VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

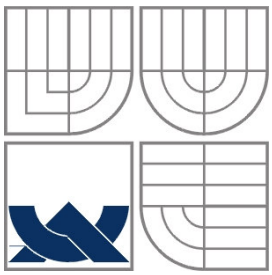
FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ROBOT PRO ROBOTOUR 2010

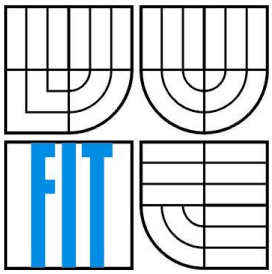
DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE BC. DAVID KUBÁT
AUTHOR

BRNO 2010



VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY



FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
ÚSTAV INTELIGENTNÍCH SYSTÉMŮ

FACULTY OF INFORMATION TECHNOLOGY
DEPARTMENT OF INTELLIGENT SYSTEMS

ROBOT PRO ROBOTOUR 2010

ROBOT FOR ROBOTOUR 2010

DIPLOMOVÁ PRÁCE
MASTER'S THESIS

AUTOR PRÁCE
AUTHOR

BC. DAVID KUBÁT

VEDOUCÍ PRÁCE
SUPERVISOR

ING. JAROSLAV ROZMAN

BRNO 2010

Abstrakt

Předmětem této diplomové práce bylo seznámit se se soutěží autonomních outdoor robotů Robotour a dále prostudovat možnosti a informace o robotovi vyvíjeném na ÚITS FIT VUT v Brně. Další část práce představovala studii různých plánovacích a lokalizačních algoritmů a návrh funkčního řešení, které by umožnilo robotovi účast v soutěži.

Konkrétním cílem práce bylo využít všech nabytých vědomostí k implementaci autonomního software pro start robota v soutěži v roce 2010. Součástí řešení je i několik souvisejících mechanických úprav. Vytvořený systém je funkčním řešením splňujícím pravidla soutěže, který je schopen startu v letošním ročníku.

Abstract

The objective of this master's thesis was to get acquainted with an autonomous outdoor robot competition – The Robotour and further to study the capabilities and information about a robot designed on the DITS FIT Brno VUT. Next part of the project was to study various planning and localization algorithms and to make a proposal of a functional solution, allowing the robot to participate in the competition.

To be specific, the goal of this work was to use the acquired knowledge for implementation of an antonomous software so the robot can start in the year 2010 race. The solution also included several related mechanical modifications. The final solution fulfills all the competiton rules and is capable of taking part in this year's plant.

Klíčová slova

Robotour, soutěž, robot, řízení, autonomní, outdoor, náklad, senzory, odometrie, sonar, kompas, mapa, JOSM, GPS, sériová linka, RS-232, zjištění polohy, orientace, cesta, plánování, UCS, navigace, překážky.

Keywords

Robotour, competition, robot, control, autonomous, outdoor, cargo, sensors, odometry, sonar, compass, map, JOSM, GPS, serial line, RS-232, localization, orientation, road, planning, UCS, navigation, obstacles.

Citace

Kubát David: Robot pro Robotour 2010, diplomová práce, Brno, FIT VUT v Brně, 2010

Robot pro Robotour 2010

Prohlášení

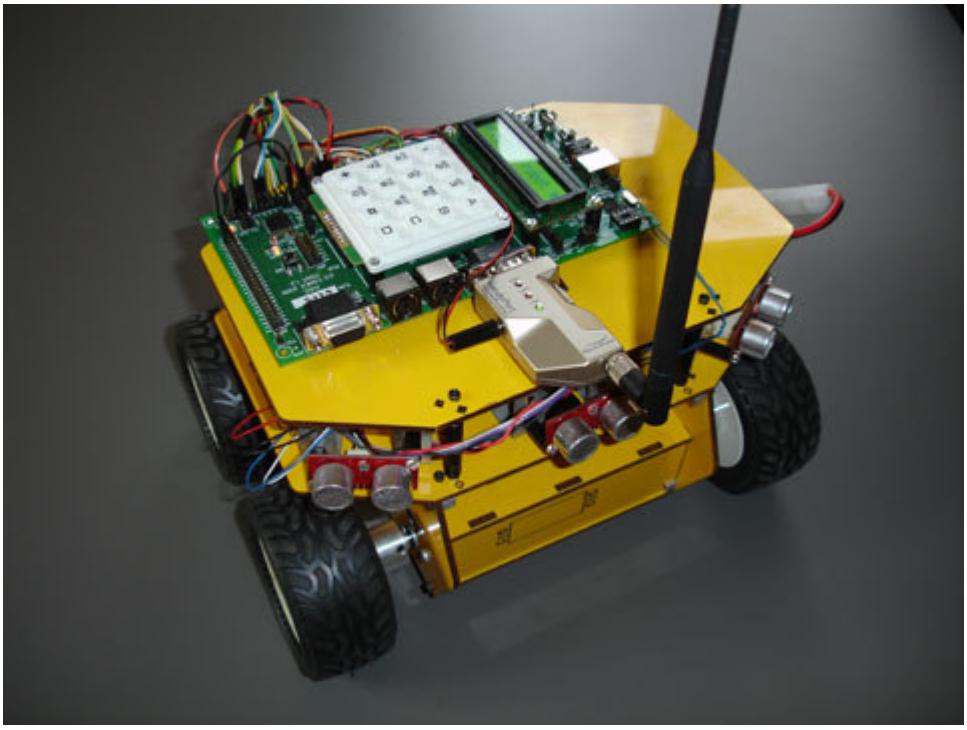
Prohlašuji, že jsem tuto diplomovou práci vypracoval samostatně pod vedením Ing. Jaroslava Rozmana.
Další informace mi poskytl Bc. Tomáš Novotný.
Uvedl jsem všechny literární prameny a publikace, ze kterých jsem čerpal.

……
David Kubát
26.5.2010

Poděkování

Zde bych rád poděkoval vedoucímu práce Ing. Jaroslavu Rozmanovi za poskytnuté podkladové materiály a odbornou pomoc během tvorby mé práce.
Druhé poděkování patří kolegovi Bc. Tomáši Novotnému za konzultace a pomoc při řešení technických problémů na robotovi.

Obsah



obrázek 1.1: robot pro Robotour 2009 [3]

1 Úvod

Komu by se nelíbilo, kdyby mu pro nákupy do obchodů jezdil robot, kdyby cestou do práce nemusel řídit, nebo kdyby se v jeho firmě přepravovaly obrovské náklady materiálu samy? Stejnou myšlenkou se zabýval i Martin Dlouhý v roce 2006, když se rozhodl uspořádat první ročník soutěže Robotour.

Tato soutěž mne inspirovala při výběru tématu pro moji diplomovou práci. Jejím cílem bude připravit vlastního robota tak, aby byl schopen účastnit se této soutěže v roce 2010. Celý postup a všechny kroky vedoucí k řešení budu postupně zaznamenávat do své zprávy. Záměrem také je, aby se i nezainteresovaný čtenář mohl dozvědět něco o této soutěži, o používaných technologiích a o možném řešení problému.

Úvodní kapitoly této práce se zabývají hlavně teoretickým úvodem do problematiky. První kapitola (kap. 2) se pokusí čtenáři nastínit atmosféru a pravidla soutěže Robotour a říct něco k její historii a vývoji. V kapitole 3 bude podrobněji popsán robot, se kterým se budu soutěže účastnit. Budou vysvětleny jeho možnosti, potenciál i nedostatky a budou navržena možná vylepšení robota. Kapitola 4 specifikuje formát použité mapy soutěžního prostředí. V kapitole 5 jsou potom vysvětleny základy zeměpisných souřadnic (GPS) a výpočtů mezi nimi.

Následující část se zabývá rozborem a popisem vlastního návrhu řešení - tedy specifikací konkrétních úkolů (kap. 6) a návrhem řešení jednotlivých problémů (kap. 7) z oblasti plánování, lokalizace a navigace.

V druhé polovině práce (kap. 8) je detailně popsána implementace všech funkčních modulů programu v hierarchické návaznosti, jak jsou za běhu postupně používány. Jednotlivé podkapitoly obsahují podrobný popis použitých funkcí, struktur a algoritmů včetně vysvětlení jejich činnosti. V kapitole jsou také zaznamenány problémy, které během implementace vznikly a popis jejich řešení. Kapitola 9 zachycuje postřehy získané během testování celého systému a také popisuje možnosti kontrolních výpisů a různých simulačních módů programu. Kapitola 10 potom shrnuje základní informace o ovládání aplikace a vstupních hodnotách. Poslední část práce (kap. 11) se zabývá rozborem mechanických a hardwarových úprav a vylepšení, kterými robot prošel.

Závěrem práce (kap. 12) je shrnut celý její průběh, její přínos a zhodnoceny dosažené výsledky. Jsou zde také uvedeny možnosti případných rozšíření a popsáno, kde by se dalo v práci pokračovat.

Tato diplomová práce navazuje na předchozí Semestrální projekt a její první část (do kap. 7) vychází z jeho obsahu.

2 Robotour 2010

Projekt Robotour [1] by se dal popsat jako „soutěž outdoor autonomních robotů“. Slovo **outdoor** tedy naznačuje, že se soutěž odehrává ve venkovním prostředí, **autonomní** potom, že se stroje mají rozhodovat a pohybovat samostatně. A přesně taková je i realita.

Soutěžní týmy mají za úkol připravit své roboty tak, aby byli schopni samostatně projet zadanou trasu po cestičkách některého z předem vybraných městských parků. Tento úkol, jakkoli se zdá být snadný, s sebou nese mnohá úskalí, se kterými se musí tým nadšených konstruktérů a programátorů vypořádat. Robot se musí být schopen sám zorientovat v terénu, zvolit správný směr, dodržet danou trasu, reagovat na překážky a náhlé události okolního prostředí. Přitom nesmí nikdy vybočit z cesty a ideálně do cíle dovézt i těžký náklad.

Ve světě najdeme i mnoho dalších podobných soutěží autonomních robotů, za všechny bych rád jmenoval Grand Challenge/Urban Challenge (USA) a Field Robotic Event (Evropa).

2.1 Historie soutěže

Soutěž Robotour vznikla v roce 2006 ku příležitosti pátého výročí vzniku serveru robotika.cz a ze snahy vyplnit prázdné místo v oblasti soutěží outdoor autonomních robotů. Cílem organizátorů bylo spojit úsilí robotické komunity, sdílet know-how a prosadit se i na mezinárodním poli. Zároveň má mít soutěž i potenciálně praktický význam - tedy schopnost samostatného přesunu robota (ideálně i s nákladem) z místa na místo.

První tři ročníky soutěže se uskutečnily v pražských Holešovicích. Ve spolupráci s tamním planetáriem soutěž probíhala na cestách okolního parku Stromovka. Poslední ročník (2009) se potom konal v Brně v Lužáneckém parku. Letošní ročník Robotour 2010 se stěhuje na Slovensko, kde proběhne dne 18. září 2010 v jednom ze tří předvybraných parků v Bratislavě.

I pravidla soutěže se rok od roku vyvíjí a mění. Tým nemusel být vždy omezen jen na jednoho robota, ale měl možnost sestavit konvoj dvou nebo dokonce tří nezávislých vozítek, které měly jet za sebou a každý mohl plnit jinou úlohu. Startovní pozice a trasa cesty byly dříve známy i několik dní před soutěží, později se staly náhodnými a týmům byly sděleny jen několik minut před startem. Start samotný býval individuální a až později se začalo startovat hromadně. V soutěži se v pozdějších ročnících začaly objevovat překážky na trati a přepravní náklad začal být povinným. Obecně lze říci, že společně s postupující dobou a s pokroky ve světě techniky jsou pravidla soutěže stále přísnější. Na druhou stranu, stálými změnami zároveň předkládají soutěžícím každoročně nové problémy a nutí je k jejich řešení a tedy i ke stále novým a zajímavým nápadům.

2.2 Pravidla

Jak jsem již naznačil v předchozí kapitole, pravidla soutěže prošla za 5 let historie značnými změnami. Platná pravidla pro aktuální ročník Robotour 2010 jsou v plném znění uvedena na webových stránkách organizátorů [2]. Zkrácená verze zachycující nejdůležitější body a informace je uvedena v následujících odstavcích.

2.2.1 Úkol

Úkolem robota je v maximálním časovém limitu 30 minut dopravit náklad ze startu na místo určení vzdálené až 1 km. Nákladem se rozumí 5l sud piva, ať už plný (lepší bodování), či prázdný. Během celého pokusu se musí robot pohybovat samostatně a nesmí vyjet z ohraničených cest. Interakce týmu je omezena na zadání cíle bezprostředně před startem. Start i cíl jsou pro všechny společné, avšak předem neznámé. Zorientovat se v mapě a určit svou startovní pozici musí tedy robot zvládnout sám.

2.2.2 Mapa

V tomto bodě přináší letošní ročník asi největší revoluci v historii soutěže. Zatímco v předchozích ročnících bylo prostředí závodu předem známé a každý tým si jej mapoval samostatně, letos budou muset všichni roboti použít totožnou mapu jednoho ze tří bratislavských parků. Tato mapa vznikne vektorizací ortofotomapy daného prostředí a týmy si ji budou moci dále zpřesňovat. Digitální mapa bude ve formátu Java OpenStreetMap (JOSM) a bude stále aktualizována na webu organizátorů. Další podrobnosti k formátu JOSM popisuje kapitola 4.

Tento krok má za úkol odstranit omezenost robotů na předem pečlivě a individuálně zmapované prostředí. Naopak by je měl předurčit pro pohyb a orientaci v jakémkoliv prostředí popsaném univerzálním formátem mapy JOSM.

2.2.3 Roboti

V konstrukčních vlastnostech robota nechávají pravidla soutěžícím poměrně volnou ruku, ale několik omezení se přeci jen najde. Letos soutěží každý tým jen s jedním robotem!

- Robot musí být na elektrický pohon; použití tekutin, žíravin, pyrotechnických materiálů a živých bytostí je zakázáno.
- Všechny systémy a součástky na robotech musí odpovídat platným zákonům a bezpečnostním předpisům, aby nemohlo dojít k ohrožení účastníků soutěže či veřejnosti.
- Robot musí mít EMERGENCY STOP tlačítko, které ho v případě hrozícího nebezpečí zastaví. Tlačítko musí být snadno přístupné, červené a musí být pevnou součástí robota.

- Maximální velikost a hmotnost robota je limitována libovolnými dvěma dospělými osobami, které by měly být schopné robota kdykoliv přenést několik desítek metrů. Minimální velikost naopak plyne z nutné potřeby převážet 5l pivní soudek (alespoň prázdný).

2.2.4 Překážky a provoz na trase

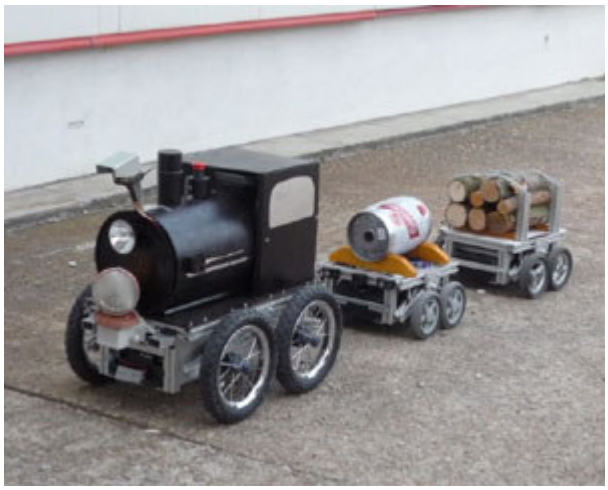
Jak již bylo zmíněno dříve, pokud robot vyjede z cesty, soutěžní pokus pro něj okamžitě končí. Stejný postih následuje i v případě jakékoliv kolize robota s překážkou. Těmi mohou být jak překážky přirozené (jako třeba parkové lavičky), tak překážky umělé - typicky krabice od banánů či kterýkoliv návštěvník parku. Ty přirozené by měl robot být schopen objet, na ty umělé může vhodně upozornit, vyčkat až budou odstraněny a pokračovat v cestě.

Vzhledem ke skutečnosti, že v letošním ročníku startují roboti hromadně z jednoho místa a budou mít i společný cíl, budou častou překážkou v provozu i ostatní roboti. Tato situace se obecně vyhodnocuje jako u jakékoliv jiné překážky. Robot tedy může svého soupeře objet, nebo vyčkat, až odjede sám. V ideálním případě by bylo dobré dodržovat pravidla silničního provozu jako je přednost zprava, vyhýbání se vpravo, předjíždění vlevo, atd.

2.2.5 Shrnutí

Je zřejmé, že letošní ročník soutěže přinesl opět mnoho inovací, mnoho problémů a mnoho zajímavých úloh k řešení. Bude velmi zajímavé sledovat, jak se s nimi jednotlivé týmy vypořádají a zda vítěz předchozích dvou podniků, LEE, obhájí své prvenství i do třetice. Já se budu ve své diplomové práci samozřejmě snažit, aby i mé řešení mělo v soutěži šanci na úspěch.

obrázek 2.1: vítěz 2008 a 2009 – LEE [1]



obrázek 2.2: robot LOCO (Radioklub Písek) [1]

3 Hardware robota

Pro účely mé práce byl vybrán robot vyvíjený na Ústavu inteligentních systému naší fakulty. Tento robot tu vznikl před dvěma lety jako bakalářská práce kolegy Tomáše Novotného [3] a v roce loňském byl použit dalším studentem pro práci s názvem „Robot pro Robotour 2009“ [11].

Tento robot je postaven na podvozku od firmy Lynxmotion s typovým označením 4WD1 [5] a jeho srdcem a mozkem je další projekt vyvíjený na Fakultě informačních technologií VUT v Brně – mikrokontrolér FITkit [4]. Jednotlivé části robota, zvláště potom senzory, jsou rozepsány v následujících podkapitolách.

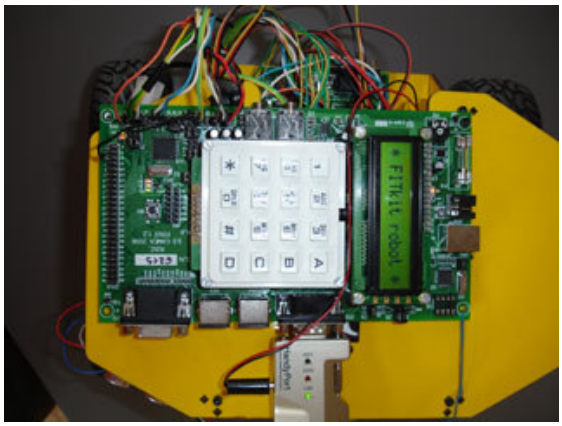
3.1 Tělo

Tělo robota se skládá ze dvou základních částí - jsou jimi již zmiňovaný podvozek 4WD1 a dále na něm namontované dvě horizontální platformy nesoucí řídící mikrokontrolér, baterie, kontaktní nepájivé pole a všechny další robotovy senzory.

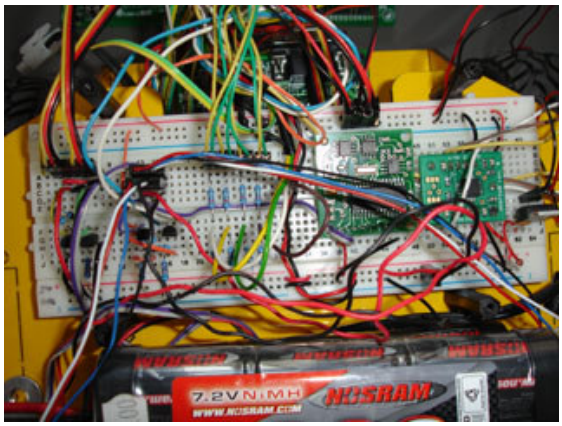
Podvozek je konstruován pro vnitřní i vnější použití. Jedná se o diferenciální podvozek poháněný čtyřmi motory (GHM-04 [5]). Ke každému motoru je také připojen optický enkodér. O ovládání kol (a potažmo zatáčení) se potom stará deska Scorpion (HB-04 [5]), která je společně s bateriemi umístěna hned na podvozku a připojena k řídícímu FITkitu.

Na nižší platformě je umístěno kontaktní nepájivé pole osazené elektronickým kompasem a akcelerometrem. Pole dále obsahuje převodníky úrovní napětí pro některé periferie. Posledním zařízením dolní plošiny je skupina sonarů - i tyto jsou připojeny ke kontaktnímu poli. Podrobnosti jednotlivých senzorů jsou popsány v kapitole 3.2 .

Horní plošina je potom vyhrazena pro řídící centrum robota. Jednotlivé piny sběrnice JP9 použitého FITkitu jsou připojeny na příslušná místa kontaktního pole. Ke vzdálenému řízení robota z terminálové konzole slouží sériové rozhraní FITkitu. Spojení s PC je možné buď klasickým sériovým kabelem, nebo bezdrátově pomocí dvojice HandyPort Bluetooth modulů.



obrázek 3.1: detail FITkitu [3]



obrázek 3.2: detail nepájivého pole [3]

3.2 Senzory

Pokud jsme mikrokontrolér nazvali srdcem a mozkem robota a motory (jako jediné efektory) jeho pohybovou soustavou, potom nezbývá, než popsat senzory jako robotova smyslová ústrojí. Stejně jako člověku mu pomáhají orientovat se v prostředí, ve kterém se nachází, vnímat své okolí a dle těchto informací také jednat.

3.2.1 Sonary

„Sonar (z anglického SOund NAvigation and Ranging) je jedno ze zařízení používaných pro detekci předmětů v okolí robota. Dalšími možnostmi jsou například infračervená nebo dotyková čidla.

Princip je vcelku jednoduchý. Zařízení měří čas, který trvá od vyslání do přijmutí odraženého zvukového signálu. Ze znalosti rychlosti zvuku v prostředí (v našem případě vzduch) lze následně vypočítat vzdálenost objektu. Na prakticky stejném principu je založena echolokace netopýrů.

Měření však nemusí být přesné a může být nepříznivě ovlivněno několika faktory. Zvuk může například narazit pod takovým úhlem, že dojde k jeho odražení od vysílajícího sonaru a ozvěna není zachycena. Falešné měření může také způsobit odraz od několika předmětů, který se vrátí za podstatně delší dobu – to vede k závěru, že objekty jsou od nás mnohem dále. Při použití více sonarů může docházet k příjmu zvuků vyslaných jiným sonarem. Tato situace může nastat i při použití jednoho sonaru na robotovi, pokud je v okolí další robot se sonarem, který pracuje na stejné frekvenci.“ [3]

Náš robot je osazen sonary typu SRF08 firmy Devantech. Tyto jsou umístěny na přední straně robota – jeden uprostřed a dva na krajních rozích platformy, aby mohly detekovat blížící se překážky ve směru jízdy. Sonary pracují na frekvenci 40kHz a jsou vybaveny i světelnými čidly. Rozsah měření sonarů je od 3 centimetrů do 6 metrů.

3.2.2 Kompas

Již po staletí je kompas, společně s mapou, pomůckou číslo jedna při potřebě orientace a navigace v neznámém prostředí. Jeho funkce je založena na působení magnetického pole Země. Díky němu ukazuje magnetická střelka kompasu vždy k severu a my můžeme správně zorientovat mapu a určit směr cesty.

Elektronický kompas, jakým je osazen náš robot (CMPS03 firmy Devantech), pracuje na stejném principu, byť trochu jinak provedeném. Zařízení nevyužívá pohyblivé magnetické střelky, nýbrž zmagnetovaného plíšku vyrobeného z feromagnetického materiálu (směs železa a niklu). Tento má v případě nulového vlivu externího magnetického pole a za daného proudu určitý odpor. Vnější magnetické pole Země však původní směr zmagnetování ovlivní a tím dojde ke změně odporu plíšku. Při konstantním proudu pak už jen stačí měřit rozdíly napětí v obvodu způsobené těmito změnami.

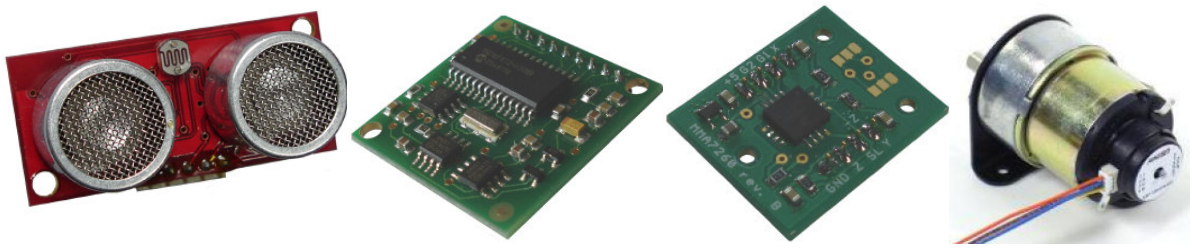
Výstupem kompasu při správném odečtení je natočení robota vůči magnetickému severu země v rozsahu 0 – 359°.

3.2.3 Akcelerometr a optické enkodéry

Poslední dvojice z robotových senzorů. Oba dva, i když každý jiným způsobem, slouží ke sledování pohybu robota a tedy i k mapování jeho trasy a zpětnému odhadu aktuální pozice.

Čtyři optické enkodéry (QME-01 [5]) měří rychlost otáčení jednotlivých motorů. Každému pootočení hřídele motoru odpovídá určitý počet impulzů odečtených z kolečka enkodéru. Při správném přepočtu jsme potom schopni zjistit přesné pootočení každého kola a tedy i rychlost, vzdálenost a směr pohybu robota.

Tříosý akcelerometr (typ ACC7260) umožňuje měřit zrychlení způsobené gravitací nebo nerovnoměrným pohybem. Dalšími přepočty by bylo možné zjistit i náklon robota (poměr zrychlení ve 2 osách), či vektor okamžité rychlosti (integrace všech tří os). Modul je nastaven na nejvyšší citlivost a umožňuje tak měřit zrychlení v rozsahu ±1,5g (při horší citlivosti až ±6g), což je pro možnosti robota zdaleka postačující



obrázek 3.3: detail senzorů – (zleva doprava) sonar, el. kompas, akcelerometr, opt. enkodér [3]

3.3 Ovládání

Jak jsem již naznačil v kapitole 3.1, řídící mikrokontrolér pouze zajišťuje správnou činnost všech periferií robota. Samotné ovládání robota je řízeno z připojeného PC nebo lépe notebooku. Pro navázání spojení, je třeba nastavit COM port a sériovou linku na tyto hodnoty: rychlost 115 200 bps, 8 data bitů, 1 stop bit, bez parity. Po připojení jsou na terminál neustále zasílány naměřené hodnoty ze senzorů v následujícím formátu: [3]

```
Sonary: 25 31 24 [cm] Kompas: 119 [st] Akc: -0.003 +0.002 +1.002 [g]
Sco: 1381 1381 Rych: 0 0 0 0 [100*cm/s]
```

První tři čísla určují vzdálenost (v cm) k případným překážkám naměřenou levým, středovým a pravým sonarem. Následuje údaj o natočení robota (ve stupních) z elektronického kompasu. Další tři hodnoty jsou z odečteny z akcelerometru a udávající zrychlení robota ve všech třech osách (x, y, z). Další dvě čísla (v příkladu totožná) vyjadřují signál pro ovládání motorů odesílaný FITkitem desce Scorpion. První z nich určuje rychlost a druhé směr. Hodnota 1381 je středová a robot v tomto stavu

nehybně stojí. Poslední čtyři hodnoty zobrazují rychlost robota na základě signálů z optických enkodérů všech čtyř kol (přední levé, pravé, zadní levé, pravé).

Posíláním signálů do terminálu je potom možné robota řídit. Jsou k tomu vybrána tlačítka ve tvaru šipek – klávesy ‚i‘ a ‚k‘ ovládají pohyb vpřed/vzad a klávesy ‚j‘ a ‚l‘ zatáčení vlevo/vpravo, přičemž platí, že čím intenzivnější signál jde z klávesnice, tím rychleji robot jede nebo ostřeji zatáčí. Klávesa ‚o‘ slouží k okamžitému zastavení robota.

Pro svoji práci plánuji použít 9“ notebook Asus Eee 901, který s sebou robot poveze a na kterém poběží navigační program. Notebook bude připojen sériovým kabelem k FITkitu a spuštěný program tak bude moci komunikovat přímo přes sériové rozhraní.

3.4 Způsobilost pro soutěž

Co se velikosti týče, patřil by robot rozhodně k těm menším v soutěži. Po přidání třetího „patra“ platformy by ale neměl být problém, aby alespoň předepsaný prázdný soudek uvezl. Podvozek je dost robustní a motory na to sílu rozhodně mají.

Senzory jsou na tom bohužel už o poznání hůře. Citlivost čelních sonarů je sice dostatečná ke včasné detekci nečekaných překážek, ale v místech, kde kolem cesty nebudou lavičky, keře, či jiné zjistitelné předměty, bude robot prakticky slepý a náchylný k vyjetí z cesty. Zde se bude třeba spolehnout pouze na velmi přesnou odometrii, což by naopak při kombinaci enkodérů, akcelerometru a kompasu mohlo být robotovou silnou stránkou.

Celkově si tedy myslím, že má robot reálnou šanci se do závodu kvalifikovat a nějaké ty body i nasbírat.

3.5 Možná vylepšení a úpravy

Z rozboru senzorů plyne, že zde by bylo třeba udělat radikální změny. V dnešní době jsou k dispozici vymoženosti jako laserové skenery, ultrazvukové snímače, gyroskopy, kamery s vysokým rozlišením a notebooky s dostatečným výkonem na složité zpracování videa. A soutěžní týmy tyto technologie hojně využívají. Pro nás je to už ale přeci jenom naprosto jiná „cenová skupina“ a navíc, vzhledem k rozměrům robota, by se jednalo spíše o nutnost konstrukce robota nového, než o skromné vylepšení. Přeci jen ale existuje pár drobností, které by bylo třeba udělat.

První z nich je umístění onoho velkého červeného vypínače někam na tělo robota. Jednak je to jednou z podmínek nutných k účasti v závodu a navíc to jistě bude i praktičtější způsob zapínání a vypínání robota, než je stávající rozpojování napájecích kabelů.

Druhou nutnou úpravou bude zbudování nosné konstrukce pro řídící notebook a také uchycení pro předepsaný 5l pivní soudek. Pro tyto účely bude na robota připevněna také třetí plošina.

Další úpravou, kterou mi doporučil autor robota, je nahrazení nepájivého polem pájivým a ideálně celé jeho zapouzdření. Usnadnila by se tak výrazně manipulace s tímto modulem a navíc by

se eliminovala možnost vypadnutí či rušení některého z kontaktů při pohybu a drncání po nerovném terénu. Nepájivé pole si splnilo svůj hlavní úkol během sestavování a testování robota.

Poslední zvažovanou úpravou je zapojení GPS přijímače. To by bylo možné realizovat buď jako další senzor umístěný na těle robota a řízený mikrokontrolérem a nebo jako modul připojený přímo k ovládacímu notebooku, například pomocí rozhraní USB. V předchozích ročnících sice nebyly GPS přijímače moc úspěšné, neboť jejich přesnost (navíc zhoršena městským šumem a stromy v parku) nedostačuje pro udržení robota na úzkých parkových cestách, ale mohou být dobrým pomocníkem při zjištění startovní pozice robota.

4 Mapa

Vzhledem k výrazné změně pravidel, co se použití mapy týče, rozhodl jsem se, že jí věnuji celou zvláštní kapitolu. Podrobná znalost specifikací mapy bude nutná pro celou následující práci.

Již jsem uvedl, že všechny týmy budou nuceny použít jednu oficiální mapu terénu. Dále také, že tato mapa bude kódována ve formátu Java OpenStreetMap [6]. Tento souborový formát je postaven na obecném značkovacím jazyce XML. Jako referenční souřadnicový systém pro popisovaná vektorová geodata je použit WGS 84 [8], tedy celosvětově užívaný standardizovaný systém popisující polohu souřadnicemi zeměpisné šířky a délky tak, jak je všichni důvěrně známe z internetových map a GPS navigací.

Dle specifikací obsahuje JOSM popis tři základní druhy záznamů: [2]

- **`node`** – vrchol, který má klíčové atributy id, lat a lon
- **`way`** – cesta, která se skládá z odkazů na node pomocí *<nd ref="node id">*
- **`relation`** – vazby, které lze použít např. pro popisy budov

„Každý záznam má dále atributy *user/uid* a *timestamp*, tj. který uživatel a kdy daný bod editoval. Číslo v atributu *changeset* pak odpovídá pořadí revize mapy. Určitě zajímavé informace lze nalézt také v *<tag>* (například *šířku* dané *way* atp.).“

```
<node id="242744133"
    lat="50.1057063" lon="14.42758"
    version="2" changeset="772392"
    user="Petr" uid="17615"
    visible="true"
    timestamp="2009-01-11T13:03:48Z">
  <tag k="created_by" v="JOSM"/>
</node>
```

obrázek 4.1: příklad záznamu node [2]

```
<way id="33898726" visible="true"
    timestamp="2009-05-01T20:55:40Z"
    version="1" changeset="1041618"
    user="Petr" uid="17615">
  <nd ref="388382600"/>
  <nd ref="388382681"/>
  <nd ref="275620115"/>
  <tag k="highway" v="footway"/>
  <tag k="source" v="cuzk:km"/>
</way>
```

obrázek 4.2: příklad záznamu way [2]

5 Souřadnice GPS

Jak ukazuje předchozí kapitola, použitá mapa ve formátu JOSM využívá pro zápis pozice jednotlivých bodů souřadnice v systému WGS-84. Vzhledem k faktu, že se souřadnicemi bude navrhovaný program pracovat od samého začátku až do konce své činnosti, rád bych jim věnoval tuto kapitolu. Vysvětlovat zde však princip činnosti celého GPS systému by bylo asi výrazně nad rámec mé práce, ale alespoň jisté základy věřím, že nebude od věci osvěžit.

Každý bod v tomto systému je popsán svojí zeměpisnou šířkou a zeměpisnou délkou. Tyto údaje jsou zapisovány ve stupních a to v rozsahu -90° až +90° zeměpisné šířky a 0° až 360° zeměpisné délky. Vzhledem k pozici České republiky a prakticky i většiny Evropy se potom bavíme hlavně o *severní šířce* (N) a *východní délce* (E). Následující příklad ukazuje zápis souřadnic středu města Brna (totožný bod) v několika různých formátech:

```
+49.195133 +16.608183
N 49°11.708 E 16°36.491
49°11'42.497"N, 16°36'29.477"E
```

5.1 Vzdálenost bodů

V programu budeme často potřebovat zjistit, jak daleko jsou od sebe 2 konkrétní takto popsané body. Tento výpočet řeší následující rovnice:

$$\begin{aligned} a &= \sin^2\left(\frac{\Delta lat}{2}\right) + \cos(lat_1) * \cos(lat_2) * \sin^2\left(\frac{\Delta lon}{2}\right) \\ c &= 2 * a \tan 2\left(\sqrt{a}, \sqrt{(1-a)}\right) \\ d &= R * c \end{aligned} \tag{1}$$

V těchto vztazích písmeno ***R*** reprezentuje střední poloměr Země (6 371km). *lat* je zeměpisnou šířkou a *lon* potom zeměpisnou délkou prvního a druhého bodu. Písmeno ***d*** reprezentuje výslednou vzdálenost bodů v kilometrech.

5.2 Azimut

Druhou velmi potřebnou informací pro nás bude azimut z jednoho bodu do druhého. Jeho výpočet je popsán touto rovnicí:

$$\Theta = a \tan 2\begin{pmatrix} \sin(\Delta lon) * \cos(lat_2), \\ \cos(lat_1) * \sin(lat_2) - \sin(lat_1) * \cos(lat_2) * \cos(\Delta lon) \end{pmatrix} \tag{2}$$

Proměnné *lat* a *lon* reprezentují, stejně jako v předchozí rovnici, zeměpisnou šířku a délku. Výsledný úhel **Θ** vychází po převodu z radiánů na stupně v rozsahu -180° .. +180°.

6 Stanovení úkolů

Nejdříve ze všeho je třeba představit si situaci, v jaké se bude robot nacházet a dále potom se zamyslet, jaké problémy a v jaké posloupnosti bude robot nucen řešit. Z těchto problémů plynou v praxi logické úkoly a posléze algoritmy, které bude robot nucen vykonat na své strastiplné cestě za vítězstvím.

6.1 Start

Odstartování samotné bude hned prvním robotovým úkolem. S ohledem na změnu v pravidlech, která nařizuje hromadný start v jednu chvíli a z jednoho místa, už nebudou soutěžící moci startovat roboty ručně, aby se jim ve vzniklém zmatku nepletli do cesty. Bude tedy třeba implementovat časovač, aby byl robot schopen autonomního startu v požadovanou chvíli. Méně jak 1 minutu před startem už nebude možné jakkoli s robotem komunikovat.

Startovní oblast každého robota je velká 1,5 x 1,5 metrů, oblasti jsou navzájem bezprostředně přilehlé a start proběhne v místě s minimální šířkou cesty 3 metry.

6.2 Orientace

V daný čas tedy robot odstartuje, ale mluvit v tomto momentě o startu jako začátku fyzického pohybu by bylo možná poněkud předčasné. Představme si situaci, kdy nás někdo vysadí, s černým pytlem na hlavě, uprostřed lesa v naprosto neznámé oblasti. Pokud nebudeme mít velké štěstí na nějaký výrazný orientační bod, zabere nám určení své vlastní pozice na mapě poměrně dlouhou dobu a nemalé úsilí, i s kompasem v ruce a dobrým zrakem i sluchem. V noci je tento úkol téměř nadlidský.

Náš robot to nebude mít o nic jednodušší. Po „startu“ se ocitne na naprosto neznámém místě v poměrně rozsáhlé oblasti. Navíc díky svým omezeným senzorům bude prakticky slepý. I přesto však bude muset co nejpřesněji určit v které části parku, na které cestě a v jakém jejím úseku se pravděpodobně nachází.

6.3 Plán trasy a cesta

Ani v tuto chvíli ještě robot nemůže vyrazit na svou „pouť“. Na druhou stranu, čeká ho teď asi nejjednodušší úkol. Tím je nalezení vhodné trasy pro svoji cestu k cíli. Svoji aktuální pozici již zná, tu cílovou zná dokonce již od začátku (zadáno před startem) a musí si tedy podle své mapy určit postupně všechny cesty, křižovatky a odbočky, které k cíli vedou a které bude schopen projet..

Až teď se tedy konečně můžeme rozjet. Úsek za úsekem, křižovatku za křižovatkou, bude třeba počítat vzdálenost a natočení kol k úspěšnému projetí následující části. Během toho všeho je důležité zůstat vždy všemi koly na cestě a stále myslet na možnost výskytu nečekané překážky.

7 Návrh řešení

Když pominu autonomní startování, které nebude problém realizovat pomocí synchronizace se systémovým časem a odpočtu, první úlohou pro řešení bude ono stanovení startovní pozice robota v mapě.

7.1 Zjištění startovní pozice

Bez možnosti využití systému GPS, navigačních majáků nebo jiných zařízení je pro našeho robota řešení tohoto problému velice obtížné. Na globální mapě (např. celé Evropy) by robot neměl nejmenší šanci ani na velmi přibližný tip.

7.1.1 Možné řešení

Jistou variantou řešení by bylo, aby robot po startu začal systematicky prohledávat své okolí a svoji cestu si mapoval. Po zaznamenání každého dalšího celku (cesta, křižovatka či delší úsek) by se spustil porovnávací algoritmus, který by se snažil nalézt a přiřadit prozkoumanou oblast odpovídajícímu místu ve své referenční mapě. Při neúspěchu by robot pokračoval zmapováním další části a takto stále dokola, dokud by nedošlo ke shodě a stanovení pozice. Náročnost této úlohy roste v závislosti na velikosti území zmapovaném referenční mapou.

Nutnou podmínkou tohoto řešení je, aby byl robot schopen přesně odlišit cesty od ostatních ploch a překážek, případně uměl překonat i menší překážky a terénní nerovnosti. A to i takové, na které by při normálním průjezdu soutěžní trasy nemohl narazit. Náš robot, bohužel, na tyto úkony není ani zdaleka vybaven. Věřím, že terénní překážky by nějak zvládl, ale na přesné sledování cesty jsou jeho senzory příliš nedokonalé. To by musely být celé délky cest lemovány stěnami a to je v reálném prostředí nemožné.

7.1.2 Zamýšlené řešení

Pro nalezení vhodného řešení bylo třeba zamyslet se nad tím, co všechno máme k dispozici a čím si můžeme pomoci a úlohu trochu zjednodušit.

Klíčovým senzorem robota v této části byl jeho kompas. Vzhledem k faktu, že v rámci startovního prostoru bude možné robota postavit dle vlastní potřeby, budeme předpokládat, že robot bude stát vždy rovnoběžně s osou cesty, pokud možno po směru předpokládané trasy k cíli. V tento moment je robot schopen pomocí kompasu změřit svůj azimut. Dále je možné brát každé dva sousední body všech cest v mapě, vypočíst azimut úseku mezi nimi a tento porovnat s robotovým. Úsek (cesta) s nejmenší odchylkou bode robotovou startovní pozicí.

V komplexní mapě velkého území by to byl opět velmi těžko řešitelný úkol a odpovídajících kandidátů by bylo několik. Proto je třeba si trochu pomoci a rozsah mapy co možná nejvíce omezit.

První a velmi výraznou pomoc je možné najít v samotných pravidlech. V kapitole 2.2.1 jsem zmínil, že cíl může být vzdálen až 1km od startovní pozice. Při pohledu z druhé strany tedy můžeme po zadání cílových souřadnic vyloučit všechna místa a cesty za touto hranicí a zbývá nám kruh o poloměru 1km. Další omezení by bylo možné z odhadu, že start bude například dále než 300 metrů od cíle, čímž by vzniklo pomyslné mezikruží, ale to už by byla příliš velká spekulace.

Druhá pomoc se nabízí ze samotné definice mapového formátu JOSM. Zde jsem uvedl (kap. 4), že u záznamů typu `way` (cesta) je možné zaznamenat typ příslušné cesty v mapě. Dále vycházím opět z pravidel, využiji faktu, že soutěž se koná na cestičkách v některém z parků a všechny ostatní cesty (dálnice, silnice, ...) je možné opět vyloučit. Pokud by snad zadaná mapa obsahovala pouze záznamy o možných cestách v parku, úloha se nám ještě zjednodušuje.

Po těchto omezeních v mapě už by bylo možné začít měřit, počítat a s velkou pravděpodobností i dojít k reálné startovní pozici. Vzhledem k tomu, že je vše závislé na robotově azimutu, bylo by dobré jeho měření několikrát zopakovat (mezi nimi třeba i popojet o 10cm vpřed či vzad) a výsledky vhodně zprůměrovat, abychom došli k co nejpřesnějšímu číslu.

obrázek 7.1: možné nalezení startovní pozice

Obrázek 7.1 znázorňuje celý postup možného nalezení startovní pozice robota ve zjednodušené verzi (cíl max. 300 metrů od startu) na mapě brněnského parku Lužánky. Zelený křížek značí cílovou pozici. Červená kružnice vymezuje oblast možných startů a červeně jsou také označeny cesty mimo tuto oblast. Oranžovou barvou jsou označeny cesty, které neodpovídají příznaku parkové cesty či chodníku. Zbylé cesty jsou označeny modře a na nich jsou také modře vyznačeny jednotlivé body (node). Po vypočtení hodnot azimutů mezi nimi stanovíme za startovní pozici úsek mezi dvěma body označený zeleným kolečkem, neboť jeho hodnota 218° odpovídá i hodnotě naměřené robotem.

7.1.3 Zvolené řešení

Ani jedna z popsaných metod mi ale nepřišla jako stoprocentně funkční a vhodná. Proto jsem byl velmi rád, když jsem zprovoznil a úspěšně otestoval spojení externího GPS přístroje s řídícím notebookem. Tímto GPS zařízením je outdoorový přístroj Garmin Colorado® 300 (obrázek 7.2), který je vybaven velmi citlivým a přesným GPS čipem. Přístroj připojený přes USB rozhraní je mimo jiné také schopen posílat do notebooku aktuální data ve formátu NMEA. Pomocí originální aplikace - Garmin Spanner - jsou data dále předávána na simulovaný systémový COM port, odkud již je možné je pohodlně číst a zpracovávat.

Následující řádek znaků je ukázkou jedné z 12 NMEA vět, které jsou neustále cyklicky odesílány z GPS do počítače:

```
...
$GPGLL,4912.8577,N,01630.5557,E,173257,V,S*42
...
```

Zvýrazněná část věty reprezentuje aktuální naměřenou polohu zařízení (a potažmo celého robota) v souřadnicích systému GPS – `N 49°12.8577 E 016°30.5557`.

Z několika měření (přesnost zařízení je cca 5 metrů) je snadné aproximovat bod nejpravděpodobnější pozice robota. Následným porovnáním s jednotlivými body mapy už nic nebrání stanovení přesného umístění robota na parkové cestě.

Toto řešení je navíc naprosto univerzální pro jakkoliv velkou a podrobnou mapu a nevyžaduje proto již žádné další omezení, úpravy či filtrování.

Nyní již konečně známe startovní i cílové souřadnice a můžeme začít s plánováním trasy.

obrázek 7.2: Garmin Colorado® 300 (převzato z www.garmin.com)

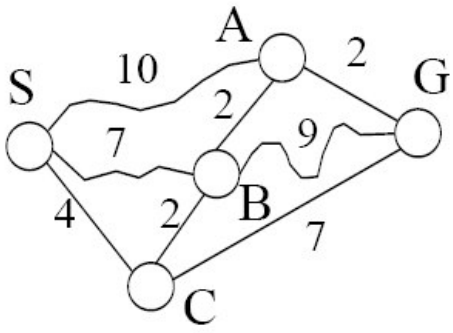
7.2 Plánování trasy

Tento úkol bude pro robota výrazně jednodušší než ten předchozí i než ten následující. Nejsou k němu potřeba žádné senzory či efektory - jde čistě o výpočetní algoritmy.

Pro tento účel jsem si zvolil jeden ze základních typů algoritmů umělé inteligence – Uniform Cost Search (UCS). Jedná se o slepou metodu vyhledávání cesty hranově ohodnoceným grafem. Když tedy naši mapu jednoduše převedeme na graf, kde uzly grafu budou reálné křižovatky cest a hrany grafu budou tyto cesty (ohodnoceny číslem reprezentujícím jejich délku), algoritmus nám najde cestu mapou ze startu do cíle. Algoritmus s sebou navíc nese atributy **úplnosti** a **optimálnosti**, máme tedy jistotu, že pokud cesta bude existovat, algoritmus ji určitě najde a bude-li cest více, najde vždy tu nejlepší. Využijeme verzi algoritmu s eliminací návratů v cestě.

Na začátku tedy bude nutné spočíst vzdálenosti jednotlivých křižovatek a vytvořit prázdný seznam. Do seznamu vložíme startovní uzel (křižovatku) a délku trasy k tomuto místu (tedy 0). Dále již algoritmus bude pracovat v cyklu, kdy vždy ze seznamu vybere uzel s nejnižším ohodnocením a je-li to cíl, potom končí a daný uzel reprezentuje hledanou trasu. Pokud ne, vybraný uzel expanduje, všechny bezprostřední následníky (kteří nejsou jeho předky) umístí do seznamu (včetně nového ohodnocení) a na závěr seznam redukuje – z uzlů, které se v něm opakují vícekrát, ponechá pouze ten s nejlepším ohodnocením, ostatní smaže. Je-li seznam na začátku cyklu prázdný, potom úloha nemá řešení. Princip demonstruje obrázek 7.3.

Prakticky používaná verze
s eliminací návratů k předkům
na již prošlé cestě:



Step: Open:
0 [((S),0)]
1 [((A,S),10),((B,S),7),((C,S),4)]
2 [((A,S),10),~~((B,S),7)~~,~~((S,C,S),8)~~,((B,C,S),6),((G,C,S),11)]
3 [~~((A,S),10)~~,((G,C,S),11),~~((S,B,C,S),13)~~,((A,B,C,S),8),
~~((C,B,C,S),8)~~,~~((G,B,C,S),15)~~]
4 [~~((G,C,S),11)~~,~~((S,A,B,C,S),18)~~,~~((B,A,B,C,S),10)~~,
((G,A,B,C,S),10)] ⇒ ((G,A,B,C,S),10) = Goal

obrázek 7.3: algoritmus UCS (převzato z přednášek IZU [7])

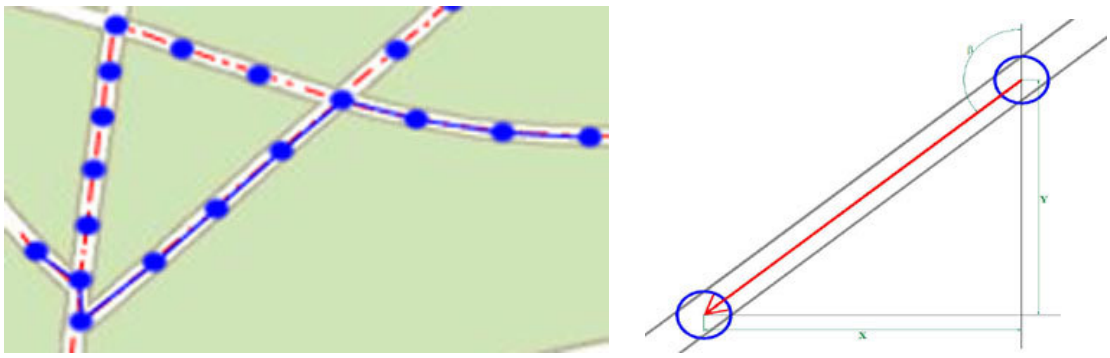
Pokud bychom chtěli algoritmu práci zjednodušit a urychlit, můžeme graf (mapu) trochu omezit způsobem naznačeným v kapitole 7.1.2. Takové území by bylo zdaleka dostačující, neboť bohatě pokrývá jak okolí startu, tak území cíle a samozřejmě i ty nejkratší cesty a trasy mezi nimi. Další omezení by bylo možné například vytvořením druhé pomyslné kružnice se středem v místě startu a stejným poloměrem jako u kružnice první. Prohledávaný prostor by potom byl omezen na plochu jejich průniku. Naopak, připustíme-li možnost, že by algoritmus v omezeném území trasu nenašel, bylo by třeba ho zvětšit (například na dvojnásobek poloměru) nebo omezení zrušit úplně.

7.3 Průjezd trasou

Když už má sestavený plán trasy, může se náš robot konečně rozjet. Dle specifikací se každá cesta v mapě skládá z mnoha trasových bodů. Trasové body by měly být natolik nahuštěné, že mezi každými dvěma bude přímá viditelnost a tedy bude existovat přímá trasa od jednoho k druhému.

Tohoto faktu využijeme při robotově navigaci. Jelikož robot nemá k dispozici senzory, které by byly s jistotou schopny sledovat povrch, po kterém se pohybuje, či okraje cesty, bude se muset spolehnout hlavně na velmi přesnou navigaci a odometrii. Průjezd trasou tedy rozdělíme na mnoho dílčích úseků. V každé podúloze bude vždy vypočten směr a vzdálenost k následujícímu bodu na trase (viz. vzorce v kapitole 5) a následně proveden posun a případné korekce. Takto by se měl robot bod po bodu dostat bez problémů až do cíle. Princip naznačen na obrázku 7.4.

Pro dosažení co nejvyšší přesnosti se budeme snažit navigovat robota po ose (uprostřed) cest. Záměrem je, aby robot v místě každého trasového bodu stál skutečně uprostřed cesty. Tím zajistíme nejmenší možnou pravděpodobnost vyjetí robota z cesty a eliminujeme tak možné nepřesnosti, které mohou vzniknout v důsledku působení vnějších vlivů (jemné proklouznutí kol, atd.). V tomto směru nám pomáhají samy trasové body, které by měly být zaměřeny co nejpřesněji na střed cesty. Jediným úkolem tedy bude umístit robota co nejblíže středu cesty i na „startovním roštu“. To se samozřejmě nemusí povést vždy stoprocentně, ale tato chyba bude natolik zanedbatelná, že by neměla ovlivnit další navigaci robota.



obrázek 7.4: princip navigace a výpočtu trasy

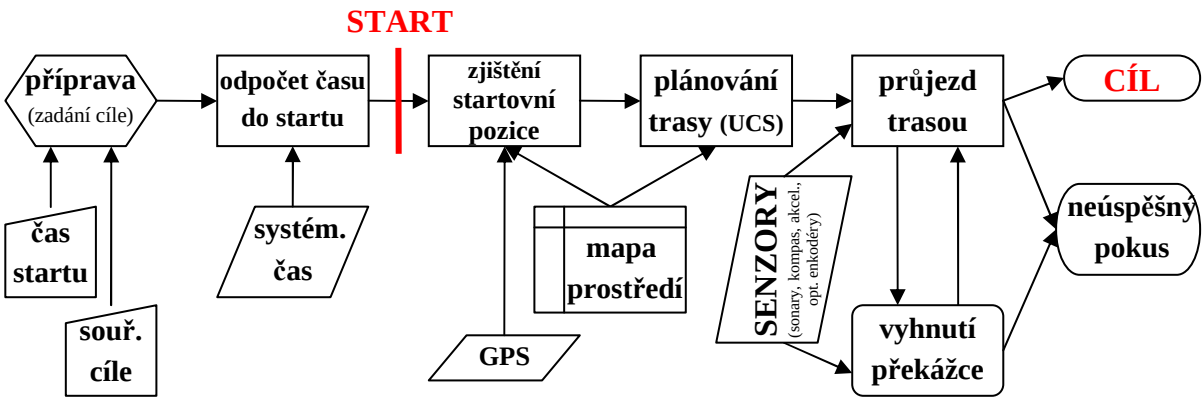
7.4 Překážky

Posledním problémem, se kterým se bude robot potýkat, je neustálá možnost výskytu překážek na trase (kap. 2.2.4). Před i během vykonávání jakéhokoliv pohybu bude tedy nutné detekovat (pomocí sonaru), zda je cesta skutečně volná.

Při detekci překážky je možné počkat, zda tato nezmizí sama (chodci, ostatní roboti), případně po delší době zvukově upozornit a vyčkat na odstranění (umělé překážky - krabice). Pokud ani po uplynutí dalšího času nedojde k odstranění překážky, musí se robot pokusit o úhybný manévr (lavičky a jiné objekty parku). Po vyřešení problému bude robot pokračovat ve své trase a očekávat další nástrahy.

7.5 Schéma činnosti

Celou činnost robota od startu k cíli a návaznosti jednotlivých bloků zobrazuje následující schéma:



obrázek 7.5: schéma činnosti robota

8 Implementace

Navrženému řešení odpovídá i jeho implementace. V následujících podkapitolách bych rád postupně popsal a rozebral činnost jednotlivých součástí systému. Kde to bude možné, upozorním na případné problémy, které se během implementace vyskytly, nebo zdůrazním jiné zvláštnosti zvoleného řešení.

Jednotlivé naprogramované moduly prakticky velmi blízce kopírují schéma činnosti z předchozí stránky (obrázek 7.5).

8.1 Implementační prostředí

Pro implementaci celého systému jsem si zvolil jazyk C/C++. Při výběru jsem vycházel z předpokladu, že výsledný program nevyžaduje žádné složité grafické výstupy ani prvky ovládání. Veškerá jeho činnost spočívá ve čtení a zápisu do souborů a externích zařízení. Celá interakce s uživatelem je tak redukována na úvodní výzvu k zadání několika jednoduchých údajů a k jejich přečtení ze vstupu, k čemuž konzolová aplikace bohatě dostačuje. Možnosti volby typů proměnných i vlastní konstrukce algoritmů jsou zde také dostačující a navíc je výsledná aplikace relativně malá a rychlá. Ke svému běhu nepotřebuje instalaci dalších prostředí či doplnění externích knihoven – vystačí se standardními. V neposlední řadě jsem potom zohlednil i fakt, že s tímto jazykem mám zkušenosti z předchozích prací a projektů (včetně bakalářské práce).

Program je tedy určen pro prostředí a testován v prostředí operačního systému Microsoft Windows XP. Neměl by ovšem mít problémy se správnou funkčností ani v prostředí starších systémů jako MS Windows 2000 či 98. U novějších jako je Vista a Windows 7 už jsi nejsem jistý správnou funkčností komunikačního modulu.

Jako vývojové prostředí pro svoji práci jsem zvolil program Dev-C++ (konkrétně ve verzi 4.9.9.0). Tento program dokáže přehledně zvýrazňovat syntaxi zdrojových kódů a obsahuje dostatečně silné nástroje pro vývoj a ladění jednoduchých i složitějších aplikací. I s ním už mám předchozí velmi dobré zkušenosti.

8.2 Funkce Main( )

Jedná se o hlavní řídící funkci celého systému, ale zároveň je její zdrojový kód také nejkratší a nejjednodušší. Po spuštění aplikace se tato funkce postará, aby byly postupně volány a vykonány všechny moduly programu, přesně jak bylo navrženo.

Následující úsek kódu (obrázek. 8.1) ukazuje celou funkci `main()` tak, jak se nachází v souboru `main.c` a dává tak čtenáři představu o struktuře celé aplikace a funkci jednotlivých modulů.

```
int main(int argc, char *argv[])
{
  system("cls"); // cisti obrazovku

  nacti();       // nacte mapu z ext. souboru
  cil();         // vyzve k zadani cilovych souradnic
  timer();       // vyzve k zadani casu startu
  fix();         // zjisti uvodni pozici robota
  route();       // sestavi plan trasy Start - Cil
  drive();       // naviguje robota zvolenou trasou

  return 0;      // konec programu
}
```

obrázek 8.1: funkce main()

8.3 Načtení mapy

První činností programu je načtení příslušné mapy soutěžního prostředí, neboť tato je dále neustále využívána ve všech následujících modulech.

O načtení se stará modul obsažený ve zdrojovém souboru `mapa.c` . Vše podstatné zde obstarává funkce `nacti()` volaná z funkce `main()` - viz. předchozí obrázek. Tato funkce obsahuje jeden velký cyklus, který po úspěšném otevření souboru s mapou (více v kap. 8.3.2) prochází jednotlivé jeho řádky a hledá elementy obsahující záznamy o jednotlivých trasových bodech a cestách (jak byly popsány v kapitole 4). Zajímavé jsou pro nás tedy *tagy* uvozené jako: **<node** pro jednotlivé body, **<way** pro celé cesty a **<nd** pro identifikaci bodů na těchto cestách. Dalším zajímavým atributem cesty by mohl být tag **<tag k="highway" v="footway"/>** , který nese informaci o typu dané cesty (v tomto případě chodník / parková cesta).

Všechny nalezené elementy i se svými atributy jsou postupně ukládány do připravené struktury (více kapitola 8.3.1). Po dosažení konce mapového souboru je tento uzavřen a řízení je vráceno funkci `main()`.

Doplňkovou funkcí tohoto modulu je funkce `mapa()`, která může být volána z kteréhokoliv místa programu (defaultně volána, pokud si uživatel přeje mapu po načtení vypsat) a zajišťuje výpis celé struktury s načtenou mapou ve formátu, jaký ukazuje následující příklad:

```
Node 001 : N 49.227863 E 16.595516  *
Node 012 : N 49.227894 E 16.595634  *
Node 032 : N 49.227879 E 16.595585
Way 01 : end1 = 001 (000) ; end2 = 012 (011) ; + 01 nodes  (footway)
  node: 032 (026)
```

obrázek 8.2: příklad výpisu mapy

Každý trasový bod (`Node`) i cesta (`Way`) jsou označeny svým identifikátorem. Čísla v závorkách jsou potom indexem daného bodu ve struktuře (problém popsán na konci kapitoly 8.3.1). Hvězdičkou označené body jsou křižovatkami cest.

Položky `end1` a `end2` značí krajové body cesty. Kromě těchto krajových může cesta obsahovat i další body, jejichž počet je reprezentován řetězcem `+ XX nodes` a na dalších řádcích následuje jejich seznam. Cesty s koncovkou `(footway)` jsou cesty, které nás v závodě zajímají (parkové pěšiny, ap.).

8.3.1 Struktura mapy

Datová struktura pro načtení mapy je definována v hlavičkovém souboru `mapa.h` . V praxi se jedná o 2 různé struktury – jedna pro trasové body a druhá pro samotné cesty. Jejich složení nyní rozeberu.

8.3.1.1 Struktura trasových bodů

Na obrázku 8.3 je zobrazena definice typu struktury pro uložení jednotlivých trasových bodů. Skládá se z identifikátoru `id`, který odpovídá ID bodu, jak je uvedeno v mapovém souboru. Další *integerovou* položkou struktury je proměnná `xroad`, která indikuje, zda je trasový bod křižovatkou nějakých cest (==1) či nikoli (==0).

Dvě *floatové* hodnoty `lat` a `lon` odpovídají GPS souřadnicím daného bodu, tedy jeho zeměpisné šířce a délce. Vzhledem k faktu, že mapa dle specifikací JOSM popisuje souřadnice ve formátu pouze stupňů (např: **`lat="49.227862" lon="16.595517"`**), je zvolený datový typ asi nejvhodnější. A to i přes svoji drobnou nepřesnost, která se může v řádu 10^{-6} vyskytnout, ale pro naše potřeby už je zanedbatelná (0,000001 odpovídá odchylce cca 10cm).

```
15 typedef struct {
16   int id;
17   float lat;
18   float lon;
19   int xroad;
20 } NODE;
```

obrázek 8.3: struktura pro uložení trasových bodů

Tento nový typ struktury je pojmenován `NODE` . V hlavičkovém souboru jsou dále definovány i proměnné tohoto typu – pole trasových bodů `nodes[]` . Poslední pomocnou proměnnou pro práci s trasovými body je čítač `nd` , který uchovává přesný počet načtených trasových bodů.

Menší zmatek vzniká při práci s identifikátory jednotlivých trasových bodů. Je totiž nutné velmi pečlivě odlišovat ID bodu dle mapy (tak jak je uloženo ve struktuře) a index daného bodu v poli nodes`[]` . Typické jsou tedy podobné případy: **`nodes[26].id = 32`** . Uchovávat zdánlivě

zbytečný identifikátor `id` je bohužel nezbytně nutné, neboť je to jediné pojítko mezi cestami a body, které na nich leží (dle popisu JOSM).

8.3.1.2 Struktura cest

Druhá struktura obsahuje záznamy o celých cestách, její definici zachycuje následující obrázek.

```
25 typedef struct {
26   int id;
27   int end1;
28   int end2;
29   int wp;
30   int wpts[16];
31   int foot;
32 } WAY;
```

obrázek 8.4: struktura pro uložení záznamů o cestách

Stejně jako u jednotlivých bodů, i u cest se uchovává jejich originální identifikátor v proměnné `id` . Zde už však slouží pouze pro lepší orientaci při práci, testování a kontrolních výpisech. Další položkou struktury je proměnná `foot,` která rozlišuje parkové cesty zajímavé pro závod (== 1) od všech ostatních (==0), jak už bylo naznačeno dříve.

Zbylé čtyři položky zaznamenávají odkazy na jednotlivé body konkrétní cesty. Proměnné `end1` a `end2` obsahují idnex do pole `nodes[]` na okrajové body cesty (křižovatky). Pole `wpts[]` uchovává indexy všech dalších trasových bodů, které se na cestě vyskytují a proměnná `wp` je potom čítačem jejich počtu. Například tedy cesta, která se skládá ze 3 trasových bodů, bude uložena jako dva body okrajové plus jeden vnitřní (podobně jako na obrázku 8.2).

Nový typ struktury je pojmenován WAY a podobně jako tomu bylo v prvním případě, i zde jsou definovány proměnné tohoto typu v poli cest – `ways[]` a čítač `wa` obsahující celkový počet načtených cest.

Jedinou podmínkou správného načtení celé cesty, včetně všech jejích bodů, do struktury je požadavek, aby tyto body v době načítání cesty již byly načteny ve struktuře bodů. O správný převod načtených identifikátorů trasových bodů na jejich indexy z pole `nodes[]` se stará poslední funkce tohoto modulu – `int pozice(int id)`. Vzhledem k faktu, že načítání probíhá sekvenčně od začátku do konce souboru, tento požadavek znamená, že v mapovém souboru musí být vždy nejdříve uloženy příslušné body a až potom cesta z nich sestavená. Toto pořadí je ale logické a v mapách bývá zachováno, proto nemyslím, že by s tímto měly být jakékoli problémy. O konkrétní použité mapě už ale více v následující kapitole.

8.3.2 Soubor s mapou

Vzhledem k faktu, že během tvorby této diplomové práce nebyly ještě na webových stránkách soutěže k dispozici žádné oficiální mapy, byl jsem nucen si pro návrh a testování programu vytvořil vlastní mapový podklad.

Vytvořená mapa pokrývá území menšího parku na Božetěchově ulici v Brně. Toto místo jsem nakonec zvolil záměrně, neboť se nachází v bezprostřední blízkosti naší školy a tedy je nejvhodnějším kandidátem na testování celého systému. Nakonec tedy byla tvorba vlastní mapy v jistém směru i výhodou.

Mapa se nachází v souboru `mapa.xml` (aplikace počítá s tímto jménem a očekává soubor ve stejné složce jako svůj spouštěcí soubor). Mapový soubor přesně dodržuje požadavky a formu danou formátem JOSM tak, jak byla naznačena i v kapitole 4.

Plánek pokrytého území i s vyznačenými křižovatkami a cestami (včetně ID) je k nahlédnutí v příloze 1. Na jednotlivých cestách jsou křížky vyznačeny i jejich trasové body.

8.4 Souřadnice cíle

Po úspěšném načtení mapy ze souboru přichází na řadu první důležitá interakce s uživatelem. Tímto krokem je zadání cíle závodu robotovi, které bude v praxi probíhat asi jen pouhých pár minut před začátkem závodu.

Tuto část programu zabezpečuje modul, jehož zdrojové kódy se nachází v souboru `GPS.c` , zejména potom jeho funkce `cil()`. V této části se dále nachází i dvě funkce, které jsou dále hojně využívány pří zpracování cílových souřadnic, zjišťování startovní pozice a hlavně při samotném řízení robota. Rád bych jimi tedy proto začal.

8.4.1 Vzdálenost a azimut

Jak již nadpis napovídá, jedná se tedy o funkce zajišťující výpočet azimutu a vzdálenosti mezi dvěma různými trasovými body.

Obě funkce - `int vzdalenost(int A, int B)` pro výpočet vzdálenosti a `int azimut(int A, int B)` pro stanoveni azimutu očekávají jako své parametry indexy příslušných trasových bodů z pole `nodes[]`. Návratem funkcí je potom údaj o vzdálenosti v centimetrech nebo úhel azimutu v celých stupních. Tyto jednotky byly zvoleny jako nejvhodnější a dostatečně přesné měřítko pro naše podmínky.

Obě funkce pracují přesně podle vzorců uvedených v kapitolách 5.1 a 5.2. Pro správný výpočet je vždy nutné nejdříve převést vstupní údaje o zeměpisné šířce a délce bodů ze stupňů na radiány (rovnice 3). Dále výpočet probíhá podle daných vzorců za použití příslušných goniometrických funkcí ze standardní knihovny `math.h` .

$$rad = \deg * \frac{\pi}{180°} \qquad \deg = rad * \frac{180°}{\pi} \tag{3, 4}$$

Při výpočtu vzdálenosti pracujeme s poloměrem Země v jednotkách centimetrů, aby nám takto vyšla i výsledná hodnota vzdálenosti. Po výpočtu azimutu je třeba převést výsledný úhel **Θ** zpět z radiánů na stupně (rovnice 4) a dále normalizovat výstupní rozsah z -180° .. +180° na standardní a praktičtější 0° .. 360°. To je realizováno pomocí jednoduchého vztahu za použití funkce modulo:

```
azimut = ( Θ + 360 ) % 360
```
(5)

8.4.2 Zadání cíle

Nyní už ale zpět k vkládání cílových koordinátu a funkci `cil()`. Ta hned ze svého úvodu vyzve uživatele k výběru formátu, v jakém chce pozici cíle zadat (viz. obrázek 8.5).

Na výběr jsou tři možnosti. První dvě nabízí možnost zadat GPS souřadnice ve dvou nejrozšířenějších formátech – tedy stupňovém (tak jako je použit i ve standardu JOSM) a kombinaci stupňů a minut (známý hlavně mezi příznivci turistiky, outdoor sportů, GeoCachingu, ale i z autonavigací).

Třetí možnost je potom účelově zvolena pro testování a ulehčení při použití v závodě. Umožňuje zadat přímo ID konkrétního trasového bodu odpovídající popisu v mapovém souboru závodního prostředí.

Uživatel zde má také možnost celý program okamžitě ukončit.

```
Nejprve prosim zadejte souradnice cile zavodu.
Vyberte format souradnic:
  0 - N DD.DDDDDD E DDD.DDDDDD
  1 - N DD°MM.MMM E DDD°MM.MMM
  5 - pro prime zadani ID bodu cile
  X - pro ukonceni programu
Vase volba: _
```

obrázek 8.5: výběr formátu souřadnic

Pokud nebyl ukončen, pokračuje algoritmus dále a dle uživatelovy volby volá jednu z dalších funkcí, která se postará o správné načtení údajů. Tyto načítací funkce používají při svém běhu ještě jednu pomocnou funkci z modulu časovače (`timer.c`), který je popsán v kapitole 8.5.

Tato pomocná funkce `int ctiCislo(int min, int max)` zpracovává vstup z klávesnice v nekonečném cyklu tak dlouho, dokud není načtena číselná hodnota v rozmezí jejích vstupních parametrů `min` a `max`. Hodnota stisknuté číselné klávesy je potom funkcí navrácena.

8.4.2.1 Formát N DD.DDDDDD E DDD.DDDDDD

Při zvolení možnosti `'0'` je volána funkce `deg()`, která se postará o správné načtení souřadnic ve formátu stupňů. Uživatel zadává jednotlivé číslice zvolených souřadnic, o formátování a oddělovače se stará algoritmus sám (viz. obrázek 8.6).

Jak již bylo řečeno, tento tvar souřadnice přímo odpovídá formátu popsanému standardem JOSM a také formátu, který jsem zvolil za referenční pro tuto práci, neboť má nejblíže k datovému typu `float`, který jsem si zvolil pro ukládání údajů o souřadnicích.

Načtené údaje už tedy není třeba nijak dále upravovat a mohou být uloženy do globálních proměnných `latCil` a `lonCil` pro další zpracování.

```
Zadejte souradnice cile ve formatu N DD.DDDDDD E DDD.DDDDDD.
Souradnice cile: N 49.195133 E 016.608_
```

obrázek 8.6: zadání souřadnic ve tvaru stupňů

8.4.2.2 Formát N DD°MM.MMM E DDD°MM.MMM

Zvolení možnosti `'1'` vede k volání funkce `mnt()`, která umožňuje načtení souřadnic ve tvaru stupňů a minut. Stejně jako v předchozím případě, se algoritmus stará o všechno formátování a doplňuje do zápisu správnou šablonu (obrázek 8.7). Uživatel tedy opět může bez váhání zadávat jen správná čísla.

```
Zadejte souradnice cile ve formatu N DD°MM.MMM E DDD°MM.MMM.
Souradnice cile: N 49°11.708 E 016°3_
```

obrázek 8.7: zadání souřadnic ve tvaru stupňů a minut

V tomto případě je dále nutné, aby byly načtené souřadnice před uložením do globálních proměnných přepočteny na referenční tvar dle vzorce:

$$DD.dddddd° = DD + (MM.mmm / 60)° \quad (6)$$

Obě předchozí funkce počítají s operativností robota na severní a zároveň na východní polokouli, čemuž odpovídají i šablony, do kterých uživatel souřadnice zadává. V případě potřeby by nebyl problém pozměnit zdrojové kódy, ale v době této práce se to zdá být velmi nepravděpodobné.

Po načtení souřadnic je třeba zjistit, kde se nachází na mapovém podkladu. O to se stará další funkce modulu – `int find(float lat, float lon)`. Ta systematicky prochází jednotlivé trasové body v poli `nodes[]` a hledá ten nejbližší k zadaným souřadnicím. Tento trasový bod je poté definitivně zvolen za cíl závodu a jeho index je uložen do globální proměnné `cilID` .

Zde vycházíme z předpokladu, že se cíl závodu skutečně bude nacházet v místě některého z bodů mapy. Z předchozích ročníků plyne zkušenost, že typickými místy pro start i cíl závodu jsou křižovatky parkových cest. Tento předpoklad nám tak ušetří případné složité dopočítávání dalších bližších bodů na jednotlivých trasách.

8.4.2.3 Zadání ID bodu cíle

Konečně poslední možností nabídky – `'5'` – volá funkci `nid()`. Ta vyzve uživatele k zadání číselného identifikátoru cílového bodu (obrázek 8.8), který se skládá z 8 číslic, ale volbu lze předčasně potvrdit klávesou *Enter*.

Po načtení už u této varianty není třeba nic přepočítávat ani vyhledávat. Algoritmus pouze zkontroluje, za se zadaný bod v načtené mapové struktuře skutečně vyskytuje. Pokud ne, oznámí to uživateli a celý program se ukončí. Pokud ano, zapíše se jeho index do proměnné `cilID` a funkce navrátí řízení.

```
Zadejte ID cislo ciloveho bodu (JOSM node ID).
ID cile: 26
```

obrázek 8.8: zadání ID bodu cíle

8.4.2.4 Křižovatka

Po načtení souřadnic a volbě cílového trasového bodu je řízení vráceno funkci `cil()` a ta má za úkol vyřešit jeden poslední problém. Pro správnou funkci dalších algoritmů pro plánování trasy (více v kapitole 8.8) je nezbytně nutné, aby byl cílový bod ve struktuře označen jako křižovatka. Tato podmínka plyne ze samotného principu algoritmu UCS (kap. 7.2) a právě v tomto místě algoritmu ji zajišťuje další z funkcí tohoto modulu – `krizovatka(int node)`.

Tato funkce může detekovat 3 různé situace, které dále rozeberu. První z nich je fakt, že cílový bod je skutečně křižovatkou parkových cest. V tomto případě funkce okamžitě končí a vše je tedy v pořádku a připraveno pro budoucí výpočty.

Druhý případ nastává, pokud zvolený trasový bod neleží na žádné ze známých cest načtené mapové struktury. V tomto případě je tato skutečnost oznámena uživateli a program svoji práci ukončí.

Poslední možností je situace, kdy bod sice leží na některé z cest, ale není křižovatkou. K řešení tohoto problému vedou 2 kroky. Prvním je označení tohoto bodu jako křižovatky. Druhým krokem je potom vytvoření dvou nových cest v mapě, které překrývají cestu původní a vzniknou jejím rozdělením v místě tohoto bodu. Algoritmus zajišťuje správné rozdělení a překopírování všech trasových bodů původní cesty do nových struktur.

Po skončení procesu se řízení vrací funkci `cil()` a protože tímto proces zadávání cílových údajů končí, vrací se celé řízení funkci `main()`. K tomuto modulu se ale ještě v kapitole 8.7 vrátíme, neboť obsahuje i další funkce používané při úvodní lokalizaci po startu robota.

8.5 Časovač

Druhou velmi důležitou informací od uživatele je stanovení času startu závodu. Dle pravidel soutěže (kap. 2.2) je také nutné, aby byl robot schopen ve stanovený čas odstartovat samostatně. O tuto záležitost se stará modul časovače obsažený ve zdrojovém kódu souboru `timer.c` .

Po načtení a zpracování cílových souřadnic je tedy volána další funkce programu, která se postará o správné odstartování robota, tou je funkce `timer()`.

8.5.1 Zadání času startu

Stejně jako při zadávání cílových koordinátů, i zde program nabídne více možností (obrázek 8.9). Uživatel může zadat buď přesný čas startu (s přesností na minuty) nebo může zadat tzv. offset – tedy dobou (v minutách) za kterou závod začne.

I v tomto momentě je možné program okamžitě ukončit stisknutím klávesy `'X'`.

```
Zadejte prosim cas startu zavodu.
Stisknete:
  0 - pro zadani presneho casu startu
  1 - pro zadani casoveho offsetu do startu
  X - pro ukonceni programu
Vase volba: _
```

obrázek 8.9: výběr způsobu zadání času startu

Dle uživatelovy volby je volána jedna z následujících funkcí, která se postará o správné zadání časových údajů. I tyto procedury využívají pomocnou funkci `ctiCislo` , která byla zmíněna již výše v tomto textu.

8.5.1.1 Přesný čas startu

Pokud zvolíme `'0'` – přesný čas startu, je volána funkce `cas()`. Ta vyzve uživatele k zadání času ve tvaru HH:MM (obrázek 8.10) a dále se stará o jeho správné načtení, včetně automatického doplnění oddělovače.

Následně je využito standardní knihovny `time.h` ke zjištění aktuálního systémového času. Pokud by byl zadaný čas toho dne již minulostí, program to ohlásí uživateli a ukončí svoji činnost. V opačném případě jsou časy od sebe odečteny a výsledek převeden na minuty – tedy počet minut

zbývajících do startu závodu. Tento údaj je uložen do globální proměnné `ta` pro zahájení odpočtu (kapitola 8.5.2) a funkce navrací řízení.

```
Nyni prosim zadejte presny cas zacatku zavodu ve tvaru HH:MM.
Cas startu: 17:4_
```

obrázek 8.10: zadání přesného času startu

8.5.1.2 Časový offset

Druhá volba – `'1'` – nám umožňuje zadat dobu, za kterou závod začne. Volaná funkce `offset()` očekává na vstupu údaj v minutách (obrázek 8.11). Je zde možné zadat až tříciferné číslo a nebo volbu kdykoliv potvrdit klávesou *Enter*.

Zde již není třeba nic zjišťovat ani převádět. Údaj je rovnou uložen do globální proměnné `ta` a funkce svoji činnost končí.

```
Nyni prosim zadejte, za jak dlouho zavod zacne (v minutach).
Doba do startu: 35_
```

obrázek 8.11: zadání časového offsetu

8.5.2 Odpočet

V tomto momentě je řízení předáno poslední funkci v tomto modulu. Tou je funkce `odpocet()`, která se postará o správné načasování startu.

Jak plyne z předchozích odstavců, v proměnné `ta` máme uloženu dobu do startu závodu. Pokud je tato doba větší než 30 minut program to oznámí uživateli a sám se ukončí. Jedná se možná o zbytečný test, neboť se nepředpokládá, že by tato hodnota mohla před závodem být větší než 15 minut, ale v praxi by bylo nepraktické a nešetrné k bateriím a celému systému aby stál delší dobu zbytečně zapnutý v nečinnosti. Navíc by zde mohlo dojít k odhalení možné chyby při zadávání času (uživatel se spletl o hodinu a podobně).

Po tomto testu program ještě jednou a naposledy vypíše uživateli všechny zadané údaje (obrázek 8.12) a vyzve ho k potvrzení – `Enter` nebo zrušení – `'X'` startu.

```
Zavod zacina za 10 minut.
Souradnice cile: N 49.228397 E 016.597427
Pro zruseni startu stisknete X, pro zahajeni odpoctu stisknete ENTER!
```

obrázek 8.12: závěrečná výzva

Po potvrzení této výzvy je zahájen finální odpočet do startu závodu. Program toto oznámí na standardní výstup a za oznámení vypíše každých 30 vteřin znak tečky `'.'` jako odezvu odpočtu. Po uplynutí daného času je ještě oznámeno posledních 20 vteřin do startu a na obrazovku vypsán jeden symbol hvězdičky za každou tuto vteřinu (obrázek 8.13).

Následně program oznámí start závodu a řízení je vráceno zpět až do funkce `main()`.

```
Byl zahajen odpocet . . . . . .

Zbyva poslednich 20 vterin do startu: ********** START!!!
```

obrázek 8.13: odpočet startu

Tímto končí jakákoliv interakce robota (programu) s uživatelem. Od této chvíle už algoritmus vykonává všechny další kroky samostatně tak, aby dovedl robota co nejefektivněji do cíle. Výjimkou jsou pouze informativní výpisy indikující činnost dalších algoritmů a dále informace o průběhu navigace (řízení) a varování ohledně případných překážek na trati.

8.6 Komunikace

Než budu pokračovat v chronologickém popisu dále, musím udělat menší odbočku. Pro správnou práci dalších funkcí musím představit jeden trochu speciální modul. Ten se ukrývá ve zdrojovém souboru `COM.cpp` a stará se o veškerou komunikaci programu s externími zařízeními. Tedy s připojenou GPS a dále samozřejmě se samotným robotem.

8.6.1 Rozhraní spojení

Tato zařízení jsou připojena pomocí rozhraní USB, ale v operačním systému je připojení simulováno jako klasický sériový COM port. Důvodem je hlavně fakt, že samotný robot komunikuje právě přes toto sériové rozhraní (standard RS-232), ale použitý notebook není tímto rozhraním bohužel vybaven.

U použité GPS je to potom jediná cesta, jak číst data z vlastní aplikace (oficiální Garmin aplikace komunikují přímo přes USB rozhraní). Při implementaci se tato varianta nakonec ukázala výhodnou, neboť nebyl problém kontrolovat správnost komunikace pomocí systémového hyperterminálu.

obrázek 8.14: zapojení periferií

8.6.2 SFile

V operačním systému Windows je komunikace přes sériovou linku realizována jako čtení a zápis do speciálního druhu „souboru“. Protokol je ale dosti složitý a podrobně je popsán na webových stránkách Microsoft MSDN.

Mně se podařilo najít jednoduchou knihovnu napsanou v jazyce C++, která zařídí veškerá potřebná nastavení spojení a dalších náležitostí pro správnou komunikaci (viz. [9]). Zdrojové kódy knihovny jsou obsaženy v souborech `SFile.cpp` a `SFile.h` . Při použití této knihovny už je práce se sériovou linkou velmi příjemná a jednoduchá. Ve vlastním kódu lze definovat proměnnou typu `SFile`, pro další příklad ji označme `sf` .

Otevření souboru odpovídajícímu sériovému portu se provede voláním funkce `sf.Open();`. Poté již je možné z linky číst jednotlivé znaky pomocí funkce `sf.ReadByte();` případně zapisovat znaky zpět voláním `sf.WriteByte();`. Po ukončení komunikace soubor uzavřeme funkcí `sf.Close();`.

Můj komunikační modul tedy spolupracuje s touto knihovnou a za použití uvedených 4 funkcí zajišťuje prakticky veškerý tok dat mezi notebookem, robotem a GPS. Zdrojový kód samozřejmě obsahuje celé funkce určené k ovládání periferií, ale k jejich podrobnějšímu rozboru se budu vracet až později v textu na místech jejich použití (kapitoly 8.7 a 8.9).

8.6.3 Nastavení sériové linky

Poslední detail, který je třeba si spojit se sériovým portem, je nastavení komunikačních parametrů linky. Základních parametrů, které nás zajímají, je 5. Jsou to: rychlost komunikace (v bitech za sekundu), počet datových bitů, parita, počet stop-bitů a řízení toku. Tato nastavení jsou pro obě použitá zařízení různá a jejich správné hodnoty shrnuje následující tabulka.

	robot	**GPS**
rychlost	57600	4800
datové bity	8	8
parita	lichá	žádná
stop-bity	1	1
řízení toku	žádné	žádné

tabulka 8.15: parametry sériové komunikace

Tyto hodnoty jsou definovány také v hlavičkovém souboru tohoto modulu – `COM.h`. Aby je bylo možné správně používat, bylo třeba poupravit i funkci `Open()` z použité knihovny `SFile`.

8.7 Lokalizace startu

Po rozboru komunikačních kanálů se vrátíme zpět k robotově závodu, kde je hned využijeme v praxi. Kapitolu 8.5 jsme uzavřeli v momentě, kdy robot úspěšně odstartuje závod a dle rozboru řešení víme, že prvním jeho úkolem bude zorientovat se na načtené mapě a stanovit svoji startovní polohu.

Za tímto účelem se opět vrátíme k modulu ze souboru `GPS.h` (jak již bylo v jeho popisu avizováno) a to konkrétně k jeho poslední proceduře – `fix()`.

8.7.1 GPS

Nejdříve je třeba načíst data z připojeného GPS přístroje. O to se stará funkce `readGPS()` komunikačního modulu (viz. předchozí kapitola). Ta inicializuje komunikaci s navigační periferií a v deseti cyklech z ní přečte věty ve formátu NMEA.

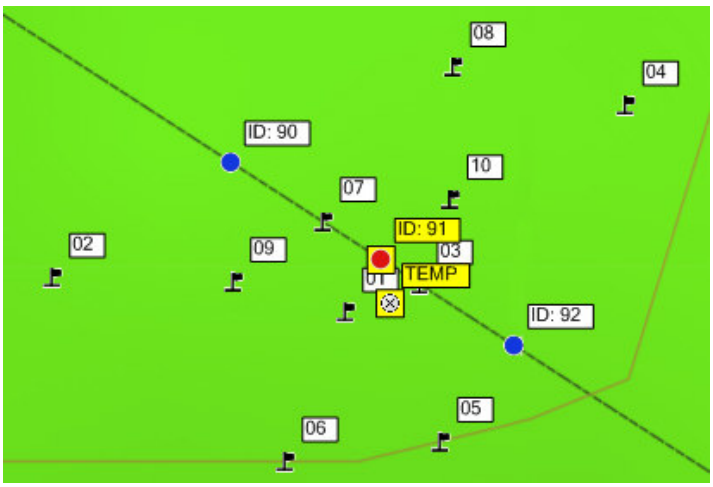
Tyto řetězce jsou rovnou zpracovávány. Jsou z nich extrahovány potřebné informace (aktuální souřadnice) a tyto jsou ukládány do polí `lat` a `lon` (definována v souboru `GPS.h`). Každé úspěšné přečtení a uložení dvojice souřadnic je indikováno výpisem symbolu hvězdičky.

Po ukončení desátého cyklu je komunikační soubor uzavřen a řízení se vrací funkci `fix()`.

8.7.2 Nalezení startovní pozice

Naměřené souřadnice jsou dále zprůměrovány do jednoho dočasného bodu a velmi podobně, jak tomu bylo v případě zadávání cílových souřadnic (kapitola 8.4), je i zde nutné určit odpovídající nejbližší bod na některé z cest v mapě. Celý proces zachycuje obrázek 8.16.

Opět je tedy volána funkce `int find(float lat, float lon)` a odpovídající index je uložen do globální proměnné `startID`. Tento startovní trasový bod je dále také testován funkcí `krizovatka(int node)`, aby byl zajištěn bezproblémový chod následujícího plánovacího algoritmu. Po jejím průběhu se řízení opět vrací funkci `main()`.



obrázek 8.16: aproximace bodu (TEMP = zprůměrované měření 01-10; ID: 91 = startovní bod)

8.8 Algoritmus UCS

V tuto chvíli už jsou známy pozice startovního i cílového bodu a program může směle přejít do fáze plánování trasy. Odpovídající modul se nachází ve zdrojovém souboru `route.c` a stejně tak i procedura, která má nalezení cesty na starosti se nazývá `route()`.

Tato procedura pracuje přesně podle pravidel algoritmu UCS (popsaný v kapitole 7.2). Po alokaci potřebné paměti a inicializaci pracovních polí (popsáno dále) jsou cyklicky volány jednotlivé pomocné funkce, dokud není nalezeno nejlepší správné řešení (cesta). Průběh procedury popisuje následující úsek odpovídajícího zdrojového kódu.

```
void route()
{ ...
  alokuj();                  // alokace prac. prostoru
  inicializuj();             // inicializace prac. prostoru
  while(1) {                 // cyklus algoritmu UCS
    best = nejlepsiRadek();  // aktualni nejlepsi cesta
    if (UCS[best][posledniNaRadku(best)] == cilID)
      break;                 // nalezeno reseni -> konec cyklu
    expandujRadek(best);     // expand nejlepsiho radku
    skrtniNavraty();         // skrtani reseni s cestami..
    skrtniDuplicity();       // ..zpet a s duplicitni reseni
  }
  sestavTrasu(best);         // sestaveni vysledne trasy
  uvolni();                  // uvolneni prac. prostoru
  return;                    // navrat
}
```

obrázek 8.17: procedura route()

Jednotlivé funkce a principy jejich řešení jsou podrobně popsány a rozebrány v následujících podkapitolách.

8.8.1 Příprava pracovního prostoru

Prvním úkolem procedury je příprava potřebného pracovního prostoru pro algoritmus UCS. Tato příprava obnáší alokaci dvou dynamických polí. Jedno z těchto polí reprezentuje graf cest a křižovatek závodního parku, druhé pole potom slouží k uchovávání záznamů průběhu výpočtu algoritmu UCS a prakticky jeho celé činnosti.

K určení správné velikosti těchto polí je třeba nejdříve zjistit počet všech bodů načtené mapové struktury, které jsou označeny jako křižovatky (zahrnuje i startovní a cílový bod). Jejich počet je uložen do globální proměnné `xroads` a navýšen o `1`.

8.8.1.1 Alokace, inicializace a uvolnění

O správnou práci s pamětí se starají funkce volané na začátku - `alokuj()` a na konci - `uvolni()` tohoto modulu. Jejich těla jsou velmi jednoduchá, obsahují pouze *for*-cykly k vytvoření / uvolnění obou polí o správné velikosti (viz. další dvě kapitoly). K tomu jsou využity standardní funkce jazyka C pro práci s pamětí – `malloc()` a `free()`.

Během inicializace - funkce `inicializuj()` – jsou obě pole (opět v několika cyklech) naplněny úvodními hodnotami dle popisu následujících dvou kapitol. Řízení je potom vráceno proceduře `route()` a výpočet trasy může začít.

8.8.1.2 Graf mapy

První pole tedy představuje hranově ohodnocený graf reprezentující načtenou mapovou strukturu. Jedná se o dvojrozměrné pole prvků typu *integer* nesoucí označení `graf[][]` a jeho výška i šířka jsou stanoveny na hodnotu `xroads` .

Při prvním inicializačním průchodu je první řádek a první sloupec pole naplněn indexy všech křižovatek mapy (proto jsou rozměry o 1 větší). V druhém průchodu je potom pro každou dvojici křižovatek (zbylá plocha pole) volána funkce `int spoj(int nd1, int nd2)`, která zjišťuje, zda mezi danými dvěmi křižovatkami existuje spojovací cesta. Pokud je taková cesta nalezena, na příslušné místo v poli `graf[][]` je uložena hodnota odpovídající délce této cesty. V opačném případě je zde uložena hodnota `-1`.

Tyto délky cest jsou dále potřebné pro činnost algoritmu UCS. Protože s nimi ale algoritmus pracuje pouze jako s *fitness* hodnotami pro výběr nejlepšího řešení, není nutné aby tyto hodnoty odpovídaly skutečné délce příslušných úseků v reálném světě. Rozhodl jsme se proto, že pro stanovení délky jednotlivých cest využiji znalosti počtu jednotlivých jejich trasových bodů. Vycházím z faktu, že body jsou v mapovém popisu rovnoměrně rozmístěny a jsou mezi nimi přibližně stejně velké vzdálenosti, čímž zůstane měřítko zachováno. Délkou cesty pro účely algoritmu UCS je tedy hodnota z proměnné `ways[].wp` dané cesty.

-	1	2	3	4	5	6	7	8	9	10	11
1	-	-	-	-	-	-	-	-	-	-	-
2	-	-	2	-	-	-	-	-	-	-	-
3	-	2	-	5	-	2	-	-	5	-	-
4	-	-	5	-	2	-	-	-	-	5	-
5	-	-	-	2	-	1	1	-	-	-	-
6	-	-	2	-	1	-	-	1	-	-	-
7	-	-	-	-	1	-	-	1	-	2	-
8	-	-	-	-	-	1	1	-	2	-	-
9	-	-	5	-	-	-	-	2	-	5	-
10	-	-	-	5	-	-	2	-	5	-	2
11	-	-	-	-	-	-	-	-	-	2	-

obrázek 8.18: příklad výpisu pole graf[][]

8.8.1.3 Seznam UCS

Druhým polem je seznam uzlů algoritmu UCS (v teorii ounačený jako *Open*). Opět se jedná o dvojrozměrné pole, jehož šířka je rovna hodnotě `xroads`. Výška pole je tentokrát mnohonásobně větší – stanovena je na `xroads^2` , což by měla být dostatečná velikost pro jakýkoliv graf mapy. Jako nejnáročnější byl přitom počítán případ, kdy by mapa měla propojenu každou křižovatku se všemi zbylými a optimální cesta by vedla přes všechny křižovatky.

Každý řádek pole přitom odpovídá jednomu zpracovanému uzlu algoritmem UCS. Na jeho první pozici je hodnota odpovídající celkové ceně cesty daného uzlu a za ní následuje řetězec indexů příslušných křižovatek této trasy. Pokud je na první pozici hodnota -1 znamená to, že daný uzel byl ze seznamu vyřazen (expandován nebo „škrtnut“).

Během inicializace je celé pole vyplněno hodnotami -1 a následně jsou na první dvě pozice prvního řádku zapsány hodnoty `0` a `startID` . Tak vznikne první uzel seznamu *Open* určený k expandování.

```
-1 |  2  -  -  -  -  -  -  -  -  -  -
-1 |  2  3  -  -  -  -  -  -  -  -  -
 2 |  2 10  -  -  -  -  -  -  -  -  -
 8 |  2 11  -  -  -  -  -  -  -  -  -
-1 |  2  3  1  -  -  -  -  -  -  -  -
 7 |  2  3  4  -  -  -  -  -  -  -  -
 4 |  2  3  6  -  -  -  -  -  -  -  -
 7 |  2  3  9  -  -  -  -  -  -  -  -
```

obrázek 8.19: příklad výpisu pole UCS[][]

Výpis aktuálního stavu obou polí je kdykoliv možný voláním funkce `vypis()`. Formát výstupu naznačují obrázky 8.18 a 8.19. Defaultně je výpis nabídnut po sestavení celé cesty ke konci tohoto modulu. Tato nabídka je spíše pro zajímavost a testování, neboť během závodu už její volbu uživatel nemůže potvrdit. Po 3 vteřinách proto program pokračuje plynule dál.

8.8.2 Hlavní cyklus UCS

Po inicializaci potřebných polí už může být spuštěn hlavní cyklus celého algoritmu. V prvním kroku cyklu je dle popisu vybrán nejlepší z uzlů seznamu `UCS[][]`. Za tento výběr odpovídá pomocná funkce `int nejlepsiRadek()`, která projde celé toto pole a navrátí index řádku s nejnižší číselnou hodnotou na první pozici.

Další pomocnou funkcí, která je využita v tomto cyklu, ale i na jiných místech modulu, je funkce `int posledniNaRadku(int radek)`, která vrací index pozice posledního prvku na řádku pole `UCS[][]`, jehož index je předán v argumentu. Tato informace je důležitá, neboť poslední prvek řetězce je používán při expanzi uzlů a hlavně při kontrole ukončovací podmínky celého cyklu.

Pokud algoritmus zjistí, že vybraný nejlepší uzel má na své poslední pozici hodnotu odpovídající proměnné `cilID` (a na své první hodnotu `startID`, která tam je už od inicializace a nikdy se nezmění) znamená to, že žádaná cesta byla nalezena a řídící cyklus se přeruší. Tento řádek potom obsahuje řetězec indexů všech křižovatek od startu až do cíle. Následně je volána poslední funkce modulu - `void sestavTrasu(int radek)`, která se postará o sestavení kompletní trasy. Její popis je uveden v kapitole 8.8.3. Argumentem při jejím volání je pochopitelně index onoho „vítězného“ řádku.

V případě, že vybraný uzel na své poslední pozici neobsahuje index cíle, je třeba v cyklu pokračovat, daný uzel expandovat a jeho potomky uložit na další řádky pole `UCS[][]`. O to se starají následující tři funkce.

8.8.2.1 Expanze uzlu

O samotné expandování uzlu se stará funkce `void expandujRadek(int radek)`. Ta vezme poslední prvek daného řádku a nalezne všechny možnosti (trasové body), kterými se z tohoto místa dá pokračovat. Pro každý takový nalezený bod je potom v poli `UCS[][]` vytvořen nový řádek.

Na tyto nové řádky je vždy zkopírován řádek původní a na jeho konec je zapsán index příslušného následujícího trasového bodu. Na závěr je aktualizován první prvek řádku – přičtena délka přidaného úseku cesty.

Expandovaný řádek je následně označen za neplatný.

8.8.2.2 Neplatné uzly

Po vytvoření nových řádků je potřeba ze seznamu uzlů vyřadit ty, které tímto přestávají být relevantní a další chod algoritmu by pouze zpomalovaly.

První takovou skupinou jsou uzly, které vznikly v poslední expanzi a jejich poslední posun byl do bodu, kterým už odpovídající trasa procházela (tvoří smyčky). O odstranění těchto řádků se stará procedura `void skrtniNavraty()`.

Expanzí mohou vzniknout také uzly, jejichž poslední posun je do místa, do kterého vede už i jiný uzel. Z těchto duplicit je poté třeba vybrat tu horší variantu (delší cesta) a odstranit ji ze seznamu. O odstranění této skupiny neplatných uzlů se stará procedura `void skrtniDuplicity()`.

```
Planuji trasu: * * * * * * * * * * * * * * *
Plan trasy sestaven!
Prejete si vypsat grafy algoritmu UCS? (Ano/Ne): _
```

obrázek 8.20: Výstup algoritmu UCS

Každé jedno kolo řídícího cyklu je na výstupu indikováno symbolem hvězdičky. Celý výpis odpovídající průběhu nalezení a sestavení trasy je zobrazen na obrázku 8.20.

8.8.3 Sestavení trasy

Úkolem této procedury je sestavení celé výsledné trasy – bod po bodu, tak jak ji robot bude projíždět. V argumentu je funkci předán index „vítězného“ řádku jako výstup algoritmu UCS.

Za tímto účelem je spočtena celková délka cesty, tedy počet vnitřních trasových bodů (první položka na řádku) a k němu připočten počet křižovatek (zbylé položky řádku). Následně je alokováno pomocné dynamické pole `trasa[]` o spočtené velikosti. Toto pole je definováno v hlavičkovém souboru drive.h a je následně využíváno po celou dobu řízení robota při průjezdu trasou.

Algoritmus bere postupně křižovatku po křižovatce a indexy příslušných trasových bodů, které se mezi nimi nachází, kopíruje do vytvořeného pole. Poté, co zpracuje poslední bod, se řízení vrací funkci `route()`, kde je uživateli nabídnut výpis celého průběhu algoritmu (zmíněno již dříve). Nakonec je uvolněna všechna paměť využitá během výpočtu trasy a dále se řízení programu navrací až do funkce `main()`.

8.9 Řízení

Poslední část implementace s sebou přinesla veliké problémy. Už od registrace tématu této diplomové práce jsem věděl, že robota nebudu mít stále k dispozici. Výpůjčka domů nepřipadala v úvahu a navíc v zimním i letním semestru s robotem pracovali i další studenti, takže musel zůstat k dispozici. S robotem tedy bylo možné manipulovat a testovat jej v prostorách naší školy po předchozí domluvě a pod dohledem Ing. Rozmana.

Kromě implementace jsem také na robotovi podnikal několik hardwarových úprav (popsáno v kapitole 10). Při jedné z nich – v době, kdy jsem začal tento modul vytvářet a potřeboval jsem již s robotem aktivně pracovat – jsem ale zjistil vážné technické závady. S robotem nešla navázat komunikace a nereagoval ani na žádné signály. Tato skutečnost byla o to více zarážející, neboť při práci na svém semestrálním projektu jsem robota viděl pracovat naprosto bezproblémově.

Při hledání příčin problému bylo testováno komunikační rozhraní sériové linky i samotný řídící mikrokontrolér. Problém vyřešil až autor robota – kolega Novotný, který odhalil hned několik pochybení (chybné zapojení nepájivého pole, zastaralé verze robotova softwaru a ovladačů).

Náprava, která přišla až těsně před dokončením této práce, s sebou navíc přinesla i zásadní změnu ve formátu dat, v jakém robot zasílal informace ze svých senzorů. Nový protokol pracuje velmi nedeterministicky, údaje ze sonarů chodí zkreslené, údaje z kompasu zcela chybné a údaje z akcelerometru a odometrů chybí úplně.

Část své práce jsem proto vytvářel na simulovaných datech dle původního protokolu. Později jsem tedy byl nucen program pozměnit a přizpůsobit novým skutečnostem. Efektivita řešení (zvláště potom část pilotování) není díky těmto skutečnostem úplně ideální a bude vyžadovat další ladění a testování.

8.9.1 Spojení

Nyní již tedy k samotným algoritmům. Jak již bylo řečeno dříve, o komunikaci přes sériovou linku se starají funkce obsažené ve zdrojovém souboru `COM.cpp` a ani v tomto případě tomu nebude jinak. Modul obsahuje tři funkce. Jednu z nich pro čtení dat ze senzorů a dvě pro pohyb vpřed a otáčení robota.

8.9.1.1 Čtení ze senzorů

Pro čtení dat ze senzorů slouží funkce `int readRobot(int code)`. Jejím jediným parametrem je číselná hodnota reprezentující senzor, o jehož údaje máme zájem. Čísla `1-3` zde zastupují jednotlivé sonary – levý, středový a pravá (v tomto pořadí). Číslo `6` je potom kód pro čtení azimutu z elektronického kompasu. Čísla 4 a 5 jsou rezervována pro čtení ze dvou zadních sonarů. Vyšší čísla by bylo možné použít pro čtení dalších údajů při správné funkčnosti ostatních senzorů.

Funkce byla navržena tak, aby nejdříve přečetla celý jeden řádek ze sériové linky a dle parametru v něm nalezla žádaný údaj. Dle nového protokolu jsou ale informace ze senzorů zasílány odděleně po blocích ve tvaru: `$ID:value;` , kde `ID` je identifikátor senzoru (např: SFR = Sonar Front Right pro přední pravý sonar) a `value` je příslušná naměřená číselná hodnota. Tyto bloky jsou zasílány cyklicky ze všech senzorů na robotovi. Cyklus ovšem vykazuje značný nedeterminismus. Občas je senzor přeskočen úplně, někdy je blok odsazen na nový řádek, někdy navazuje bezprostředně za blokem předchozím.

Nové řešení proto načítá libovolný řetězec znaků o délce větší dvou cyklů, což by mělo být dostatečné i pro případ chyby v předaných datech či vynechání některého senzoru. V řetězci je dále dle parametru `code` nalezen odpovídající identifikátor a přečtena příslušná hodnota. Ta je na závěr funkcí navrácena.

I zde přinesl nový protokol několik novinek a proto jsou hodnoty před navrácením patřičně poupraveny. Vzdálenost je sice měřena v centimetrech - tak, jak bychom potřebovali - k její hodnotě je ale přičtena konstanta 10, kterou je potřeba zpětně odečíst. Azimut je přijímán jako celé číslo s přesností na desetinu stupně (tedy např. 1805 = 180,5°) a na požadovaný formát v celých stupních je před předáním přepočten (včetně zaokrouhlení).

Při každém volání této funkce je vždy třeba znovu otevřít a po jejím průběhu zase uzavřít komunikaci se sériovou linkou. Toto je nutné, protože chceme přijímat vždy aktuální data. Pokud bychom po otevření komunikace četli data vícekrát s časovým odstupem, byla by data v daný moment již irelevantní, neboť by se díky bufferování jednalo o data neaktuální.

8.9.1.2 Ovládání robota

K ovládáni robota slouží poslední dvě funkce komunikačního modulu. Pro pohyb vpřed je to funkce `void forward(int dist)`, jejímž jediným parametrem je vzdálenost (v centimetrech), jakou má robot urazit.

Druhou funkcí je funkce `void turn(int diff)`, která zajišťuje otáčení robota do požadovaného azimutu. Úhel natočení v celých stupních je předáván v parametru. Pro otočení vpravo je parametr kladný, pro otočení vlevo je to záporné číslo. Pokud by byla hodnota parametru větší než 360°, budou přebytečné periody odečteny. Pokud i po tomto odečtení zůstane hodnota větší než 180°, otočí se robot na opačnou stranu o velikost doplňkového úhlu.

Na sériovou linku je nejdříve posílán patřičný signál k pohybu – symbol `'i'`, `'j'` nebo `'l'`. Tento signál nastaví rychlost pohybu, přičemž rychlost kol při pohybu vpřed bude vyšší než při otáčení. Pro nastavení vyšší rychlosti je třeba zaslat signál opakovaně. Po započetí pohybu vyčká algoritmus patřičnou dobu (odpovídající zamýšlené vzdálenosti / úhlu natočení) a poté pošle signál k zastavení pohybu – symbol `'o'`.

Vzhledem k technickým komplikacím jsem už bohužel tyto rychlosti a časy nebyl schopen dostatečně otestovat a správně nastavit. K daleko lepší funkčnosti a přesnosti pohybu by také velmi pomohlo zprovoznění odometrů na kolech robota.

I tyto funkce při své činnosti vždy znovu inicializují a uzavírají komunikaci přes sériovou linku. Zde už to ale není z nutnosti aktuálnosti dat (neboť žádná nečteme), ale jednoduše to plyne z požadavku funkce předchozí.

8.9.2 Algoritmus řízení

Vlastní algoritmus starající se o správné řízení robota je obsažen v proceduře `drive()` zdrojového souboru `drive.c` a je zároveň poslední procedurou volanou z funkce `main()`.

Algoritmus pracuje nad dříve vytvořeným polem trasových bodů `trasa[]`. Pro každé dva za sebou jdoucí trasové body (jeden úsek trati) je nejdříve vypočtena jejich vzdálenost a azimut od jednoho k druhému. Spočtená vzdálenost je následně rozdělena na menší části o pevně stanovené délce (defaultně 32cm). Dokud robot neprojede celý jeden úsek od aktuálního trasového bodu k dalšímu, vnitřní cyklus algoritmu se postará o postupné projetí všech těchto částí. Vnější cyklus je potom zodpovědný za opakování tohoto procesu tak dlouho, dokud není robot v cíli závodu.

8.9.2.1 Průjezd jednotlivých částí

Každý průběh vnitřního cyklu (jedna část aktuálního úseku) se skládá z několika fází. V první fázi je provedena korekce azimutu robota. Ze sériové linky je přečten údaj o aktuálním natočení (funkce `readRobot()`), který je porovnán s vypočteným azimutem v daném úseku a robot je následně pootočen do správného směru pomocí funkce `turn()`.

V druhé fázi je zjištěno, zda je prostor před robotem volný alespoň na vzdálenost jednoho posunu. To obstarává pomocná funkce `int prekazka()` – více v následující kapitole. Pokud je trať volná, robot se posune o danou vzdálenost (funkce `forward()`) a cyklus se opakuje.

Po projetí všech částí daného úseku se robot nachází v následujícím trasovém bodě. Pokud tento už není bodem cílovým, vypočtou se údaje k dalšímu úseku cesty a opakuje se nové kolo vnějšího cyklu. V momentě, kdy robot dojede do cíle, je paměť pro pole trasových bodů uvolněna, řízení je navráceno funkci `main()` a program svoji činnost úspěšně končí.

8.9.2.2 Překážky

Tato pomocná funkce je volána před každým pohybem robota. Má za úkol zajistit, aby nedošlo k žádné kolizi s cizím objektem. V první fázi je pomocí známé funkce `readRobot()` středovým sonarem prověřen prostor před robotem. Pokud je zde dostatek místa (počítáno je s délkou zamýšleného pojezdu plus 10cm rezerva), funkce se navrací a robot může vyrazit.

Pokud je v cestě detekována překážka, robot se nepohne a v intervalu jedné vteřiny začne kontrolovat situaci před sebou. Po cca čtvrt minutě robot začne na překážku upozorňovat i zvukovým signálem a čeká na její odstranění. V momentě, kdy se tak skutečně stane a robot má volný „výhled“, pokračuje v pohybu, jak měl naplánováno.

V případě, že překážka nezmizí ani po více než minutě, zahájí robot úhybný manévr. Při něm je čteno z obou krajových sonarů a na stranu, kde je zdánlivě více místa, se robot vydá. Snaží se překážku objet po trase s pravoúhlými zatáčkami ve smyslu pohybu do strany, vpřed a do strany zpět tak, aby skončil v ose původně zamýšlené trasy, jen s překážkou za zády. Po celou dobu manévru opět kontroluje, zda je cesta možná. Kdyby náhodou nebyla, robot se vrátí po stejné cestě zpět a zkusí manévr zopakovat z druhé strany překážky. Po jejím úspěšném překonání je řízení vráceno funkci `drive()` a robot pokračuje v pohybu, jak bylo naplánováno.

Při jakémkoliv kontaktu s překážkami je průběh celého řešení situace oznamován na standardní výstup, jak ukazuje následující obrázek.

```
!!! POZOR !!!
Prekazka na trati . . . . . .
Odklidte prosim prekazku ..........
Zahajuji uhybny manevr.
```

obrázek 8.21: Upozornění na překážku

9 Testování

Činnost každého modulu a každé funkce byla v průběhu implementace důkladně testována. Modul pro načtení mapového souboru počítá se striktním dodržením formátu JOSM. Pokud bude tento dodržen, mapa bude do struktury načtena bez problémů. Příklad kódu je v příloze 2.

Moduly pro načtení cílových souřadnic a času startu jsou ošetřeny a testovány pro vstup neplatných znaků, krajní hodnoty a co to šlo, tak i pro nesmyslné vstupní hodnoty. Měly by zaručit, že pokud jsou data načtena, jsou v pořádku a relevantní.

Další moduly a funkce pro výpočet trasy, algoritmus UCS i operace s GPS souřadnicemi byly testovány a ověřovány pomocí odpovídajících výpočtů na papíře či jiných online nástrojů. Vzhledem k tomu, že do jejich chodu uživatel již nezasahuje, by měly nad korektními daty pracovat bezproblémově a univerzálně.

Komunikace s externími zařízeními by měla být také bez problémů. Při práci s GPS se jedná o rozšířený a standardizovaný protokol spojený s kvalitním zařízením. Bezchybný chod je tedy prakticky zaručen a během testů zařízení také ani jedenkrát neselhalo. U robota je spojení také stabilní, avšak nový protokol už není tolik spolehlivý. S případnými chybami ale program počítá a umí je řešit. Další testování už bylo opět bezproblémové. Pokud tedy nenastanou jiné nečekané technické problémy, měla by obě zařízení pracovat správně.

Modul řízení a vyhýbání se překážkám byl testován hlavně nad simulovanými daty, chybí zde tedy interakce s reálným prostředím a s ní i patřičné úpravy, které s sebou nasazení v terénu jistě přinese. Tyto úpravy by se ovšem týkaly hlavně drobností v řízení a přesnosti zasílání příkazů robotovi. Celkové řízení pohybu a její logika pracují správně.

Systém jako celek byl potom testován sadou mnoha různých více či méně smysluplných vstupů a situací. Program vždy skončil svoji činnost standardně, ať už dosažením požadovaného cíle, či chybovou hláškou odpovídající nestandardní situaci na kterou byl testován.

9.1 Kontrolní výpisy

Za účelem testování jsou na mnoha místech ve zdrojových kódech vloženy i řádky kontrolních výpisů na standardní výstup. Někdy byly dokonce pro tyto výpisy vytvořeny celé funkce.

Ty nejzajímavější jsou vypisovány v průběhu celé činnosti programu, další jsou uživateli defaultně nabídnuty k výpisu ve specifických momentech (načtená mapa, algoritmus UCS), či připraveny pro použití v některém módu simulace (viz. následující kapitola).

Zbylé jsou potom v kódu zakomentovány pro případné kontroly, kdyby byl program v budoucnu dále upravován. Zde se jedná hlavně o kontrolu správného načtení dat či výsledků výpočtu některých funkcí.

9.2 Možnosti simulace

Z nutnosti využití externích zařízení ke správné funkci programu a faktu, že ne vždy mohou být tato zařízení k dispozici, plyne i nutnost možnosti částečně simulačního režimu programu. Tuto potřebu jsem dále rozvinul tak, aby správným nastavením bylo při testování možné jednotlivé moduly programu nezávisle na sobě zapínat a vypínat.

Všechna tato nastavení se provádí modifikací funkce `main()`, konkrétně zakomentováním některých volání funkcí a odkomentováním jiných, dle možností popsaných v následujícím textu a podrobných pokynů, které se nacházejí v komentářích na odpovídajících místech přímo v kódu.

Prvním modulem, který můžeme vypnout, je dialog pro zadávání cílových souřadnic (volání funkce `cil()`). Číslo ID zvoleného cílového trasového bodu (dle popisu mapy JOSM) je možné zadat přímo v kódu dle pokynů v komentářích. Obdobně potom funguje i zadání startovní pozice robota (přeskočí funkci `fix()`) v nepřítomnosti GPS přístroje.

Dalším modulem, bez kterého se při testování můžeme obejít, je časovač startu – funkce `timer()`. Bez něj dojde ihned po zadání cílových souřadnic ke startu robota (bez vyžádání posledního potvrzení).

Modul pro načtení mapy není možné vypnout, neboť načtená mapová struktura je nutná ke správné činnosti všech následujících modulů. Podobně funkce pro sestavení trasy – `route()` – je nutná pro následný průjezd ze startu do cíle. Vypnout ji proto lze pouze v případě, že je vypnuta i funkce `drive()`.

Poslední možností je vypnutí samotné pilotovací funkce `drive()` v případě, že robot není během testování připojen. V tomto případě je možné volat alespoň funkci `cesta()`, která zajistí výpis cestovního itineráře sestavené trasy. Vše opět dle pokynů v komentářích zdrojového kódu. Během testování tohoto modulu je také možné simulovat skutečný pohyb pokud umístíme robota středem podvozku například na kartónovou krabici tak, aby točil kolečky ve vzduchu. Změny na kompasu je možné simulovat natočením celé krabice i s robotem a překážky pomocí další krabice, či přiblížením robota ke zdi místnosti.

10 Návod k obsluze

Ovládání aplikace není nikterak složité. Sestává prakticky jen z posloupnosti několika údajů, které je třeba před samotným startem zadat. V této kapitole tyto údaje shrnu v odpovídajícím pořadí a u každé uvedu smysluplné rozsahy zadávaných hodnot.

Prvním úkolem po automatickém načtení mapy je zadání souřadnic cíle závodu. Uvažovány jsou pouze souřadnice severní šířky a východní délky v těchto rozsazích:

- `N  00.000000 – N  90.000000` severní šířky ve formátu stupňů
- `E 000.000000 – E 099.999999` východní délky ve formátu stupňů
- `N  00°00.000 – N  90°00.000` severní šířky ve formátu stupňů a minut
- `E 000°00.000 – E 099°59.999` východní délky ve formátu stupňů a minut

V případě třetí možnosti - zadávání identifikátoru konkrétního trasového bodu se jedná o rozsah `00000001 – 99999999` a zvolené ID musí skutečně existovat ve zpracovávaném mapovém souboru.

Druhým zadávaným údajem je informace o času startu. V případě přesného času lze zadat libovolně mezi `00:00` a `23:59` . V případě časového offsetu je to potom maximálně trojciferná hodnota v minutách, tedy `1 – 999` . V obou případech mají ale význam pouze hodnoty odpovídající méně než půl hodině od aktuálního systémového času. Ostatní jsou odmítnuty a program končí.

Posledním úkolem je potvrzení startu klávesou `ENTER` nebo zrušení startu klávesou ‘`X`‘. Všechny další činnosti provádí během závodu robot automaticky a některá hlášení zobrazuje na standardním výstupu.

Výpis celého jednoho běhu programu (simulovaného závodu) je k nahlédnutí v příloze 3.

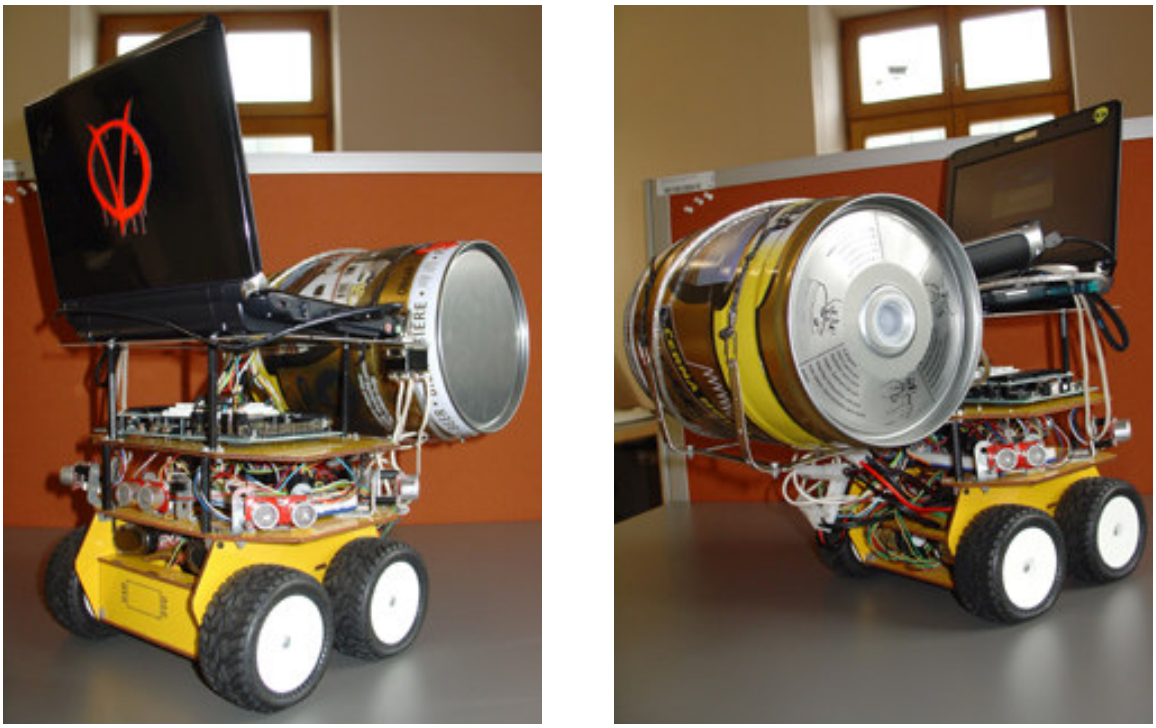
11 Hardwarové úpravy

Dle potřeb soutěže a jejích pravidel pro rok 2010 jsem provedl několik mechanických úprav na robotovi, které v následující kapitole podrobněji popíši.

Nejzásadnější změnou bylo doplnění třetího patra (platformy) na tělo robota. Tato platforma slouží k umístění řídícího notebooku. Zhotovit jsem ji nechal z 1,5mm nerezového plechu. Její velikost, tvar a další výřezy jsou uzpůsobeny pro použitý notebook Asus Eee PC 901. Detailní výkres, podle kterého byla deska vyrobena je uveden v příloze 4 této práce. Zdrojové soubory výkresu ve formátu `DWG` (AutoCAD) jsou umístěny na přiloženém CD v podsložce `zpráva/obrázky` .

Tento výkres popisuje také další 4 ocelové prvky, které jsem nechal pro potřeby soutěže vyrobit. Tyto děrované pásky sešroubované do mřížové konstrukce slouží jako nosič pro povinný robotův náklad.

Povinným nákladem je zde samozřejmě myšlen onen prázdný pětilitrový soudek na pivo. Jeho umístění a hmotnost cca 550g lehce narušují těžiště robota. Po umístění notebooku a GPS (hmotnost i s doplněnou platformou necelé 2kg) na jejich místa přímo nad středem náprav se tento handicap ale prakticky anuluje a robot je stabilní.



obrázek 11.1: robot pro Robotour 2010

Tyto obrázky zachycují nový vzhled robota po připojení a umístění všech součástí. Další fotografie jsou umístěny na přiloženém CD ve složce `foto` .

Posledním doplňkem je vypínač napájení z baterií. Dle pravidel by se mělo jednat o kulaté červené tlačítko, ale takové jsem bohužel nesehnal v žádném ze specializovaných brněnských obchodů, takže jsem nakonec koupil obyčejný dvojokruhový přepínač s červenými popisky, na který by se v při soutěži připevnila ještě další zvýrazněná červená plocha. Hlavní výhodou teď hlavně je, že už není třeba pro vypnutí robota odpojovat obě baterie manuálně přes konektory. Vypínač je připevněn v kruhovém otvoru po levé straně doplněné platformy.

Pro správné zapojení periferních zařízení jsou také nutné 2 propojovací kabely. První z nich připojuje k řídícímu notebooku samotného robota. K tomu je využit adaptér s označením HL-340 společnosti HL Technologies a křížená propojka F/F. Na straně Notebooku má klasicky USB-A konektor, na straně robota je to potom konektor DB9 sériové linky. Ovladače pro tento adaptér se nacházejí na přiloženém CD ve složce `aplikace`.

Druhý kabel slouží k připojení externího GPS zařízení Garmin Colorado 300. Zde už se jedná o běžný kablík s koncovkou USB-A na straně notebooku a USB-mini na straně GPS přístroje. O správnou emulaci COM portu s připojenou GPS v operačním systému Windows se stará aplikace Garmin Spanner. Její instalační soubor je uložen taktéž ve složce `aplikace` na přiloženém CD.

obrázek 11.2: pohled na zapojená zařízení

12 Závěr

Úkolem této práce bylo seznámit se se soutěží Robotour a s robotem vyvíjeným na ÚITS FIT v Brně. Tyto znalosti bylo následně nutné využít při návrhu a implementaci vhodných řešení a algoritmů umožňujících robotovi účast v této soutěži v roce 2010.

V úvodní části práce jsem prostudoval mnoho materiálů. Při zjišťování potřebných informací jsem nabyl mnoho nových vědomostí a poznatků. V další části, během návrhu svého řešení, jsem získal cenné vědomosti z oblasti plánovacích, lokalizačních a navigačních algoritmů, které jsem se snažil při svém řešení co nejlépe zužitkovat.

Nejzajímavější pro mě byla určitě druhá část práce. Zde jsem zjistil, že i zdánlivě snadné a předem promyšlené řešení může během implementace přinést spoustu nových problémů. Z těch plynula i potřeba studia nových faktů a materiálů a vznikala nová dílčí řešení. Dále jsem měl možnost poznat, že i při dobře rozloženém plánu práce může dojít k velkým problémům, pokud dojde k neočekávanému selhání technického zařízení, se kterým pracujeme. Do budoucí praxe je to určitě dobrá zkušenost, že s tímto rizikem bude vždy dobré počítat.

Během své práce jsem si také okrajově vyzkoušel funkci konstruktéra při doplňování a úpravách na hardwaru robota. Tato část mi dala cenné zkušenosti z oblasti strojního inženýrství, ale i z managementu při zajišťování dodavatelů či samotné výroby mechanických součástí. V praxi jsem si tedy vyzkoušel onen zásadní rozdíl mezi vývojem softwaru a výrobou fyzických předmětů a každá chybka či nedorozumění v zadání bylo vykoupeno spotřebou dalšího materiálu a energie při jejich nápravě.

Již po prostudování pravidel soutěže a detailů o robotovi bylo jasné, že vzhledem ke svým schopnostem a možnostem není reálné, aby se zařadil mezi ty nejlepší v soutěži. Na druhou stranu si ale myslím, že mé řešení je dostatečné a splňuje všechny předpoklady pro kvalifikaci do soutěže a reálný start v letošním ročníku v Bratislavě. Všechny části aplikace pro lokalizaci, zpracování mapy a plánování trasy jsou plně funkční a pracují bez problémů. Komunikace mezi všemi zařízeními pracuje taktéž správně a tak hodnotím i volbu implementačního jazyka za vyhovující a dostatečnou.

Jediné pochybení je možné jen při samotném řízení robota, kde bude záležet na mnoha vnějších faktorech jako je podklad použitých cest a přesnost jejich zmapování, nečekané překážky a nástrahy na trati aj. Výrazné zlepšení jízdních vlastností robota a celého ovládání pohybu přinesou teprve až rozsáhlejší testy a pokusy, hlavně potom nasazení v reálném terénu. Vzhledem k technickým potížím jsem byl ale k datu dokončení této práce schopen pouze základních pokusů a potřebné testy budou teprve následovat.

Vzhledem ke svým konstrukčním vlastnostem už robot moc dalších možností na rozšíření nenabízí, ale mezi ty nejreálnější zajisté patří zapojení videokamery (stereokamery), která je na ústavu k dispozici. Tato úprava by s sebou přinesla cenné zlepšení právě v části ovládání robota a rapidně by zvýšila jeho šanci na hladký a bezpečný průjezd tratí. Technologicky by se tak robot na startovním poli posunul zase o úroveň výše.

Na hodnocení úspěšnosti v samotné soutěži je ale zatím ještě moc brzo a ani plnohodnotné srovnání s minulým ročníkem není prakticky možné. Prvním důvodem je samozřejmě fakt, že letošní ročník se bude konat až za 4 měsíce a druhým je i skutečnost, že předchozí řešení nebylo nikdy dotaženo do funkčního stavu a robot se tak soutěže v loňském roce vůbec nezúčastnil.

Srovnání dále znemožňuje i značná odlišnost loňských pravidel a letošních pravidel soutěže. Letošní ročník s sebou přinesl zásadní změny v celém systému a tedy i použité návrhy, řešení a algoritmy jsou naprosto odlišné.

Vzhledem ke zmíněným technickým problémům bych velmi rád v práci i nadále pokračoval a funkční řešení dokončil. Ideální by byl samozřejmě skutečný start na soutěži v září v Bratislavě.

Přestože se jedná jen o soutěž, věřím, že úkoly řešené v rámci této práce a celá problematika autonomní navigace robotů, automobilů a potažmo i jakýchkoliv jiných strojů je velmi perspektivním směrem oblasti techniky a informatiky.

Celkově mě celá práce velmi bavila a hodnotím ji jako velmi přínosnou. Při jejím vypracování jsem si značně rozšířil obzory ve všech souvisejících tématech a utvrdil se, že obor lokalizace a navigace je přesně tím směrem, kterým bych se chtěl dále zabývat ve svém budoucím zaměstnání či dalším vědeckém výzkumu.

Literatura

[1] DLOUHÝ, Martin, WINKLER, Zbyněk. *robotika.cz* [online]. 2006 [cit. 2009-12-02]. Dostupný z WWW: <http://robotika.cz>

[2] DLOUHÝ, Martin, WINKLER, Zbyněk. *Robotour 2010 - pravidla* [online]. 2009-11-23 [cit. 2009-12-02]. Dostupný z WWW: <http://robotika.cz/competitions/robotour2010/cs>

[3] NOVOTNÝ, Tomáš. *Řízení robota pomocí FITkitu*. Brno, 2008. 28 s. Vysoké učení technické v Brně. Fakulta informačních technologií. Vedoucí bakalářské práce Ing. Jaroslav Rozman.

[4] FUČÍK, Otto, VAŠÍČEK, Zdeněk, KOŘENEK, Jan. *FITkit* [online]. Brno : c2006-2009 [cit. 2009-12-03]. Dostupný z WWW: <http://merlin.fit.vutbr.cz/FITkit/>.

[5] Lynxmotion, Inc. *Lynxmotion Robot Kits* [online]. c2009 [cit. 2009-12-03]. Dostupný z WWW: <http://www.lynxmotion.com>.

[6] *OpenStreetMap* [online]. [2004] [cit. 2009-12-06]. Dostupný z WWW: <http://www.openstreetmap.org/>.

[7] ZBOŘIL, František. UCS – Uniform Cost Search. *IZU – Základy umělé inteligence* [online]. 2008 [cit. 2009-12-21], s. 6-14. Dostupný z WWW: <https://www.fit.vutbr.cz/study/courses/IZU/private/0809izu_2.pdf>.

[8] *World Geodetic System* [online]. [2003] , last modified on 6 December 2009 [cit. 2009-12-06]. Dostupný z WWW: <http://en.wikipedia.org/wiki/World_Geodetic_System>.

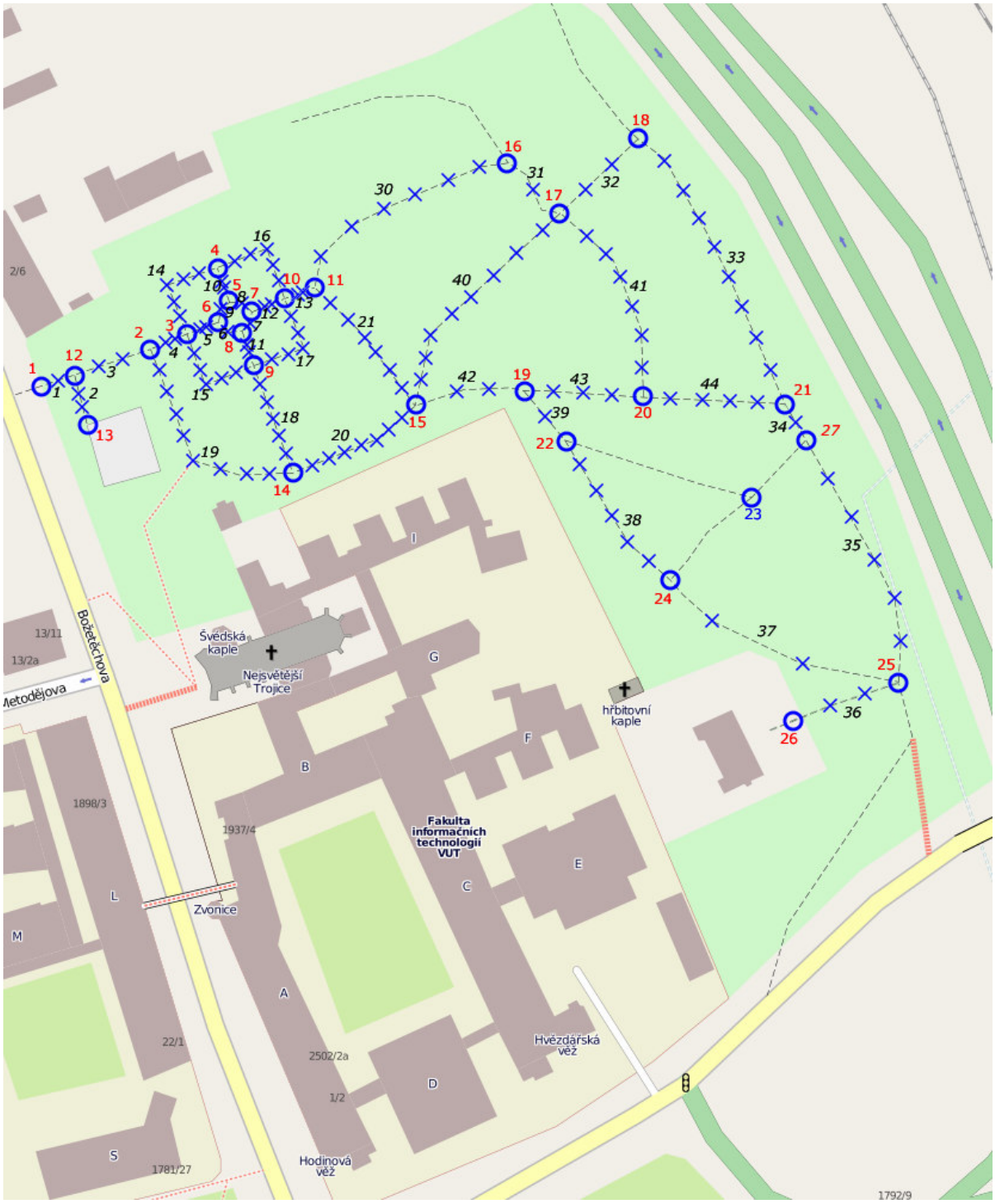
[9] CODEPLUG, X. *Cprogramming.com* [online]. 2003 [cit. 2010-04-09]. Simple RS232 Question. Dostupné z WWW: <http://cboard.cprogramming.com/attachments/cplusplus-programming/4005d1070392893-simple-rs232-question-sfile-zip>.

[10] THRUN, Sebastian; BURGARD, Wolfram; FOX, Dieter. *Probabilistic Robotics : Intelligent Robotics and Autonomous Agents*. The MIT Press, 2005. 667 s. ISBN 978-0-262-20162-9.

[11] DOUBEK, Milan. *Robot pro Robotour 2009*. Brno, 2009. 28 s. Vysoké učení technické v Brně. Fakulta informačních technologií. Vedoucí bakalářské práce Ing. Jaroslav Rozman.

Seznam příloh

Příloha 1. Mapa parku Božetěchova



- modrá kolečka s červenými čísly reprezentují trasové body - křižovatky - a jejich ID
- modré křížky reprezentují ostatní trasové body
- černá čísla jsou ID jednotlivých cest

Příloha 2. Část XML zápisu mapy

```
<?xml version="1.0" encoding="UTF-8"?>
<osm version="0.6" generator="CGImap 0.0.2">
 <bounds minlat="49.226489" minlon="16.595405" maxlat="49.228826"
maxlon="16.598918"/>

...

 <node id="00000001" lat="49.227862" lon="16.595517" user="xkubat"
uid="06" visible="true" version="1" changeset="1" timestamp="2010-
03-10T09:08:03Z">
  <tag k="created_by" v="JOSM"/>
 </node>
 <node id="00000002" lat="49.227956" lon="16.595909" user="xkubat"
uid="06" visible="true" version="1" changeset="1" timestamp="2010-
03-10T09:08:13Z">
  <tag k="created_by" v="JOSM"/>
 </node>

 <node id="00000032" lat="49.227877" lon="16.595585" user="xkubat"
uid="06" visible="true" version="1" changeset="1" timestamp="2010-
03-10T11:52:11Z">
  <tag k="created_by" v="JOSM"/>
 </node>
 <way id="00000001" user="xkubat" uid="06" visible="true"
version="1" changeset="1" timestamp="2010-03-10T11:52:41Z">
  <nd ref="00000001"/>
  <nd ref="00000032"/>
  <nd ref="00000012"/>
  <tag k="created_by" v="JOSM"/>
  <tag k="highway" v="footway"/>
  <tag k="source" v="gmaps:ortofoto"/>
 </way>

...

</osm>
```

- úsek kódu popisuje tři trasové body a jednu cestu, která je spojuje
- tučně jsou v kódu vyznačena místa, které jsou při načítání klíčová
- na začátku a konci jsou tagy identifikující formát XML a JOSM

Příloha 3. Ukázka běhu aplikace

```
Mapa uspesne nactena!
Prejete si mapu vypsat? (Ano/Ne): N

Nejprve prosim zadejte souradnice cile zavodu.
Vyberte format souradnic:
  0 - N DD.DDDDDD E DDD.DDDDDD
  1 - N DD°MM.MMM E DDD°MM.MMM
  5 - pro prime zadani ID bodu cile
  X - pro ukonceni programu
Vase volba: 5

Zadejte ID cislo ciloveho bodu (JOSM node ID).
ID cile: 2

Zadejte prosim cas startu zavodu.
Stisknete:
  0 - pro zadani presneho casu startu
  1 - pro zadani casoveho offsetu do startu
  X - pro ukonceni programu
Vase volba: 1

Nyni prosim zadejte, za jak dlouho zavod zacne (v minutach).
Doba do startu: 10

Zavod zacina za 10 minut.
Souradnice cile: N 49.227955 E 016.595909
Pro zruseni startu stisknete X, pro zahajeni odpoctu stisknete ENTER!

Byl zahajen odpocet . . . . . .
Zbyva poslednich 20 vterin do startu: * * * * * * * * * * * * * * * * * * * * START!!!

Urcuji svoji startovni pozici: * * * * * * * * * hotovo!

Planuji trasu: * * * * *
Plan trasy sestaven!
Prejete si vypsat grafy algoritmu UCS? (Ano/Ne): N

Prujezd usekem:   1 -  32 (  526[cm],  73[deg] )

!!! POZOR !!!
Prekazka na trati . . . . . Volno.


Prujezd usekem:  32 -  12 (  398[cm],  62[deg] )

!!! POZOR !!!
Prekazka na trati . . . . . .
Odklidte prosim prekazku ......... Volno.


Prujezd usekem:  12 -  35 (  526[cm],  73[deg] )

Prujezd usekem:  35 -  36 (  478[cm],  62[deg] )

Prujezd usekem:  36 -   2 ( 1107[cm],  76[deg] )

!!! POZOR !!!
Prekazka na trati . . . . . .
Odklidte prosim prekazku ..........................................
Zahajuji uhybny manevr!


!!! POZOR !!!
Prekazka na trati . Volno.


!!! CILOVA POZICE BYLA DOSAZENA !!!

Pokračujte stisknutím libovolné klávesy...
```

Příloha 4. Výkres doplňkových součástek