet „kliků“ enkodéru na 1 metr uražené vzdálenosti
kg - jednotka SI hmotnosti (kilogram)
LCD- liquid crystal display
m - délková jednotka SI (metr)
m/s, rad/s - jednotka přímé a úhlové rychlosti
MHz - jednotka frekvence
o - počet otáček kola na 1metr dráhy
o_k - otáčky kola
o_m - otáčky motoru
ot - otáčka
PC - personal computer - počítač
PI - proporcionálně- integrační
PWM - Pulse-width modulation
rad - úhlová jednotka velikosti (radián)
RGB- red, green, blue barevný model snímače
RoBIOS - Robot Basic I/O System = operační systém EyeBota
s - dráha kola na jednu otočku
s^{-1} - jednotka otáček za sekundu
TPU- timer procesor unit
V - jednotka SI elektrického napětí (Volt)
v - označení přímé rychlosti [m/s]
VDC - stejnosměrné napětí v jednotkách Volt
v_{max} - maximální možná rychlost pohybu vozidla
vω-řízení - způsob řízení pomocí nastavení přímé a úhlové rychlosti
W - jednotka výkonu (Watt)
ω - označení obvodové rychlosti [rad/s]
ω_{max} - maximální velikost úhlové rychlosti pro maximální výchylku φ
www - World Wide Web

Použité zdroje:

[1] BRÄUNL, Thomas. *Robotics & Automation Lab* [online]. Feb. 2008.
Dostupné z: < http://robotics.ee.uwa.edu.au/eyebot/ >.

[2] RŮŽIČKA, David. *Učíme se C* [online, pdf]. 15.08. 2000.
Dostupné z: < http://www.builder.cz/art/cpp/clanek1666892044.html >.

[3] SVĚTLÍK, Miroslav. *Využití platformy Eyebot pro řízení mobilních robotů* [pdf]. 2008.
VUT Brno

[4] SARSHAR, Sizarta. *EYEBOT PROGRAMMING* [pdf].
Dostupné z: < http://robotics.ee.uwa.edu.au/eyebot /theses>.

[5] GREPL, Robert a kolektiv. Mechatronika : vybrané problémy. 1. vyd. VUT Brno, 2008. 148 s. ISBN 978-80-214-3804-0.

[6] REINHOLDTSEN, Peter. *Eyebot image processing primitives* [pdf]. 17.01.2000.
Dostupné z: < http://robotics.ee.uwa.edu.au/eyebot /theses>.

[7] Kolektiv autorů. *Wikipedia* [online].
Dostupné z: < http://wikipedia.org/>.

[8] Kolektiv autorů. *RGB color model* [online]. April 2007.
Dostupné z: < http://en.wikipedia.org/wiki/Rgb#RGB_devices >.

[9] HYUNRYONG, Jung. *line-follow.c* [online]. 02.12.2002.
Dostupné z: < http://robotics.ee.uwa.edu.au/eyebot/examples-rob >.