Středoškolská technika 2017

Setkání a prezentace prací středoškolských studentů na ČVUT

Robot ASURO

Petr Scheubrein

Gymnázium Třebíč
Masarykovo nám. 9/116, Třebíč, PSČ 674 01

Obsah

Abstrakt

Tato práce se zabývá programováním robota ASURO za účelem vytvoření praktického prostředí pro další vývoj robota a prozkoumáním různých alternativ implementace jeho funkcionalit. Zahrnuje také příklad řešení finální soutěžní úlohy z kurzu robotiky popisující jak autorem naimplementovanou vítěznou sestavu, tak jiné možné postupy řešení, jež jsou potřebné pro ilustraci kompromisů a variability postupu při vývoji v krátkem čase bez mnoha systémových kapacit robota.

Klade si za cíl seznámit čtenáře se základním přístupem řešení takových problémů a zároveň ho vybavit řadou obecně užitečných algoritmů pro jeho budoucí bádání, či soutěžení.

Robot

Přestože se má práce zabývat především softwarem robota, je nezbytné uvést základní specifikace robota.

Robot je postaven ze stavebnice ASURO. Jedná se o set, který se skládá z neosazeného tištěného spoje, několika mechanických sučástek (ouzubená kolečka, osy kol etc.) a mnoha prostých a v případě ztráty, či poškození snadno nahraditelných elektronických součástek, pomineme-li dostupnost některých čipů a mikrospínačů. Krom toho je obsažen i pingpongový míček, který je v konstrukci robota používán jako kluzný podvozek, usb dongle hlavně pro přenos programu z počítače do robota přes infračervené rozhraní a CD, které poskytje zejména decentně z němčiny přeložený anglický manuál popisujicí stavbu robota a základy jeho programování.

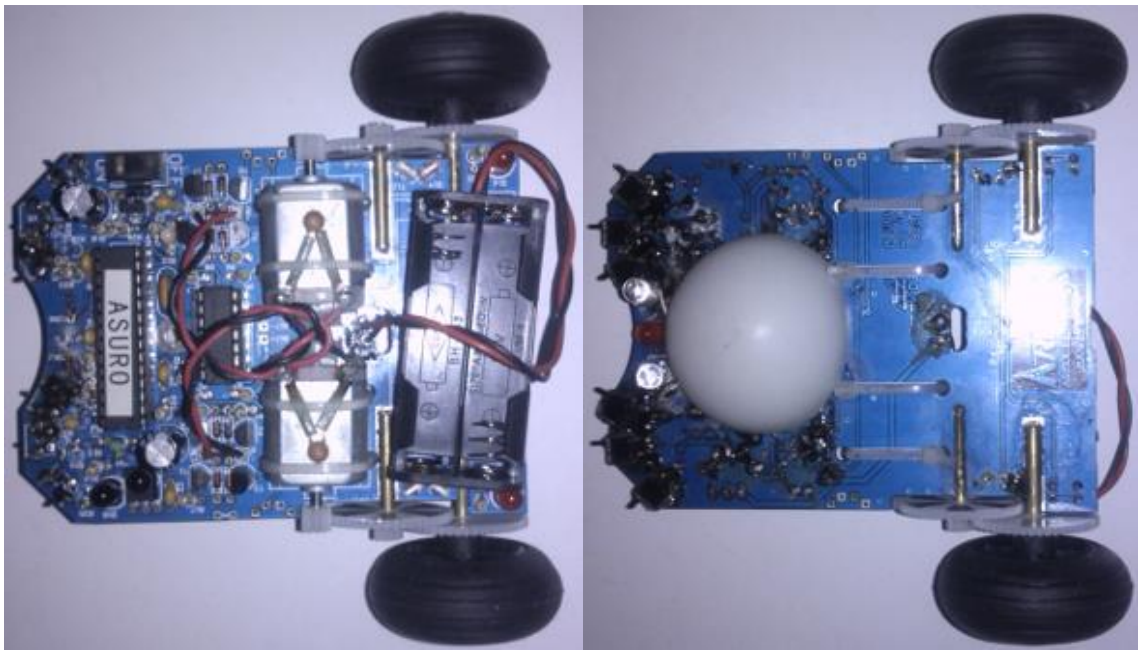
Řídící jednotkou robota je čip Atmega8 firmy Atmel poskytujicí 8KiB programové paměti (pamět do které jsou nahrány instrukce programu) z části zaplněné předinstalovaným bootloaderem, 1KiB SRAM paměti ve které je uložen celý stack (všechna volatilní paměť (smaže se po odpojení zdroje napětí), kterou může program standartně využívat) a 512b EEPROM non-volatilni paměti. Čip je buzen 8MHz oscilátorem a má k dispozici i některé další časovače, zejména 36kHz oscilátor, který je z programového prostředí relativně bezproblémově přístupný.

Celý robot je napájen čtyřmi AAA bateriemi, což dává robotovi dostatek energie na několik hodin operace v laboratorních podmínkách, ovšem při již při brzském poklesu napětí na betriích začne vykazovat nejdříve značné zpomalení motorů a později i neschopnost konzistentního startování procesoru.

Co do senzorů robota je připraveno 6 různě natočených mikrospínačů v přední části rbota, dva světelné senzory (fototranzistory) směřujicí naspod robota a dva odometrické sety sestávajicí ze světelné diody a fototranzistoru namířených na černobílé značky převodů pro určování rychlosti koleček.

Méně interaktivní součástky přímo ovladatelné pomoci přiložených knihoven jsou předně dvě červené LED diody na zadní straně robota, dva motory pohánějicí kolečka robota, barevná (zelená, oranžová, červená) indikační LED dioda a červená LED dioda umístěná mezi spodními fototranzistory, nejspíše jako konzistentní zdroj světla pro fototranzistory.

Nakonec je na robotovi infračervený vysílač a přijímač pro komunikaci s počítačem



Robot ASURO shora a zespod

Software

Robot komunikuje s počítačem pomocí infračerveného usb donglu, jehož dosah činí asi 1 metr. Dongle je nejčastěji připojen pomocí přiloženého prodlužovacího usb kabelu, kterým se dá kompenzovat nejen maximální vzdálenost přenosu, ale i fakt, že při rušení okolními zdroji infračerveného záření je třeba mít robota přimo u vysílače a stínit okolní šum.

Robot byl v mé péči vyvíjen pod operačním systémem GNU/Linux, což přinášelo některá úskalí, jež byla bezproblému vyvážena výslednou jednoduchostí používání.

Při instalaci potřebného software bych rád upozornil na potenciál poškození čipu FTDI při nainstalování špatných verzi driverů.

Všechen potřebný software je přiložen na CD nebo volně přístupný na internetu se zevrubným popisem v manuálu. Je záhodno podotknout, že díky fascinujicímu zpracovaní dodaných knihoven je často nutno přepsat několik drobností, aby bylo možné je vůbec zkompilovat a je doporučeno některé části přepsat i v zájmu rychlosti. Výraznější optimalizace však dosáhnete úpravou makefile. Především jde o kompilaci optimalizovanou na velikost výsledného programu (pro gcc kompilátor především argument `-Os`, ale případně i různé zamezování a podporování inline funkcí etc.) také proto, protože nahrávání kódu jinak trvá nepřiměřeně dlouho. V každém případě bych doporučil zběžné prohlédnutí zdrojového kódu knihovny, neboť se ukázal jako nejhodnotnější zdroj jinak mizerné dokumentace.

Komunikace

Snad nejdůležitější charakteristikou většiny programů je jejich výstup. Pro výstup je kromě funkcí pro rozsvěcení LED diod sloužicích jako jednoduchý indikátor, k dispozici funkce `SerWrite(char *data, unsigned char length)`, která očekává dva parametry: pointer na pole znaků (dále jen řetězec) potřebných k vypsání a délku takového pole.

Jak bylo poznamenáno na začátku této práce, chceme vytvořit prakticeké prostředí pro vývoj robota a to znamená nejen rychlé k použití, ale například také blbuvzdorné. Obou těchto vlastností můžeme dosáhnout zabalením funkce `SerWrite()` do funkce vlastní, dejme tomu `SerPrintf(char *fmt, ...)`, která může nejen sama spočítat délku řetězce přičítáním při průchodu od prvího znaku po nulový, ale také kontrolovat, jestli není aktuální znak `%` a podle následujicího znaku (po vzoru `printf` především `s`, `d`, `%`) určit, dle jakého typu má

zpracovávat následující argument (z konstrukce '. . . ', jejíž popis je hrubě mimo rozsah této práce). Tyto informace pak může v rámci cyklu předávat funkci SerWrite a umožňovat tak přinejmenším pohodlný výstup programu.

U téměř každé zde popsané kosrukce je možné dále iterovat na její myšlence (formátování řetězců etc.), ale čtenář by měl zvážit omezené dispozice robota a potřebnost obdobných vylepšení. Z důvodu zaměření této práce a čiré neomezenosti kreativity čtenáře budou dále takové úpravy uvedeny nanejvýše jako poznámka nebo návrh.

Přerušení

Robot bude evidentně muset provádět pravidelná měření některých senzorů a ideálně při tom neblokovat běh programu (předně odometrie). K takovým záležitostem jsou použity takzvané hardwarové přerušení. Hardwarové přerušení je v podstatě procedura zavolaná při přivedení napětí na přiřazený pin procesoru. Nás nejvíce zajímají přerušení periodická – připojená na nějaký oscilátor. Tento typ přerušení se může vyskytovat třeba u funkce na měření času, nebo pozastavení programu (`sleep()`). Takové funkce knihovna skutečně obsahuje a při letmém průzkumu objevíme i onen vektor (funkci volanou při přerušení), který přičítá zlomky milisekund měřící dobu pozastavení programu. S proměnnou držicí takový čas je neradno si zahrávat, protože tak docela nevíme jestli není použita i jinde v programu (ona není, ale ji obsahující funkce `Sleep()` je použita způsoby omezujicími manipulaci s touto proměnnou), ale není problém si do přerušení přidat jednoduchou volatilní proměnnou inkrementovanou při každém přerušení a vlajku, dejme tomu `ms_flag`, nastavenou kdykoliv hodnota této proměnné přesáhne milisekundu (v tuto chvíli je předchozí vlajka vynulována).

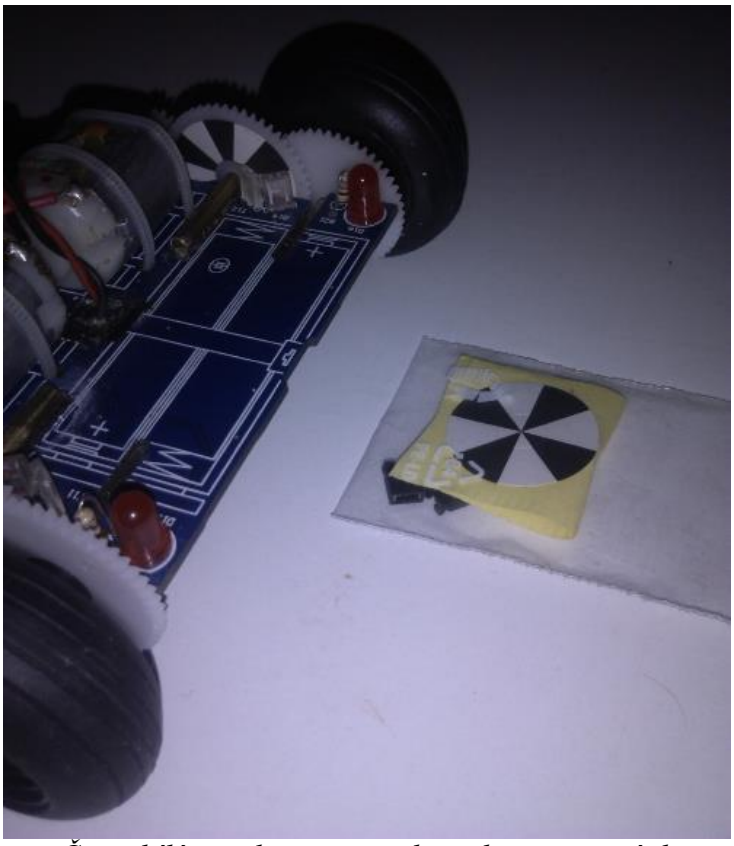
Samozdřejmě, že bychom mohli do přerušení vepsat mnohem komplexnější funkcionality, nebo přímo všechny asynchronní a pravidelné úlohy, avšak je obecně uznávanou praxí, že přerušení by mělo být co nejkratší, mimo jiné proto, že je v tomto případě voláno 72000 krát za sekundu a ubíralo by tedy značnou část procesorového času.

Jízda

S takovou výbavou se můžeme dostat k základům. Robot umí počítat milisekundy a vypisovat výstup, takže vyvíjení a opravování programu by měla být hračka. Přesto všechno se robot doposud nehnul z místa. Na scénu nastupují funkce `MotorSpeed()` a `MotorDir()`, kde první znich přijímá dvě osmibitové hodnoty typu `unsigned char` (podstatné také pokud z důvodu optimalizace používáte datové typy s pevnou šířkou z hlavičkového souboru `unistd.h`) přímo úměrně vyjadřujicí napětí proudicí do levého a pravého motoru a druhá z nich přijímá dvě makra ze setu FWD – vpřed, RWD – vzad, BREAK – brzda, FREE – volné otáčení, značící chování jednotlivých motorů.

Zaznamenejme však zmínku napětí, nikoliv rychlosti. Nastavíme-li oba argumenty funkce `MotorSpeed()` na stejnou hodnotu, je bez záruky, ba dokonce velmi nepravěpodobné, že se motory roztočí stejnou rychlostí, roztočí-li se vůbec. Tento fakt je velmi nepříjemný, pokud chceme po robotovi například rovnou jízdu.

Robot je ovšem vybaven odometrickým systémem, který jen čeká na to až jej někdo naimplementuje. Odometrické senzory vrací světlost povrchu černobílých značek na kolečku převodu, tvořící 12 nebo 8 výsečí po obvodu celého kolečka. Díky tomu můžeme počítat úhel o který se jednotlivá kolečka pohnula a v reakci na to upravovat napětí motorů. To vše jen pokud dokážeme rozpoznat rozdíl mezi bílou a černou výsečí. Musíme totiž kompenzovat kvalitu senzoru, okolní osvětlení a obecně šum, který se může měnit v průběhu celé jízdy a přitom spolehlivě určit hranici světlosti mezi černou a bílou, spolu se spolehlivým počítáním přestupů této hranice.



Černobílé značky na převodu v obou variantách

Jako přechod mezi černou a bílou barvou zdá se býti vhodný průměr těchto dvou hodnot, tedy hodnota ležící přímo mezi těmito hodnotami. Protože se budou barvy před senzorem periodicky střídat, je možně prostě průměrovat pravidelná čtení ze senzoru (perioda čtení musí být menší než doba přechodu převodu mezi bílou a černou při maximální rychlosti 1 ms je takřka ideální hodnota). Vzhledem k dynamické povaze odchylky je potřeba průměrovat jen nedávno nasbíraná měření, čímž se dostáváme k výpočetně optimalizované variantě klouzavého průměru, která byla použita. V takovém algoritmu bude místo cyklického bufferu, nebo jiných, paměťové místo zabírajicích, deterministických struktur použita aproximace založená na reprezentaci všech dosavadních měření aktuálním průměrem. Taková metoda by byla nevhodná, kdyby byly předpokládány velké fluktuace ve výsledné hranici (ke správné hodnotě se tato metoda blíží asymptoticky). Při prvních měřeních je také vhodné počítat průměr pouze z doposud naměřených vzorků. Velikost průměrovaného vzorku by měl být v závislosti na stabilitě senzorů co nejmenší násobek periody přechodu mezi barvami při operační rychlosti robota (hodnota která musí být výpočetně náročně dynamicky určována), s přihlédnutím k zamezení přetékání a ztráty přesnosti v rámci výpočtu (například při použití floating point čísel, která nedoporučuji v žádném bodě vývoje pro takovýto hardware). Reálně postačí 32. Průměr je vhodné držet separátně pro každý motor.

Měření je třeba provádět v pravidelných intervalech. Tedy podle předchozích příkladů například zarovnáváním na celé milisekundy po výpočtu pomocí konstrukace:

```
while (ms_flag) {}
ms_flag = 1;
```

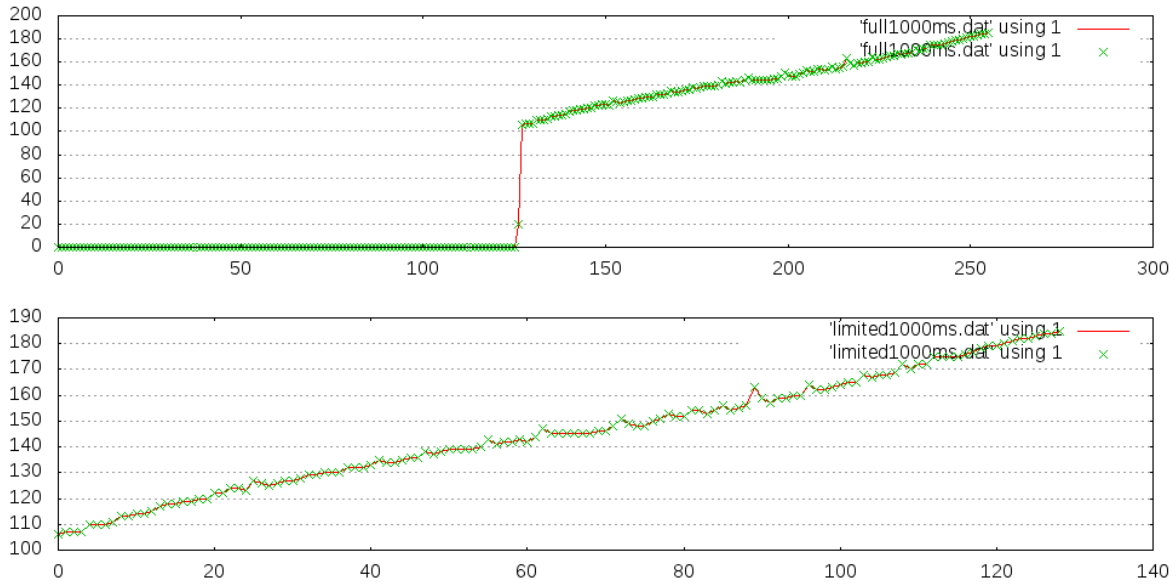
Pokud se vám stane, že hodnoty světlosti jsou velmi nestálé a způsobují opakované přičítání k počtu výsečí při jednom přechodu, existují dvě varianty řešení: zpomalení čtení

senzorů a tedy vyhlazení celé křivky grafu naměřené světlosti, nebo naopak zrychlení čtení a rozdělení hranice čtení ve dví. V takovém případě musíte kromě celkového průměru měřit i průměry maximálních a minimálních hodnot a hranici přechodu střídat mezi průměry celkového průměru a průměru maxim a průměrem celkového průměru a průměru minim tak aby se vždy dostala na hodnotu vyšši pokud má derivace měření, které způsobilo přechod, hodnotu zápornou, nebo nížší pokud ji má kladnou.

Nyní se nabízí otázka co budeme dělat s naměřenou chybou motorů. Možnosti jsem zůžil na dvě nejzajímavější: PID regulátor a metoda iterativního odhadu nápravy chyby. PID regulátor je obecně známý, proto jen krátce.

PID regulátor se skládá ze tří složek: **P**roporcionální, **I**ntegrační a **D**erivační. Proporcionální je přímo úměrná odchylce od ideálního stavu, integrační je rovna součtu těchto odchylek v čase (je možné ji limitovat pokud vám přetéká nebo uniká do extrémních hodnot) a derivační je rovna změně odchylky od posledního měření. Každá složka je vynásobena konstantou určující její dopad na výslednou hodnotu. Po sečtení všech složek dostáváme hodnotu reprezentující rozložení napětí na motorech. Kouzlo PID regulátoru spočívá v ladění oněch konstant, což je opět téma hrubě přesahující rozsah a téma této práce.

Zatímco PID regulátor je metoda univerzální, druhá metoda je přímo postavená kolem tohoto problému. Během běhu je měřen počet výsečí na jednotlivých motorech dokud neuběhne minimální interval a zároveň se rozdíl výsečí obou motorů nezvýší nad 2 (kvůli asynchronímu přechodu výsečí bude odchylka jedné výseče velmi častá). Alternativně stačí měřit v pravidelných intervalech. Nejlépe představitelné pokračování tohoto algoritmu (je vhodné zoptimalizovat eliminací floating point operací, pomocí násobení protějších hodnot na motorech v následujicím kroku) je přímá úměra, jejíž koeficient vyjadřuje sílu každého motoru jako počet výsečí / napětí. Slabší motor je nastaven na maximální napětí (pokud je požadována co nejrychlejší rovná jízda) a jeho předpokládaná rychlost je vypočtena vynásobením koeficientu tímto maximálním napětím. Napětí druhého motoru je potom nastaveno tak, aby se předpokládaná rychlost obou motorů shodovala (předpokládaná rychlost prvního motoru / koeficient druhého motoru). Možná si říkáte, že motory nereagují hned a určitě ne lineárně na změnu napětí. To bylo experimentálně prakticky vyvráceno (vizte grafy).



Oba grafy ukazují počet dvanáctin otoček kola (výsečí), o které se motor bez zátěže otočil za 1000 milisekund pri všech hodnotách napětí (0–255). Druhý graf je zaměžen na tu část grafu kde se motor otáčí

Chceme-li přidat k takto naimplementované rovné jízdě zatáčení, máme dvě možnosti: buďto připočteme k proměnné držící počet ujetých výsečí motoru ke jehož straně se chceme otáčet počet výsečí o které se chceme otočit za jednu periodu korekce kurzu (u PID regulátoru by to odpovídalo posunutí onoho ideálního stavu ke kterému regulátor směřuje), nebo jednoduše změníme směr otáčení jednoho kolečka robota pomocí funkce `MotrDir()` a ten se tak bude otáčet na místě.

EEPROM

Oba postupy jsou validní a při správném nastavení poměrně dobře odvádějí svoji práci. Především druhý algoritmus však prvních nekolik desítek milisekund nevyrovnává svoji odchylku. Kromě toho je potřeba nakalibrovat spodní fototranzistory robota, .neboť jakákoliv změna osvětlení v místnosti může robota poměrně dost zmýlit ve světlosti povrchu, při konstantních hodnotách. Bylo by proto hezké, mít možnost nějaká data uložená trvale (správné napětí motorů, světlost černého a bílého povrchu etc.) a moci je při startu robota načíst (**R**ead-**O**nly **M**emory). Tyto hodnoty se budou občas měnit, takže je potřeba aby byly vymazatelné (**E**rasable **R**ead-**O**nly **M**emory), nějakou metodou, která nevyžaduje příliš času a lidské interakce (**E**lectrically **E**rasable **R**ead-**O**nly **M**emory). K tomu slouží EEPROM paměť přístupná pomocí konstrukce `eeprom_update_byte()`, `eeprom_read_byte()` z avr knihovny `eeprom.h` a jejich alternativy pracujicí se slovy (2 byty). Jejich použití je poměrně evidentní a popsáno v dokumentaci knihovny.

Kalibrace

Otázka vyvstává s kalibračními hodnotami které chceme ukládat. Pokud chceme například detekovat černý a bílý povrch, pak můžeme třeba nechat robota po startu přejíždět po černobílém povrchu a potom využít shlukovou analýzu pro nalezení hranice mezi oběma hodnotami. Alternativně je možno použít funkci `PollSwitch()` vracejicí binární pole stisknutých tlačítek a pomocí funkce and a binární masky s příslušným tlačítkem zkontrolovat jestli je tlačítko při startu zmáčknuto, pokud ano, tak počkat na puštění tlačítka a při dalším stisku zaznamenat první hodnotu (kupříkladu černá) a pri posledním zaznamenat druhou hodnotu (bílá). Existuje jen málo důvodů proč preferovat první metodu.

Soutěžní úloha

Nyní se dostáváme k samotnému finále celého snažení: ukázková úloha. Náš úkol je prostý. Robot bude soutěžicím připraven na startovní čáře umístěné nekolik metrů před černobýlým terčem, skládajícím se z bíleho čtverce a předem daného počtu soustředných střídavě černých a bílých kruhů menších než čtverec. Úkolem soutěžícího je dostat se do středu tohoto terče, kde hodnotícím faktoren bude součet vzdáleností nejbližší části robota od středu v milimetrech po dvou jízdách. Robot bude mít před startem několik chvil na zkalibrování robota na stejné dráze na které pojede, ale nesmí do robota zadávat žádné konkrétní parametry umístění terče. Podlaha neměla být černá ani bílá.

Nejdříve se dostáváme k otázce nalezení terče. Odpověd na ni je velmi přímočará, neboť soutěžicí umisťuje robota na čáru sám, takže jej může nasměrovat přímo na terč. Potom už stačí jen jet dostatečně rovně (t.j. správně nastavit parametry předem posaných algoritmů), dokud nenarazí na terč.

Detekce terče je rovněž prostá. Robot (spodní senzory) je před startem zkalibrován na černou a bílou barvu terče. Potom jsou vymezeny hranice těchto barev tak, aby bylo možné rozpoznat mezi šedou (čtěte "středně světlou") barvou podlahy a ostrou barvou terče. Tato hranice může být určena experimentálne, rozumě jako 1/3 ve středu mezi černou a bílou, nebo jednoduše jako 2/4 mezi černou a bílou v závislosti na dostatku času při implementaci.

Odsud se na samotné soutěži ukázaly dvě slibné cesty řešení. Jedna dojede na první soustřený kruh a začne sledovat jeho obvod (pravděpodobně světlost na jednotlivých senzorech, která přímo úměrně ovládala napětí motoru, ale možnosti sledování čáry se dvěma senzory jsou nespočetné a mimo rozsah tohoto dokumentu) dokud se nesrovná s pomyslnou tečnou onoho kruhu v jeho pozici. Potom se otočí o 90° směrem ke středu terče a popojede předem známou vzdálenost obvodu krajního kruhu ke středu, počítá počet přechodů mezi barvami terče, nebo obojí v konjunkci pro zamezení případné katastrofy při špatném spočítání terčů. Tato metoda se zdá býti dobrá a na skutečné soutěži si vedla obstojně.

Druhá metoda zjistí kterým senzorem robot detekoval terč, načež začne upravovat rychlost motorů tak, aby se senzor který ještě nenajel na terč přibližoval rychleji a druhý senzor se příliš nehýbal, případně se pohyboval směrem od terče (čímž se z něho dostane a robot pod mírně opraveným úhlem popojede znovu rovně dopředu, opakujíc zarovnávací proces). Totéž potom dělá na každé kružnici, díký čemuž se stále přesněji zaměřuje na cíl. U středového kruhu popojede předem danou malou vzdálenost, aby se dostal celou svou masou nad střed terče. Tato metoda vyhrála soutěž za dosažení nevyšího možného skóre (pokud byl robot na středu, pak byla vzdálenost automaticky nula).

Závěr

Obecné postupy uvedené v této práci by mněli být vhodné pro využití v mnoha reálných situacích, nebo alespoň nasměrovat čtenáře správným směrem k prozkoumání jeho konkrétního problému. Samozdřejmě nebyly popsány a využity všechny kapacity robota, což nebylo ani cílem, ale byly popsány ukázkové metody naplnění jeho potenciálu. Doufám, že tento doukument pomůže všem z vás, kteří hledali úvod do světa robotiky a inspiroval vás ve vaších budoucích projektech.