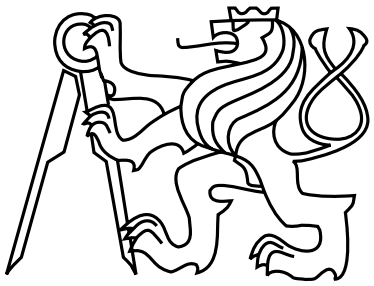
ČESKÉ VYSOKÉ UČENÍ TECHNICKÉ V PRAZE
FAKULTA ELEKTROTECHNICKÁ

KATEDRA ŘÍDICÍ TECHNIKY



Implementace stavových automatů pro soutěž Eurobot 2009

BAKALÁŘSKÁ PRÁCE

Filip Jareš

Vedoucí práce: Ing. Michal Sojka

Praha 2009

Poděkování

Děkuji Michalu Sojkovi, vedoucímu práce, za možnost účastnit se na zajímavém projektu a za jeho nevšední trpělivost. Děkuji svým rodičům a svým nejbližším za jejich lásku a podporu. Také bych chtěl vyjádřit dík autorům svobodného a otevřeného softwaru, který mohu využívat.

Prohlášení

Prohlašuji, že jsem práci vypracoval samostatně a použil jsem pouze podklady (literaturu, projekty, SW atd.) uvedené v přiloženém seznamu.

V Praze dne 7. 1. 2010 .....

Abstract

This thesis presents finite state automata used to control the CTU Dragons' team robot for the Eurobot 2009 robotic contest. Namely the finite state machine controlling the game strategy of the robot and the finite state machine for in-game element manipulators control. It describes cooperation of these automata with the rest of the software. The FSM library which was created by other team members for previous years of the contest was used for the automata implementation and is also presented in this thesis.

Abstrakt

Tato práce popisuje stavové automaty napsané pro účast robotu týmu CTU Dragons v robotické soutěži Eurobot 2009. Konkrétně stavový automat implementující herní strategii a stavový automat ovládající mechanismy použité pro manipulaci s herními prvky. Popisuje rovněž spolupráci těchto automatů s ostatním sofwarem robotu. Pro realizaci zmíněných automatů byla použita knihovna FSM vyvinutá členy týmu pro předchozí ročníky soutěže, která je v práci rovněž představena.

České vysoké učení technické v Praze
Fakulta elektrotechnická

Katedra řídicí techniky

ZADÁNÍ BAKALÁŘSKÉ PRÁCE

Student: **Filip Jareš**

Studijní program: Elektrotechnika a informatika (bakalářský), strukturovaný
Obor: Kybernetika a měření

Název tématu: **Implementace stavových automatů pro soutěž Eurobot 2009**

Pokyny pro vypracování:

1. Seznamte se s možnostmi implementace stavových automatů pro robotické aplikace a s knihovnou FSM vyvinutou pro předchozí ročníky soutěže.
2. Seznamte se s architekturou softwaru a hardwaru robota DragonBot 2009 a pravidly soutěže Eurobot 2009.
3. V souladu s pravidly implementujte herní strategii pomocí stavových automatů.
4. Ve spolupráci s ostatními členy týmu řešení pečlivě otestujte.
5. Vše důkladně zdokumentujte.

Seznam odborné literatury:

Dodá vedoucí práce

Vedoucí: Ing. Michal Sojka

Platnost zadání: do konce zimního semestru 2009/10

prof. Ing. Michael Šebek, DrSc.
vedoucí katedry

doc. Ing. Boris Šimák, CSc.
děkan

V Praze dne 27. 2. 2009

Obsah

Kapitola 1

Úvod

Cílem této práce je popsat část řídicího softwaru robotu týmu CTU Dragons použitého pro ročník 2009 soutěže Eurobot a zejména zdokumentovat stavové automaty použité pro implementaci herní strategie a pro ovládání manipulátorů použitých na robotu.

Eurobot je mezinárodní soutěž[1] pro amatérské týmy „mladých robotiků“. Na hřišti o rozměrech přibližně 3×2 metry spolu soutěží autonomní mobilní roboty dvou soupeřících týmů a plní úkol daný pravidly pro aktuální ročník. Pravidla jsou vydávána na podzim a na jaře se týmy utkávají v národních kolech. Ročník 2009 byl nazván „Chrámy Atlantidy“ a pod tímto názvem se skrývalo stavění věží z puků a trámků.

Ročník 2009 nebyl prvním, jehož se zmíněný robot (označovaný ve starších materiálech týmu jménem *DragonBot*) účastnil. Kromě toho, že na robotovi je stále co vylepšovat, přináší s sebou každý ročník soutěže nutnost změn a přestaveb.

Struktura této práce je následující. V kapitole 2 jsou představena pravidla soutěže. V kapitole 3 jsou popsány některé knihovny, jejichž služeb stavové automaty využívají, a zejména knihovna *FSM*, která byla použita pro implementaci těchto stavových automatů. Kapitola 4 popisuje strukturu hardwaru a softwaru robotu, tedy prostředí, ve kterém stavové automaty fungují. Těžištěm práce je kapitola 5, ve které je v úvodu popsána spolupráce stavových automatů robotu a především jsou v ní popsány stavové automaty, které jsem implementoval. Práci uzavírám zhodnocením v kapitole 6.

Mou vlastní prací jsou zejména stavové automaty popsané v kapitole 5 a z části jejich integrace do ostatního softwaru. Zdrojové kódy implementovaných stavových automatů jsou na přiloženém CD. Podrobnější informace o jeho obsahu jsou v dodatku B.

[1] Oficiální stránky soutěže jsou k nalezení na adrese http://www.eurobot.org/; pořadatelem českého národního kola jsou Katedra softwarového inženýrství MFF UK a občanské sdružení Robonika.

Kapitola 2

Pravidla soutěže Eurobot 2009

Jedním z hlavních cílů této práce je implementace herní strategie robotu v souladu s pravidly soutěže Eurobot 2009. V této kapitole jsou pravidla soutěže stručně popsána, zejména body, které se přímo týkají této práce. Se všemi podrobnostmi pravidel se čtenář může seznámit v oficiálních pravidlech [1].

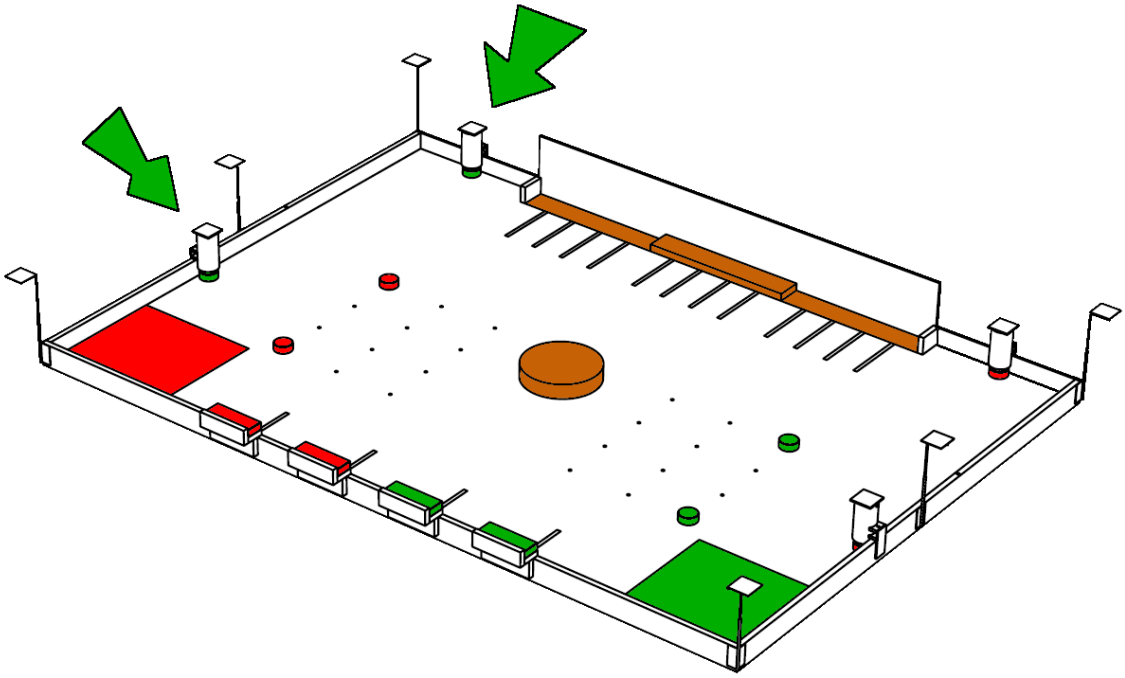
V soutěžním zápase se spolu na modrém hřišti utkávají roboty dvou soupeřících týmů. Jeden robot soutěží za červenou stranu, druhý za zelenou. Barva robotu je vybrána před začátkem zápasu a každý robot začíná zápas ve své (červené nebo zelené) startovní oblasti. Hřiště má délku 3000 mm a šířku 2100 mm a je symetrické podle své kratší osy (viz obr. 2.1). Červená a zelená startovní oblast jsou umístěny v rozích hřiště na jedné z jeho delších stran.

Na určených místech na ploše hřiště a v zásobnících na okrajích herní plochy jsou před startem umístěny herní prvky, kterými jsou dřevěné válečky o průměru 70 mm a výšce 30 mm (puky) a dřevěné hranoly (trámky) 200 mm dlouhé, 70 mm široké a 30 mm vysoké. Z těchto prvků má robot za úkol postavit na některé z konstrukčních oblastí co nejvyšší věže a „chrámy“ (stavby tvořené předepsaným způsobem z puků a trámků). Musí při tom respektovat barevné označení těchto herních prvků (záměrná manipulace s protivníkovými herními prvky není dovolena).

Centrální konstrukční oblast je vyvýšená kruhová plocha o průměru 300 mm, umístěná ve středu hřiště, pro stručnost o ní dále budeme mluvit jako o *akropoli*. Má výšku dvou na sebe postavených puků. Další tři (obdélníkové) konstrukční zóny jsou umístěny podél dlouhé strany hřiště naproti startovním oblastem. Dvě krajní jsou výškově na úrovni zbytku hrací plochy a mezi nimi je vyvýšená oblast o výšce jednoho puku.

Každý platně umístěný herní prvek je bodován. Sloupový díl (puk) je platný, pokud leží naplocho a plně v konstrukční zóně, nebo pokud leží naplocho a je podepřen alespoň jedním platným herním prvkem. Trámek je platný, pokud leží naplocho a je podepřen alespoň dvěma platnými prvky. Platně umístěný sloupový díl je ohodnocen počtem bodů rovným úrovni, ve které je nad hrací plochou. Platně umístěný trámek je započítán trojnásobkem bodů oproti sloupovému dílu platně umístěnému ve stejné výšce.

Robot smí ke stavbě použít herní prvky své barvy, rozmístěné před startem na hřišti. Dva trámky příslušné barvy jsou umístěny v podavačích na kraji hřiště, v blízkosti startovací

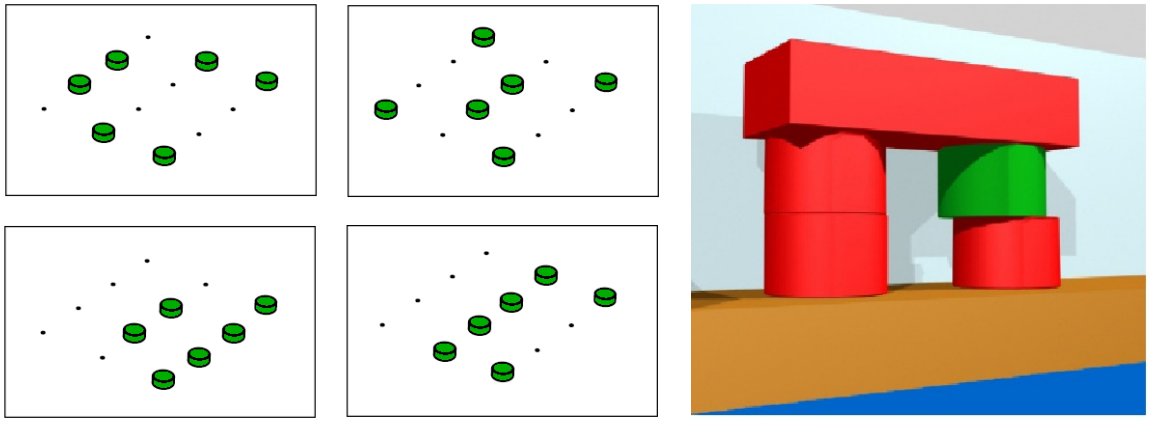


Obrázek 2.1: Hřiště s vybarvenými startovními oblastmi (barevné čtverce v rozích), červenými a zelenými herními prvky, zvýrazněnými zásobníky puků zeleného hráče a vybarvenými konstrukčními zónami (obrázek byl převzat z pravidel soutěže [1] a mírně upraven)

oblasti (viz obr. 2.1). Kromě nich pravidla povolují týmu umístit do robotu před startem ještě třetí trámek.

Pět a pět sloupových dílů odpovídající barvy je v zásobnících na druhé polovině hřiště, než je startovací oblast této barvy (pozice zelených zásobníků zvýrazňují šipky na obr. 2.1). Jeden ze zásobníků má pevně určenou pozici (ten na delší straně hřiště) a přesná pozice zásobníku umístěného na kratší straně hřiště se losuje před startem.

Kromě puků v zásobnících je na hřišti umístěno 12 puků (šest od každé barvy) v jedné z deseti předepsaných konfigurací v mřížce 3×4 bodů, která je rovněž patrná na obr. 2.1. Tyto puky, rozmístěné před začátkem zápasu na ploše hřiště, budeme pro stručnost označovat jako *volné puky*. Puky nakreslené v mřížkách na obr. 2.1 jsou na hřišti vždy (patří do každé z deseti konfigurací). Všech deset definovaných uspořádání puků je symetrických podle osy rovnoběžné s delší stranou hřiště. Příklady čtyř konfigurací jsou na obr. 2.2. Jejich kompletní přehled je v příloze A. Konfigurace použitá v zápase je losována před startem a je na robotech, aby ji rozpoznaly nebo si s ní poradily jinak.



Obrázek 2.2: (a) Příklady počátečních rozmístění prvků na hrací ploše; (b) příklad platné stavby (převzato z FAQ dokumentu k soutěži)

Jeden zápas trvá 90 sekund a roboty jsou na pokyn rozhodčího odstartovány členem týmu. To musí být provedeno vytažením startovacího lanka. Dle pravidel musí být robot dále vybaven systémem, který ho na konci 90sekundového zápasu automaticky zastaví (vypne všechny pohony včetně vnitřních zařízení robotu). Zároveň pravidla omezují počet kontrolovaných puků: robot smí mít pod kontrolou v každý okamžik nejvýše 4 sloupové díly. Roboty musí být schopny vyhýbat se překážkám (a zabránit tak kolizím s protivníkovým robotem).

Kapitola 3

Použité knihovny

V této kapitole jsou v podkapitolách 3.1 až 3.2 popsány některé z nejdůležitějších knihoven použitých při implementaci stavových automatů, jimž se věnuje tato práce. V podkapitole 3.4 je potom stručně zmíněn komunikační middleware ORTE.

3.1 Knihovna FSM

Knihovna *FSM* slouží pro implementaci stavových automatů v jazyce C a byla vyvinuta pro předchozí ročníky soutěže. V následujících odstavcích je volně citována dokumentace knihovny.[1]

Konkrétní stavový automat je v rámci knihovny *FSM* představován strukturou `struct robo_fsm` a je tvořen množinou stavových funkcí. Stavová funkce je knihovnou volána jako reakce na příchod určité události. V každém okamžiku existuje pro daný stavový automat právě jediná „aktuální“ stavová funkce. Změna stavu automatu znamená změnu aktuální stavové funkce (změnu ukazatele na aktuální stavovou funkci).

Smyslem stavových funkcí je reagovat na události. Stavové funkce pro stavové automaty vytvořené s užitím knihovny *FSM* se definují jako běžné funkce jazyka C s použitím makra `FSM_STATE(název_stavu)` namísto hlavičky. Jsou zpravidla zapsány jako jediný velký `switch` konstrukt jazyka C, který jako parametr přijímá makro `FSM_EVENT`. Toto makro nabývá hodnoty odpovídající události, která má být zpracována. Typická stavová funkce tedy na základě „došlé“ události vykoná nějaký užitečný kód, resp. vyvolá přechod stavového automatu do jiného stavu. Příklady stavových funkcí jsou na str. 30 a na str. 39.

Události registrované stavovým automatem patří buď do množiny *základních událostí* generovaných knihovnou (viz následující odstavec), nebo jsou to události vytvářené jinými stavovými automaty, případně dalšími částmi kódu.

Knihovna *FSM* definuje ve výčtovém typu `enum fsm_common_events` množinu *základních událostí*. Jedná se o události, které generuje přímo knihovna. Základní události jsou čtyři, a sice: `EV_ENTRY`, `EV_EXIT`, `EV_RETURN` a `EV_TIMER`. První z nich znamená, že byl právě vyvolán přechod do aktuálního stavu. Druhá zmíněná událost má naopak význam „právě

[1] oddíl Finite State Machines v [2]

dochází k přechodu do jiného než aktuálního stavu“. Událost `EV_RETURN` označuje návrat ze subautomatu (viz dále), který byl dříve z aktuálního stavu zavolán. Událost `EV_TIMER` je knihovnou vytvářena, když vyprší časovač, který byl dříve ve stejném stavu nastaven pomocí makra `FSM_TIMER()`.

Kromě základních událostí má uživatel knihovny možnost nadefinovat pro konkrétní stavový automat množinu událostí specifických pro tento automat. Takové události se definují ve zvláštním souboru v asociativním poli programovacího jazyka Python nazvaném `events`. Knihovna *FSM* poskytuje pro tyto účely pythonovský skript `eventgen.py`, který z uvedeného definičního souboru vytvoří `.c` a `.h` soubory používané knihovnou. Klíči ve zmíněném asociativním poli jsou identifikátory jednotlivých stavových automatů, definovaných v rámci programu, zapsané malými písmeny. Hodnotami, které těmto klíčům příslušejí, jsou jiná asociativní pole, definující sadu událostí pro daný automat, která mají jako klíče identifikátory událostí a jako hodnoty komentáře k těmto událostem. Identifikátory událostí se většinou zapisují velkými písmeny a začínají řetězcem `"EV_"`. Pro ilustraci toho, jak takové asociativní pole `events` vypadá, je zde uvedena část jeho definice pro stavové automaty popisované v této práci:

```
events = {
  "main" : {
    "EV_START" : "Changed state of start connector.",

    "EV_ACTION_DONE"    : "ACT FSM signals that ...",
    "EV_ACTION_ERROR"   : "ACT FSM signals that ...",
    "EV_PUCK_REACHABLE" : "Received through ORTE. ...",

    "EV_MOTION_DONE" : "Previously submitted motion finished",
  },
  "act" : {
    "EV_LOAD_THE_PUCK"    :
        "Signal from the main FSM to initiate the puck \
         picking up procedure",

  # ... definice dalsich udalosti vynechany

  }
  "disp" : {
  },

  # ... definice udalosti pro dalsi stavove automaty vynechany

}
```

Právě naznačený poměrně komplikovaný způsob definice událostí umožňuje knihovně *FSM* produkovat při překladu programu kompilační chyby v případě, že se uživatel knihovny dopustí nesprávného použití stavových automatů. Například pokud se pokusí zaslat automatu událost, která pro něj není definována, nebo pokud se ve stavové funkci snaží

ošetřit příchod takové události. Zároveň překladač produkuje varování, pokud uživatel zapomene ve `switch` konstruktu nějaké stavové funkce ošetřit některou z událostí příslušného stavového automatu.

Pro úplnost ještě zmiňme, že událost s sebou může nést data (ve formě ukazatele nebo proměnné typu `int` přetypované na ukazatel). Událost v knihovně *FSM* je totiž vnitřně definována jako struktura obsahující identifikátor události a ukazatel.

Při psaní stavového automatu má uživatel v rámci stavových funkcí k dispozici několik maker, z nichž nejdůležitější je makro `FSM_TRANSITION()` pro vyvolání přechodu stavového automatu do jiného stavu. Význam některých těchto maker je vysvětlen v tab. 3.1.

`FSM_TRANSITION(next_state)` Způsobí přechod stavového automatu do stavu `next_state`. K přechodu nedojde ihned. Aktuální stavová funkce je dokončena. V případě, že stav `next_state` není aktuálním stavem, je aktuální stavová funkce volána znovu s událostí `EV_EXIT` jako hodnotou, ve kterou se rozvine makro `FSM_EVENT`. Poté je (i v případě, že `next_state` odpovídá aktuálnímu stavu) volána stavová funkce odpovídající stavu `next_state` s hodnotou `EV_ENTRY` jako událostí.

`FSM_SIGNAL(fsm_id, event, data)` Toto makro slouží pro zaslání události `event` stavovému automatu označenému identifikátorem `fsm_id`; `data` je ukazatel nebo `int` přetypovaný na ukazatel, který se má poslat automatu spolu s událostí.

`FSM_TIMER(timeout)` Nastaví jednorázový časovač, který vyprší po uplynutí `timeout` milisekund. Po vypršení časovače (pokud nebyl časovač předtím zastaven voláním makra `FSM_TIMER_STOP`) knihovna vygeneruje událost `EV_TIMER`. Pokud dojde k volání makra `FSM_TRANSITION()`, časovač je zastaven automaticky.

`FSM_TIMER_STOP()` Zastavuje časovač spuštěný voláním makra `FSM_TIMER()`.

`SUBFSM_TRANSITION(substate, data)` Makro, které způsobí přechod do subautomatu (viz níže), jehož je stav `substate` počátečním stavem. Je podobné makru `FSM_TRANSITION()`, ovšem s tím rozdílem, že umožňuje spolu s událostí předat data.

`SUBFSM_RET(data)` Způsobí návrat z subautomatu do volajícího automatu a umožňuje předat data. Ve stavu volajícího automatu, v němž došlo k volání subautomatu, je po návratu vygenerována událost `EV_RETURN`.

Tabulka 3.1: Přehled některých maker, které má uživatel knihovny *FSM* k dispozici při psaní stavových funkcí

Posledním konceptem knihovny *FSM*, který zbývá zmínit, jsou subautomaty. Část stavů, definovaných ve stavovém automatu napsaném v knihovně *FSM*, může tvořit subautomat. Při přechodu z nějakého stavu do subautomatu dojde k opuštění původního stavu a vstupu do stavu v podautomatu; v původním stavu ale není vyvolána událost `EV_EXIT`. Automat se chová, jako by zůstával ve volajícím stavu a vstoupil při tom do stavu nového, do stavu ve volaném subautomatu.

Poté, co subautomat dokončí svoji činnost, vyvolá návrat do původního stavu. To znamená, že dojde normálním způsobem k opuštění příslušného stavu v podautomatu a v původním stavu volajícího automatu dojde k vygenerování události `EV_RETURN`.

3.2 Knihovna movehelper

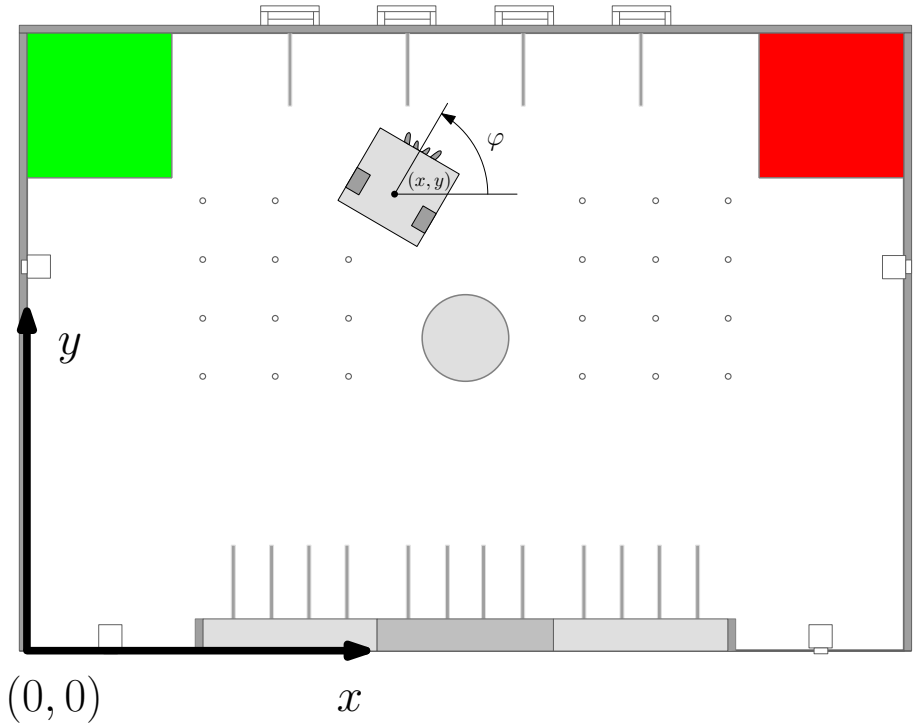
Součástí řídicího programu robotu je stavový automat *move fsm* koordinující řízení pohybu robotu. Jednotlivé stavové automaty, které tvoří řídicí program, spolu komunikují převážně zasíláním událostí. (Podrobněji je spolupráce jednotlivých stavových automatů použitých k řízení robotu popsána v úvodu kapitoly 5.) Také stavový automat *move fsm* přijímá instrukce pro svoji činnost prostřednictvím událostí. Datová struktura předávaná tomuto automatu spolu se signálem pro zahájení pohybu je ovšem natolik složitá, že vytvářet ji celou v rámci klienta stavového automatu *move fsm* (kterým je hlavní stavový automat popsaný v části 5.1) by klienta znatelně komplikovalo. Knihovna *movehelper* proto pro stavový automat *move fsm* vytváří rozhraní, které může využívat hlavní stavový automat k ukládání požadavků na pohyb robotu.

Funkce knihovny, které slouží k iniciování pohybu robotu, přijímají jako parametr souřadnice na hřišti. Jak je zavedena použitá *souřadná soustava* je znázorněno na obr. 3.1. Její počátek je umístěn do rohu hřiště nacházejícího se úhlopříčně oproti startovní oblasti červeného týmu. Souřadnice x roste ve směru dlouhé strany hřiště, y naopak směrem k zelené startovní oblasti. Definice *rozměrů hřiště* a *rozměrů robota* je umístěna v hlavičkovém souboru `robodim.h`.

Kromě souřadnic jsou parametrem některých funkcí knihovny *movehelper*, vytvářejících požadavek na pohyb robotu, také dvě struktury. Jednak struktura omezující způsob, jakým má robot požadovaný pohyb dokončit (`struct move_target_heading`), a dále struktura definující dynamická omezení trajektorie která bude naplánována (to je struktura `struct TrajectoryConstraints`). Její uspořádání je následující:

```
struct TrajectoryConstraints {
  double maxv;        // maximalni rychlost [m/s]
  double maxomega;    // maximalni uhlova rychlost [rad/s]
  double maxangacc;   // maximalni uhlove zrychleni [rad/s^2]
  double maxacc;      // maximalni tecne zrychleni [m/s^2]
  double maxcenacc;   // maximalni dostredive zrychleni [m/s^2]
  double maxe;        // maximal odchylka traj. v rozich [m]
};
```

Namísto přímého použití struktury `struct move_target_heading` může uživatel využít sadu maker, které se rozvíjejí v tuto strukturu a jejichž význam je popsán v tab. 3.2. Existence těchto maker zjednodušuje použití knihovny *movehelper*.



Obrázek 3.1: Zavedení souřadného systému hřiště a úhlu `heading` (φ)

`TURN_CW(heading)` Způsobuje, že se robot po dokončení pohybu otočí ve směru hodinových ručiček do směru daného úhlem `heading` v radiánech.

`TURN_CCW(heading)` Způsobuje, že se robot po dokončení pohybu otočí proti směru hodinových ručiček do směru daného úhlem `heading` v radiánech.

`TURN(heading)` Způsobuje, že se robot po dokončení pohybu otočí do směru daného úhlem `heading` v radiánech; směr otáčení je zvolen tak, aby byl pohyb nejkratší.

`NO_TURN()` Robot po dokončení pohybu zůstane natočen tak, jak přijel.

`ARRIVE_FROM(heading, dist)` Zvláštní makro, které způsobí, že robot dojede na bod o zadaných souřadnicích z takového směru, aby jeho orientace byla daná úhlem `heading` v radiánech, a přiblíží se k němu tak, že poslední část trajektorie o délce `dist` je přímková; (toto makro se používá při přejezdu mezi puky a při příjezdu ke kruhové akropoli).

Tabulka 3.2: Přehled maker pro určení způsobu zakončení pohybu vyvolávaného funkcemi z knihovny *movehelper*; zavedení úhlu `heading` viz obr. 3.1

Úloha, kterou řeší zelený a červený tým je symetrická podle osy hřiště rovnoběžné s jeho kratším okrajem. Řídicí program robotu toho využívá. Vše je navrženo pro hru za zelený tým a přepočet souřadnic v případě, kdy robot představuje červeného hráče, je vyřešen na úrovni knihovny *movehelper*. Funkce pro přesun robotu, tvořící rozhraní knihovny *movehelper*, proto existují ve dvou variantách. V první variantě končí jméno funkce na `_notrans`, případně nemá žádnou podobnou příponu, a ve druhé variantě končí jméno funkce příponou `_trans`. Obě varianty funkcí přijímají souřadnice (případně strukturu `move_target_heading`) tak jak byly zavedeny v předchozích odstavcích. Volání funkce ve variantě `_notrans` znamená „absolutní“ požadavek (nezávislý na barvě, za kterou robot hraje). Naopak u funkcí ve variantě `_trans` jsou předané souřadnice použity tak jak jsou pouze v případě hry za zeleného hráče. Pokud je `_trans` funkce volána při hře za červeného, jsou souřadnice a orientace patřičně přepočítány.

Některé funkce knihovny *movehelper* jsou vypsány v tabulce 3.3.

3.3 Knihovna actlib

Knihovna *actlib* je jednoduchá jednoúčelová knihovna pro účely letošního ročníku soutěže, její zdrojové kódy jsou tvořeny dvěma soubory: `actuators.c` a `actuators.h`. Knihovna slouží k zasílání povelů aktuátorům a jediné, co provádí je, že tyto povely publikuje prostřednictvím ORTE, což je komunikační middleware zmíněný v části 3.4, který činí systém (software robotu) více modulárním.

Rozhraní knihovny spočívá v několika funkcích, které zasílají povely mechanismům popsaným v části 5.2.1 (výtahu, pásům, klepetům, dvířkám, vytlačovači, servu) a které naklápí rangefinder Hokuyo. Součástí knihovny jsou také dvě funkce, které ve výsledku aktivují nebo pozastavují program *rozpuk*, stručně popsaný v části 4.2 a zmiňovaný v kapitole 5.1.3.

Funkce poskytované knihovnou actlib začínají předponou `act_`. V hlavičkovém souboru `actuators.h` jsou nadefinovány konstanty (resp. makra), které odpovídají důležitým polohám jednotlivých aktuátorů a které jsou použity při volání funkcí knihovny *actlib*.

3.4 Ocera Real-Time Ethernet – ORTE

ORTE je open source implementace komunikačního protokolu RTPS (Real-Time Publish Subscribe), jejíž použití na robotu zvyšuje flexibilitu práce na něm. Umožňuje modulárním způsobem od sebe oddělit jednotlivé části softwaru běžícího na hlavním počítači robotu a tím ho významně zpřehlednit a zjednodušit. Navíc umožňuje pustit různé části softwaru na různých strojích (umožňuje spolupráci programů na robotu s programy na počítači vývojáře, sdílení dat mezi nimi). Pro více informací o komunikačním middlewaru ORTE viz [3].

`robot_set_est_pos_trans(double x, double y, double phi)`
: umožňuje definovat polohu robotu

`robot_moveto_trans(`
`double x, double y,`
`struct move_target_heading heading,`
`struct TrajectoryConstraints *tc)`
: Způsobí naplánování trajektorie (s omezeními danými odkazovanou strukturou `TrajectoryConstraints`) pro přesun na souřadnice `x`, `y` (nebo v případě použití makra `ARRIVE_FROM()` na místo v příslušné vzdálenosti před bodem daným těmito souřadnicemi). V případě, že je trajektorie průjezdná, započne robot pohyb. O naplánování trajektorie a o případnou koordinaci řízení pohybu se stará stavový automat *move fsm*. Tato funkce ve skutečnosti pro uvedený automat pouze vytváří událost `EV_NEW_TARGET`, ke které připojí patřičná data.

`robot_move_by(`
`double distance,`
`struct move_target_heading heading,`
`struct TrajectoryConstraints *tc)`
: Funkce, která, podobně jako předchozí, vytváří událost pro stavový automat *move fsm*. Tato funkce ovšem nesvěří plánování trajektorie stavovému automatu, ale sama vytvoří jednoduchou trajektorii a předá mu ji. Je využívána v případě, kdy uživatel potřebuje vyvolat např. krátký pojezd robotu nebo otočení na místě bez detekce kolizí a podobně. Vytvořená trajektorie je pouze úsečkou nasměrovanou na základě orientace robotu v okamžiku volání. Délku úsečky určuje parametr `distance`. Směr (dopředu/dozadu) jeho znaménko.

`robot_stop()`
: způsobí okamžité zastavení pohybu robota

`robot_trajectory_new(struct TrajectoryConstraints *tc)`
`robot_trajectory_add_point_notrans(double x, double y)`
`robot_trajectory_add_point_notrans(double x_m, double y_m)`
: Trojice funkcí tvořící staré rozhraní knihovny. Tyto funkce vytvářejí trajektorii z přímkových segmentů pospojovanou splinovými (původně kružnicovými) úseky. První funkce vytvoří novou prázdnou trajektorii s omezeními danými strukturou na kterou odkazuje `tc`. Funkce `robot_trajectory_add_point_notrans()` přidává k trajektorii vytvořené předchozí funkcí nový bod a může být volána opakovaně. Poslední funkce přídá k trajektorii, vytvořené pomocí předchozích dvou, poslední bod a předá tuto trajektorii stavovému automatu *move fsm*, což započne pohyb.

Tabulka 3.3: Vybrané funkce knihovny *movehelper* (návrat. hodnoty `void`, souřadnice v m)

Kapitola 4

Architektura hardwaru a softwaru robotu

V této kapitole je popsán robot z hlediska hardwarové a softwarové konfigurace. Jejím cílem je popsat vztah mezi stavovými automaty popsanými v kapitole 5 a ostatním softwarem.

Hardwarová architektura robotu je znázorněna na obr. 4.1. Srdcem robotu je deska postavená na procesoru MPC5200b s architekturou PowerPC firmy Freescale. Procesor běží na taktovací frekvenci 400 MHz a má k dispozici 64 MB operační paměti. Na desce je 64 MB paměti Flash. Běží na ní linuxové jádro a program Busybox, který integruje UN*Xový shell a funkcionalitu množství standardních UN*Xových nástrojů v jediném spustitelném souboru. Všechen kód popsaný v této práci běží při soutěži na uvedeném procesoru.[1]

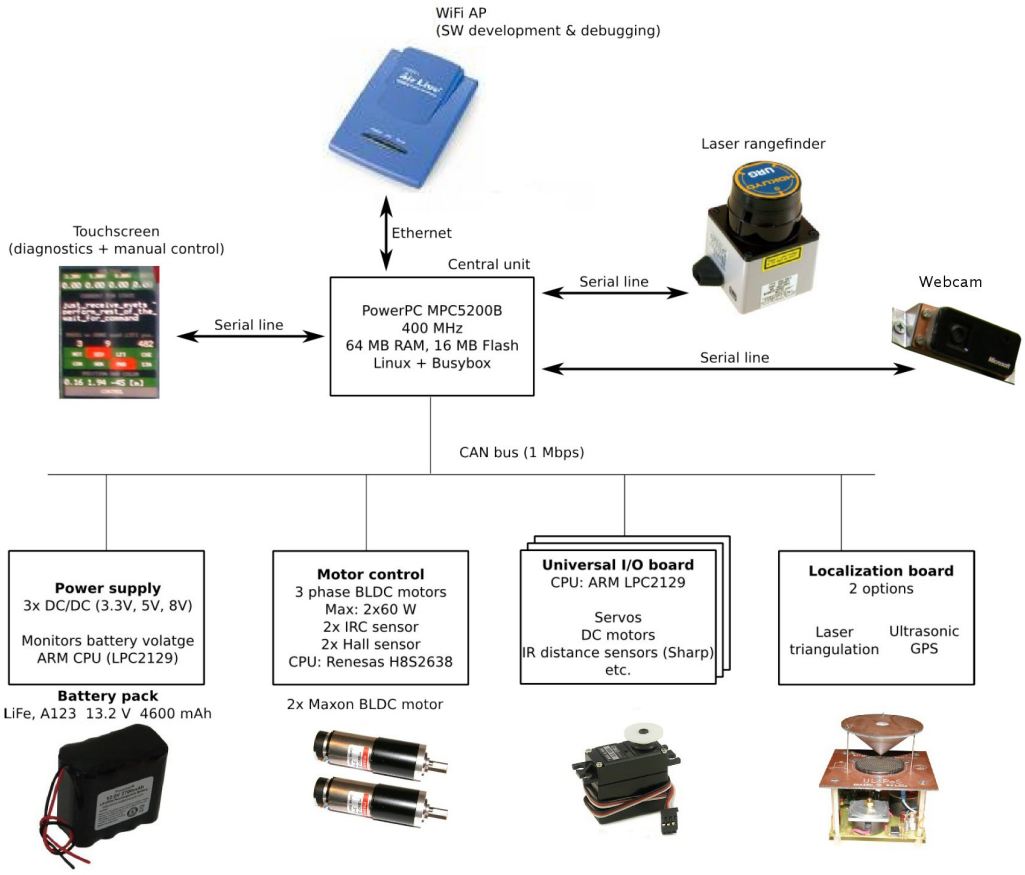
4.1 Hardwarová rozhraní hlavního palubního počítače

Hlavní rozhraní, které náš robot používá, jsou rozhraní sběrnice CAN, USB a ethernetové rozhraní. Jak je patrné z obr. 4.1, sběrnice CAN slouží pro komunikaci hlavního počítače robotu s jednotlivými periferními deskami, které mají např. na starosti řízení aktuátorů (řízení hlavních pohonů, motorů a serv použitých pro manipulaci s herními prvky a ovládání serva použitého pro naklápění rangefinderu Hokuyo). Na sběrnici CAN ovšem publikuje data o svém stavu i deska zajišťující napájení robotu nebo další důležitá součást – ultrazvukový lokalizační systém ULoPoS.

Dotykový displej, rangefinder Hokuyo a webová kamera, použitá pro rozpoznání vylosovaného rozložení puků na hřišti před startem soutěže, jsou k desce hlavního palubního počítače robotu připojeny prostřednictvím rozhraní USB.

Posledním zmíněným rozhraním je ethernet, jehož je využito k připojení palubního počítače k WiFi Access Pointu, rovněž nainstalovanému na palubě robotu. Ten umožňuje uživateli bezdrátové připojení k palubnímu počítači robotu.

[1] Díky použití ORTE pro sdílení dat mezi jednotlivými softwarovými komponentami ale může při vývoji stejně dobře běžet na počítači vývojáře a přitom např. ovládat skutečné aktuátory na robotu.



Obrázek 4.1: Hardwarová architektura robotu
(obrázek byl převzat z prezentace Michala Sojky pro kolokvium CAK [4] a doplněn)

4.2 Software hlavního palubního počítače

Na hlavním palubním počítači robotu běží kromě hlavního řídicího programu robotu další podpůrné programy, které jsou v tomto odstavci spolu se strukturou hlavního řídicího programu velmi stručně popsány.

Hlavní řídicí program robotu tvoří kromě stavových automatů běžících v jediném vlákně (pro ročník 2009 byla knihovna *FSM* upravena tak, aby umožňovala běh několika stavových automatů v jediném vlákně) a popsaných jednotlivě v úvodu kapitoly 5 také další vlákna. Je to vlákno regulátoru zajišťujícího sledování trajektorie (trajectory follower thread), vlákno zajišťující, že překážky ve sdílené mapě detekované rangefinderem Hokuyo jsou postupně „zapomínány“ a vlákno, které po uplynutí 90 sekund od startu soutěže robota zastaví. Poslední zmíněné vlákno je spouštěno přímo hlavním stavovým automatem. Přehled těchto vláken spolu s dalšími procesy důležitými z hlediska popsaných stavových automatů je uveden v tab. 4.1.

competition případně *homologation* – proces řídící chování robotu

stavové automaty běžící v rámci jediného vlákna:

main – hlavní stavový automat pro herní strategii

move – stavový automat pro koordinaci řízení pohybu robotu

act – stavový automat pro řízení mechanismů

display – stavový automat pro komunikaci s displejem

další vlákna:

trajectory follower thread

obstacle forgeting thread

vlákno pro kontrolu 90sekundového limitu

ortemanager – správce ORTE, který registruje jednotlivé publishery a subscribery a zajišťuje jejich komunikaci; udržuje spojení s ostatními instancemi programu *ortemanager* v síti

cand – program, který zajišťuje obousměrnou komunikaci po sběrnici CAN; zprávy přijaté po CANu publikuje na ORTE a naopak

rozpuk – program který v době před startem zápasu, když už je robot ve startovní oblasti v přesně definované poloze, zpracovává obrázky z webkamery a klasifikuje je, aby rozpoznal konfiguraci volných puků, která byla vylosována

Tabulka 4.1: Některé procesy běžící na robotu

Pro jednotlivé stavové automaty je možné zaregistrovat callback funkci, kterou potom knihovna *FSM* automaticky volá při změně jejich stavu. Tato callback funkce může např. publikovat informace o změně stavu příslušného stavového automatu prostřednictvím ORTE. Na našem robotu bylo toto použito pro stavové automaty *main*, *act* a *motion*, jejichž činnost bylo při ladění programů zejména potřeba sledovat.

4.2.1 Struktura robot

Ústředním místem v kódu řídícího programu robotu, napsaného v jazyce C, je struktura `struct robot`[2] obsahující data sdílená jednotlivými komponentami programu – stavovými automaty, částí obstarávající interakci s ORTE (modulem `robot_orte`) a dalšími.

Obsahuje mimo jiné také strukturu s daty hlavní smyčky stavových automatů `struct fsm_main_loop` nebo strukturu `fsm`, která sdružuje samotné struktury definující jednotlivé stavové automaty:

```
struct robot {
  /* ... */

  struct fsm_main_loop main_loop;

  /* jednotlive stavove automaty */
  struct {
    struct robo_fsm main;
    struct robo_fsm motion;
    struct robo_fsm display;
    struct robo_fsm act;
  } fsm

  /* ... */
};

extern struct robot robot;
```

[2] Viz výpis souboru robot.h na přiloženém CD.

Kapitola 5

Stavové automaty pro herní strategii a řízení aktuátorů

Část řídicího softwaru robotu je napsána formou stavových automatů. (Pojem stavový automat budu dále zkracovat jako FSM – finite state machine. Názvy konkrétních stavových automatů budu pro lepší čitelnost textu označovat kurzívou a malými písmeny: *act fsm*, *main fsm*). Na palubním počítači běží v jednom vlákně několik stavových automatů. Byly již vyjmenovány v kapitole 4 a jsou to hlavní stavový automat (*main fsm*), stavový automat koordinující řízení pohybu robotu (*move fsm*), stavový automat pro řízení manipulátorů (*act fsm*) a stavový automat obstarávající komunikaci s dotykovým displejem (*display fsm*).

Úkolem hlavního FSM je celkově řídit činnost robotu. Hlavní FSM například na začátku soutěže čeká, až bude zápas odstartován, a vydává pokyny k přesunu robotu po hřišti a ke sbírání puků. Hlavní FSM je tou částí řídicího programu robotu, která rozhoduje např. o tom, kdy je čas pro sbírání stavebních elementů a kdy je již na sbírání dalších puků pozdě, a tedy kdy je nutné pokusit se získat poslední body tím, že robot pojede vyložit již nasbírané puky k nejbližší konstrukční zóně.

Řízení pohybu robotu po hřišti koordinuje *move fsm*. Čeká na pokyn k přesunu robotu na zadané souřadnice. Po příchodu tohoto pokynu naplánuje trajektorii, ověří, zda tato trajektorie nekoliduje s překážkami na hřišti nebo s robotem protivníka, a iniciuje pohyb robotu po této nové trajektorii. Po úspěšném dokončení pohybu o tom podá zprávu hlavnímu FSM. V případě, že se na dráze objeví překážka (protivníkův robot), pokusí se o přeplánování trajektorie. Pokud je překážka v cílové oblasti (a nelze tedy naplánovat žádnou vyhovující trajektorii), sekundu počká a pokusí se o pohyb znovu.[1]

Podobně jako se *fsm move* stará o pohyb, automat *act fsm* řídí nakládání a vykládání puků. Hlavní stavový automat využívá služeb obou automatů *act fsm* a *fsm move* a koordinuje jejich práci, v tomto smyslu je jim nadřízen. Komunikuje s těmito automaty výměnou zpráv (událostí). V případě komunikace se stavovým automatem aktuátorů probíhá

[1] V případě, že tyto problémy přetrvávají, bylo by zřejmě vhodné, aby o tom *move fsm* zaslal odpovídající zprávu hlavnímu stavovému automatu; *move fsm* ve verzi použité v tomto ročníku soutěže ovšem tuto zprávu negeneroval. Záměrem bylo patrně ulehčit práci hlavnímu stavovému automatu. Tento aspekt spolupráce obou stavových automatů zůstal poněkud nedořešen.

tato komunikace přímo (*main fsm* vytváří události pro *act fsm* a naopak). Činnost automatu *fsm move* ovlivňuje hlavní stavový automat voláním funkcí pomocné knihovny *movehelper* (viz sekce 3.2).

Poslední stavový automat, stavový automat displeje, podporuje komunikaci displeje s palubním počítačem robotu. Displej je použit pro přehledné zobrazení stavu robotu a rovněž umožňuje základní nastavení, zejména umožňuje vybrat před startem barvu, za kterou bude robot soutěžit, nebo uměle vytvořit pro hlavní FSM událost odpovídající vytažení startovacího lanka. Lze ovšem říci, že zbytek řídicího programu je na tomto stavovém automatu nezávislý.

V části 5.1 této kapitoly je popsán stavový automat implementující herní strategii robotu (hlavní stavový automat) ve dvou svých verzích (viz dále) a v podkapitole 5.2 potom stavový automat pro řízení manipulátorů použitých pro nakládání a vykládání herních prvků. Stavový automat *move fsm* zde podrobně popsán nebude.

Pro implementaci všech zmíněných stavových automatů byla použita knihovna *FSM*, která byla popsána v odstavci 3.1.

5.1 Hlavní FSM

V určitém smyslu je hlavní stavový automat ústředním stavebním kamenem softwaru robotu. Primárním úkolem hlavního stavového automatu je řídit robot v souladu s pravidly soutěže tak, aby v zápase získal pokud možno nejvyšší bodové ohodnocení.

Před připuštěním do soutěže se robot musí zúčastnit vstupní prohlídky, označované v českých pravidlech jako *homologace*. Protože cílem homologace není, aby robot předvedl všechny své schopnosti a stavěl z herních prvků skutečné věže, a protože zkušenosti týmu z předchozích ročníků napovídaly, že by mohlo být vhodné připravit pro homologaci samostatné minimalistické řešení, bylo rozhodnuto připravit pro tyto účely samostatný stavový automat. Hlavní stavový automat má tedy dvě verze: homologační a soutěžní.

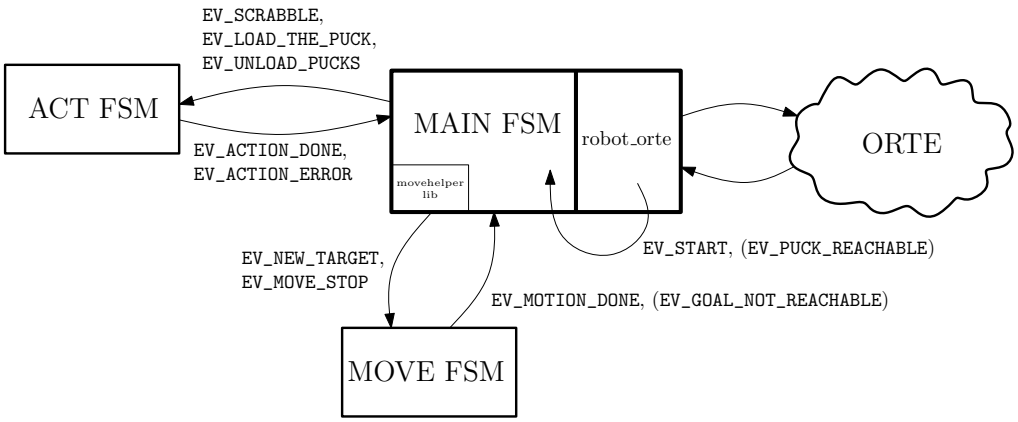
V části 5.1.1 jsou popsány události, které přijímají obě verze hlavního stavového automatu. Část 5.1.2 popisuje jednodušší homologační verzi stavového automatu a část 5.1.3 potom verzi soutěžní.

5.1.1 Události přijímané hlavním FSM

V úvodu kapitoly byl naznačen způsob spolupráce hlavního stavového automatu s automaty *act fsm* a *move fsm*. Jak bylo řečeno, hlavní FSM byl připraven ve dvou verzích. V této části je popsáno událostmi tvořené komunikační rozhraní, které je oběma verzím společné. Ilustrace komunikace hlavního FSM s okolím je na obr. 5.1.

V tab. 5.1 je přehled událostí, na které v různých stavech reaguje hlavní stavový automat. První z nich je událost `EV_START`, kterou generuje modul[2] `robot_orte` v okamžiku, kdy přijme příslušnou zprávu od ORTE. Událost odpovídá vytažení startovacího lanka z robotu.

[2]Modul ve smyslu komponenty programu napsaného v jazyce C. Zdrojový kód modulu `robot_orte` je v souboru `robot_orte.c` a rozhraní modulu v souboru `robot_orte.h`.



Obrázek 5.1: Spolupráce hlavního stavového automatu s okolím

č.	Název události	Původce	Význam
1.	`EV_START`	ORTE	došlo k vytažení startovacího lanka, zápas byl odstartován
2.	`EV_PUCK_REACHABLE`	ORTE	událost, která při použití dálkoměrného senzoru namísto mikrospínače v roli koncového spínače signalizovala přítomnost puku v dosahu chapadel robotu
3.	`EV_ACTION_DONE`	*act fsm*	stavový automat ovládající mechanismy *act fsm* signalizuje úspěšné vykonání akce
4.	`EV_ACTION_ERROR`	*act fsm*	*act fsm* signalizuje neúspěch při pokusu o vykonání akce
5.	`EV_MOTION_DONE`	*fsm move*	stavový automat koordinující řízení pohybu robotu (*fsm move*) signalizuje, že pohyb byl dokončen
6.	`EV_GOAL_NOT_REACHABLE`	*fsm move*	*fsm move* signalizuje, že cíl pohybu není dosažitelný (překážka v cílové oblasti); událost nebyla využita; *move fsm* použitý při soutěži generoval pouze událost `EV_MOTION_DONE`.

Tabulka 5.1: Přehled událostí přijímaných v různých stavech automatem *main fsm*; jejich původce z hlediska FSM a jejich význam

Událost druhá (`EV_PUCK_REACHABLE`) je uvedena pouze pro ilustraci. Ve výsledném hlavním stavovém automatu totiž nebyla využita, neboť reprezentuje událost vytvářenou na základě dat z dálkoměrného senzoru Sharp, který nakonec musel být nahrazen mikrospínačem. Umístění obou senzorů viz odstavec 5.2.1.

Události `EV_ACTION_DONE`, resp. `EV_ACTION_ERROR` představují zprávy od automatu ovládajícího mechanismy a informují o úspěšném provedení požadované akce resp. o chybě při pokusu o její vykonání. Bližší popis těchto událostí obsahuje kapitola 5.2. Podobně poskytuje hlavnímu FSM zpětnou vazbu *move fsm* prostřednictvím události `EV_MOTION_DONE`. Událost `EV_GOAL_NOT_REACHABLE`, uvedenou v tabulce 5.1, automat *move fsm* v tomto ročníku soutěže negeneroval ve snaze řešit problémy na své úrovni. Tato volba zřejmě nebyla úplně vydařená, neboť se tak robot mohl snadno ocitnout v situaci bez východiska.

5.1.2 Homologační hlavní FSM

Cíle homologační strategie a způsob jejich naplnění

Jak již bylo zmíněno, rozhodli jsme se jako tým pro samostatný zjednodušený hlavní stavový automat pro účely připouštěcí zkoušky (homologace). Myšlenka, kterou jsme tím sledovali, byla založena na potřebě projít homologací i v případě nefunkčnosti některých systémů robotu (např. nakládacích mechanismů).

Homologace je popsaná v pravidlech [1] v kapitole 7.1. Sestává ze dvou částí. První je fyzická prohlídka robotu, při které se kontroluje, zda robot vyhovuje předepsaným omezením na rozměry, umístění bezpečnostního tlačítka a podobně. Druhou částí je praktická zkouška robotu.

Odstavec 7.1.2 českých pravidel soutěže mimo jiné říká, že robot musí projít testem, ve kterém prokáže, že v soutěžních podmínkách bez protivníka

- je schopen opustit startovní oblast,
- je schopen vyhrát zápas bez přítomnosti oponenta a také že
- správně funguje jeho vypínací mechanismus (omezení na 90 s, viz kapitola 2).

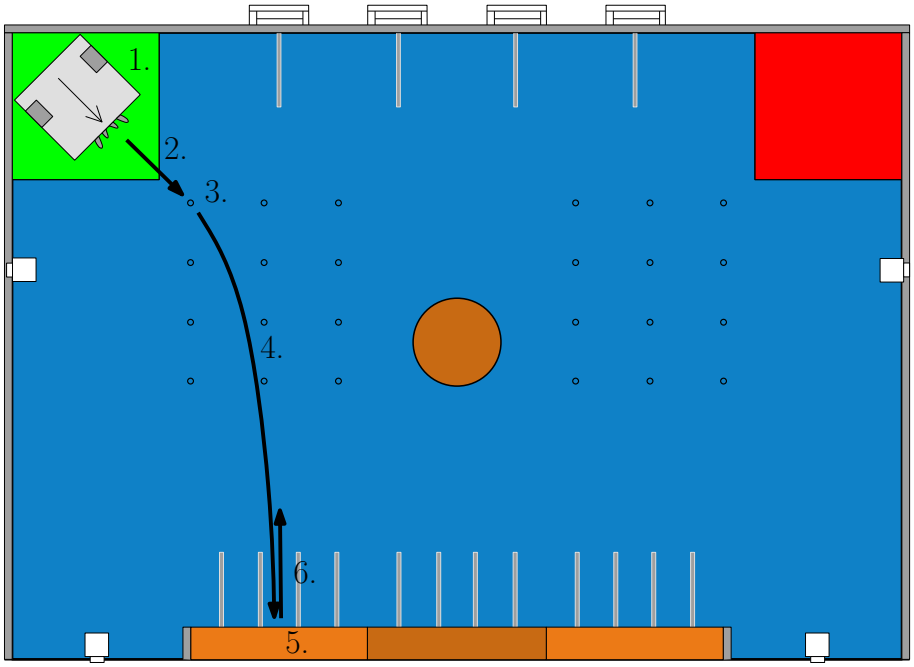
Jinými slovy to znamená, že robot musí být v homologaci schopen alespoň dotlačit nějaký (vlastní) herní element do některé z nevyvýšených konstrukčních zón a po vypršení 90sekundového limitu se musí bezpodmínečně zastavit.

Pravidla rovněž předepisují, že robot musí prokázat, že správně funguje systém vyhýbání se překážkám: *„robot musí být schopen úspěšně se vyhnout nepohyblivé překážce položené do cesty robotu, [...] čímž přesvědčivě ukáže, že překážku správně zaznamenal.“* Tento požadavek je splněn na úrovni plánování trajektorie, takže na úrovni hlavního stavového automatu jej nebylo potřeba řešit. (S využitím rangefinderu Hokuyo robot zaznamenává polohy překážek, které zapisuje do mapy ve sdílené paměti, algoritmus pro plánování trajektorie bere tyto překážky v potaz. Zvláštní vlákno zajišťuje, že „staré“ údaje o překážkách jsou zapomínány.)

Vypínací mechanismus musí zajistit, že na konci zápasu (po 90 sekundách) se robot přestane pohybovat. To je zajištěno vytvořením vlákna obsahujícího jediné volání funkce `sleep()` a volání funkce, která ukončí činnost jednotlivých stavových automatů:

```
/* vlakno zajistujici zastaveni robotu po vyprseni limitu */
void *wait_for_end(void *arg)
{
    sleep(COMPETITION_TIME); // COMPETITION_TIME == 90
    robot_exit();
    return NULL;
}
```

Popis homologační strategie



Obrázek 5.2: K popisu homologační strategie

Použitý homologační hlavní stavový automat implementuje nejjednodušší myslitelnou homologační strategii, která sestává pouze z šesti po sobě jdoucích kroků ilustrovaných čísly na obr. 5.2:

1. **Čekání na start:** Před startem je robot umístěn v definované poloze (obr. 5.2) ve startovací oblasti. Čeká na příchod signálu informujícího o vytažení startovacího lanka. Po jeho příchodu vyjede k nejbližšímu puku.

2. **Přesun k prvnímu puku** spočívá pouze v dopředné jízdě, kterou se robot přesune o 42 cm. Dorazí k puku, který je nejblíže startovací oblasti a který se na tomto místě nachází v kterékoliv z deseti konfigurací *volných puků*.[3]

[3] Termín *volné puky* označuje puky umístěné na ploše hřiště; je vysvětlen na straně 3 v kapitole 2.

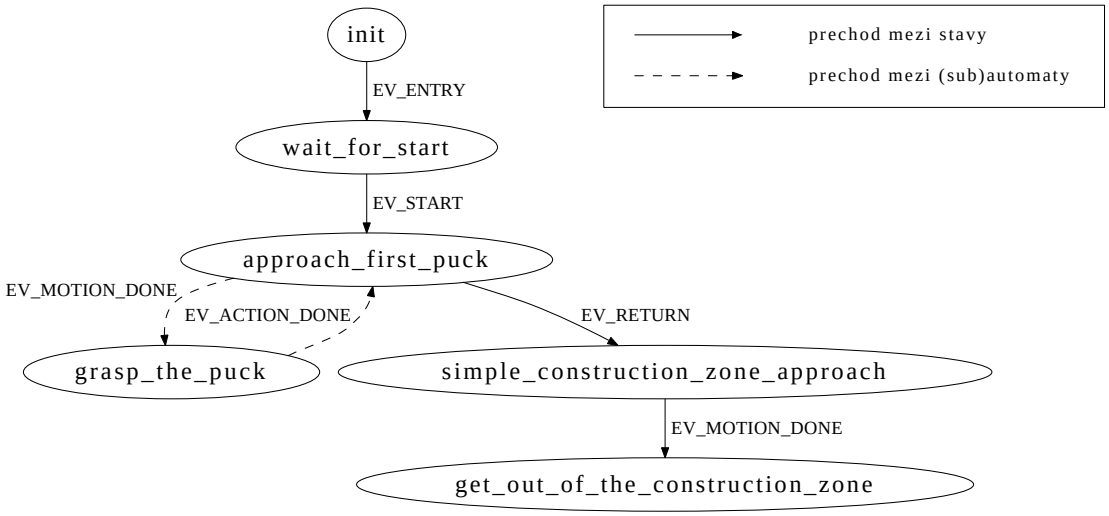
3. **Nabrání prvního puku:** Robot „uchopí“ první puk. V homologační verzi byla použita odlišná varianta stavového automatu *act fsm*, obsahující funkcionalitu *uchopení* a *„vyplivnutí“* puku. Uchopení spočívá v pouhém stisknutí puku chapadly a jeho „nasátí“ na lžíci pomocí pásů. Vyplivnutí je proces přesně opačný.

4. **Přejezd k nevyvýšené konstrukční zóně:** Robot se následně přesune k nejbližší nevyvýšené konstrukční zóně.

5. **Vyložení puku:** Dojde k „vyplivnutí“ puku.

6. **Vycouvání z konstrukční zóny:** V posledním kroku robot pomalu vycouvá z konstrukční zóny.

Popis homologačního stavového automatu



Obrázek 5.3: Stavy homologační verze automatu *main fsm*

Počátečním stavem homologačního FSM je stav `init`, ve kterém se pouze inicializují proměnné (tato inicializace by mohla být provedena i ve funkci `main()`). Po provedení inicializace dochází okamžitě k přechodu do stavu `wait_for_start` .

`wait_for_start` je stav, ve kterém se čeká na příchod události `EV_START`, reprezentující vytažení lanka ze startovacího konektoru. Robot se před startem nachází v definované poloze, která je regulátoru zajišťujícímu pohyb předána voláním funkce z knihovny *movehelper*, označené `robot_set_est_pos_trans()`, s příslušnými parametry. Po příchodu události signalizující vytažení startovacího lanka se odstartuje vlákno pro kontrolu 90 sekundového časového limitu voláním `pthread_create()`, kde jedním z parametrů je výše uvedená funkce `wait_for_end()`, a poté se vyvolá přechod do dalšího stavu.

Tím stavem je `approach_first_puck`, ve kterém robot přejíždí ze startovací oblasti k nejblíže umístěnému puku, který se na hřišti nachází vždy (v každé možné kombinaci rozmístění *volných puků*). Pohyb je při vstupu do tohoto stavu zahájen voláním funkce

`robot_move_by()` knihovny *movehelper*.[4] Poté co hlavní stavový automat obdrží událost `EV_MOTION_DONE`, signalizující, že pohyb (přesun k prvnímu puku) byl dokončen, zajistí naložení puku. Přitom (dočasně) opustí stav `approach_first_puck` voláním *subautomatu* tvořeného jediným stavem `grasp_the_puck`. Poté, co subautomat dokončí svoji činnost, je v původním stavu volajícího automatu (`approach_first_puck`) vygenerována událost `EV_RETURN`.

Při vstupu do stavu (subautomatu) `grasp_the_puck` posílá hlavní stavový automat automatu *act fsm* událost `EV_GRASP_THE_PUCK` (žádost o uchopení puku) a dále v tomto stavu čeká na příchod události `EV_ACTION_DONE`, signalizující úspěšné „uchopení“ puku. Na tuto událost reaguje hlavní FSM opuštěním subautomatu. Ve stavu `approach_first_puck` tím dojde k vygenerování události `EV_RETURN`, jejíž příchod je v tomto stavu ošetřen přechodem do stavu `simple_construction_zone_approach`.

Stav `simple_construction_zone_approach` slouží k zajištění přesunu robotu k nevyvýšené konstrukční zóně. Při vstupu do tohoto stavu je iniciován pohyb a příchod události `EV_MOTION_DONE` je ošetřen přechodem do stavu `get_out_of_the_construction_zone`.

V tomto posledním stavu homologačního stavového automatu je puk vyplivnut. Při vstupu do něj je automatu *act fsm* předána událost `EV_UNLOAD_PUCKS`, která v tomto případě způsobí „vyplivnutí“ puku.

5.1.3 Soutěžní verze hlavního FSM

Volba strategie

Pravidla popsaná v kapitole 2 dávají „velký počet stupňů volnosti“ možných řešení, pokud jde o herní strategii. Již záhy jsme se jako tým rozhodli, že nevyužijeme podavačů puků, jejichž pozice se před startem losuje. Další rozhodnutí už nebyla tak svobodná a byla podstatně ovlivněna mechanickými omezeními robotu.

Protože v době práce na stavových automatech již byla mechanická konstrukce robotu víceméně daná, byli jsme při volbě strategie ovlivněni rychlostí, s jakou byl robot schopný nabírat puky na hřišti. Zjištění, že zvolená konstrukce nabíracích mechanismů robotu v podstatě neumožňuje spolehlivě nabírat sloupové herní elementy ze zásobníků na krajích hřiště, bylo nepříjemným překvapením. Stejně tak objev, že nejsme schopni při plném nákladu čtyř sloupových elementů zároveň převážet trámek a úspěšně ho vyložit.

V průběhu příprav jsme rovněž zjistili, že za daných 90 sekund není schopen náš robot stihnout o mnoho více, než na hřišti posbírat čtyři *volné puky* (puky z šestice rozmístěné na vlastní polovině plochy hřiště) a postavit z nich jedinou věž umístěnou na centrální konstrukční zóně hřiště (*akropoli*). Sled nepříjemných zjištění jsme zakončili objevem učiněným pár dní před soutěží, a sice objevem faktu, že náš robot není ve skutečnosti schopen vykládat platně umístěné puky na mírně vyvýšenou prostřední obdélníkovou konstrukční zónu. Aby to dokázal, musel by mít vytlačovač puků (viz popis mechanismů v kapitole 5.2.1) větší rozsah

[4] Zde je slovy *„při vstupu do stavu `X` automat udělá ...“* myšleno *„reakcí automatu ve stavu `X` na příchod události `EV_ENTRY` je ...“*. Podobně i dále v textu.

pohybu a musel by umět vytlačit puky zcela mimo lžíci. V použitém uspořádání mechanismů bohužel robot nedokázal umístit puk tak, aby ležel naplocho a *plně* (celou plochou) v inkriminované konstrukční zóně, jak vyžadují pravidla [1].

Učinili jsme tato rozhodnutí:

- zcela ignorovat trámky, tedy
  - zavrhnout možnost vložit před startem do robotu volitelný trámek,
  - nepoužívat ani trámky v držácích na dlouhé straně hřiště;
- sbírat pouze volné puky a nestarat se o puky v zásobnících;
- stavět nasbírané puky na akropoli.

Zmíněné body se staly základem implementované strategie, která je vlastně velmi jednoduchá: Robot sbírá volné puky a poté, co nabere čtyři, jede je vyložit k akropoli. Poté sesbírá zbylé dva volné puky a vyloží je na akropoli vedle již postavených.

Rámcový popis strategie Pro každou konfiguraci volných puků je nadefinováno pořadí, ve kterém má robot puky sbírat. Pro každý puk v této sekvenci je nadefinován příjezdový úhel, z něhož se robot k puku přiblíží. Hodnota tohoto úhlu ovlivňuje trajektorii, po které se robot bude od jednoho puku ke druhému přesouvat.

1. **Čekání na start:** Robot je před startem umístěn ve startovací oblasti ve stejné poloze jako při homologaci. Program *rozpuk* přitom určuje vylosovanou konfiguraci volných puků na základě dat z kamery. Robot čeká na příchod signálu informujícího o vytažení startovacího lanka. Po jeho příchodu vyjede sbírat puky.
2. **Sbírání volných puků:** V každém případě robot nejdříve vyjede k puku, který se nachází na pozici v mřížce nejbližší ke startovací oblasti, protože tento puk se nachází v každé z deseti kombinací volných puků. V určeném pořadí nasbírá čtyři puky (případně dva, pokud tento krok již opakuje).
3. **Vyložení nasbíraných puků na akropoli:** Robot se přesune k akropoli. Přiblíží se k ní ze směru, který je závislý na vylosované konfiguraci puků a na pořadovém čísle posledního naloženého puku, a vyloží naložené puky.
4. **Opakování:** Zbývají-li „nezpracované“ puky, opakují se kroky 2. a 3. V případě, že již na sbírání zbývá málo času (zápas se blíží ke konci) a robot má již nějaké puky naloženy, provádí se rovnou krok 3.

`robot.pucks_inside` – počet puků naložených do robotu (0–4); globální proměnná uchovaná ve struktuře `robot`; *zdroj:* stavový automat *act fsm*

`robot.game_conf` – číslo (index) konfigurace puků identifikované programem *rozpuk* (hodnoty `0`–`9` odpovídají číslům kombinace 1–10, jak jsou uvedeny v dodatku A); hodnota je do struktury `struct robot` ukládána v zaregistrované ORTE callback funkci; *zdroj:* program *rozpuk* prostřednictvím ORTE

`free_puck_to_try_to_get_next` – pořadové číslo následujícího *volného puku*, který robot pojede sebrat. Číslování začíná od nuly; *zdroj:* hlavní stavový automat

`short_time_to_end` – booleovská proměnná, které je na začátku zápasu přiřazena hodnota `false`. Po uplynutí 60 s je hodnota této proměnné ve zvláštním časovacím vlákně nastavena na `true`, což signalizuje, že do konce zápasu již zbývá málo času a robot by se při nejbližší příležitosti měl pokusit získat kladné body vyložením nastřádaného nákladu v konstrukční zóně; *zdroj:* vlákno spuštěné při startu zápasu

`free_puck_pick_up_sequence` – je statické pole $10 \times 6 \times 3$ celých čísel (`int`) definující pořadí, ve kterém bude robot sbírat volné puky, a úhel, ze kterého bude ke každému z nich najíždět; pole obsahuje pro každou konfiguraci volných puků šest trojic hodnot (každá z deseti konfigurací obsahuje šest puků, pro každý puk z každé konfigurace obsahuje toto pole trojici hodnot); v trojici (n_x, n_y, φ) popisují čísla n_x a n_y index puku v mřížce 3×4 a mohou nabývat hodnot $n_x \in \langle 0, 2 \rangle$, $n_y \in \langle 0, 3 \rangle$, φ je příjezdový úhel ve stupních (viz obr. 5.4(b)); *zdroj:* konstantní hodnota

`preferred_acropolis_approach_angles` – pole 10×3 proměnných `int`, uchovávající pro každou konfiguraci volných puků tři hodnoty příjezdových úhlů k akropoli; *zdroj:* konstantní hodnota

Tabulka 5.2: Globální proměnné a konstanty, s nimiž *main fsm* pracuje; údaj *zdroj* označuje místo, kde se proměnná nastavuje

Globální proměnné hlavního FSM

Výsledný stavový automat používá několik globálních proměnných a předdefinovaných konstant, jejichž význam je vysvětlen v této části. Shrnuty jsou v tab. 5.2.

Detekci vylosované konfigurace puků provádí program *rozpuk* před startem. Výsledkem je číslo, které publikuje prostřednictvím ORTE. Modul[5] `robot_orte` zajišťuje uložení této hodnoty do proměnné `robot.game_conf` ve struktuře `robot`. Po odstartování zápasu se již hodnota této proměnné, ze které hodnotu čte i *main fsm*, nemění.

Druhou proměnnou umístěnou ve struktuře `robot` je proměnná `robot.pucks_inside`, která uchovává počet puků naložených v robotu. Hodnotu zapisuje stavový automat mechanismů. Hlavní stavový automat ji pouze čte.

V proměnné `free_puck_to_try_to_get_next` si *main fsm* pamatuje pořadové číslo volného puku, který příště pojede naložit. Puky jsou v této proměnné číslovány od nuly, tedy na startu tato proměnná obsahuje hodnotu `0`.

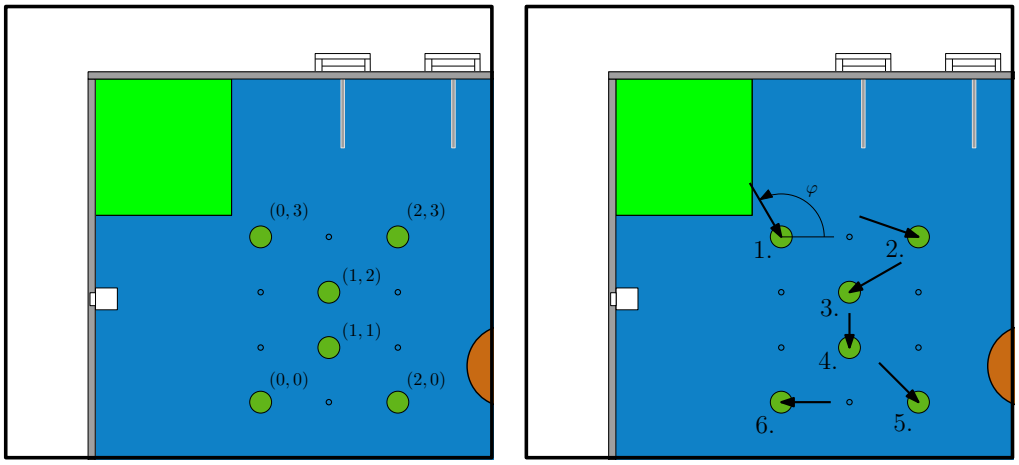
Aby bylo možné ošetřit situaci, kdy do konce zápasu zbývá již málo času, byla zavedena na úrovni hlavního FSM globální proměnná `short_time_to_end`. V inicializačním stavu hlavního FSM je jí přiřazena hodnota `false`. Ve vlákně, podobném tomu použitému v homologačním FSM, je po vypršení časovače nastavena na `true`.

Konečně zde zmíníme dvě konstanty sloužící k definici způsobu, jímž robot při soutěži přejíždí mezi volnými puky při jejich sbírání, a k definici úhlů, ze kterých robot přijíždí k akropoli při vykládání nasbíraných puků. Nejprve ovšem připomeňme, že úloha je pro zelený a červený robot symetrická. Níže uvedené offsety puků slouží k přepočtu na souřadnice puků na hřišti v soustavě zavedené v odstavci 3.2. Výsledné souřadnice jsou použity pro povely k pohybu robotu. Hlavní FSM pracuje pouze s variantou pro zeleného hráče. V případě hry za červeného jsou souřadnice přepočítávány na úrovni knihovny *movehelper*.

První ze zmíněných konstant definuje pořadí, ve kterém robot volné puky sbírá. Je nazvána `free_puck_pick_up_sequence` a je to statické pole celých čísel (`int`) o rozměrech $10 \times 6 \times 3$. Pro každou konfiguraci volných puků (viz příloha A) uchovává šest trojic čísel; jednu trojici pro každý volný puk v dané konfiguraci. V takové uspořádané trojici (n_x, n_y, φ) označuje dvojice (n_x, n_y) offset polohy puku počítaný vzhledem k puku s nejmenší x-ovou i y-ovou souřadnicí v soustavě zavedené ve odstavci 3.2. Tento „referenční“ puk má tedy offset $(n_x, n_y) = (0, 0)$. Jemu nejbližší puk se stejnou y-ovou souřadnicí má offset $(1, 0)$ a například puk umístěný nejblíže startovní oblasti (zelené) má offset $(0, 3)$. Příklady offsetů jsou uvedeny na obrázku 5.4(a), kde je offsety označena konfigurace puků s číslem 3. Třetí číslo v trojici, φ, označuje úhel, ze kterého bude robot při sbírání k puku přijíždět. Způsob zavedení tohoto úhlu pro první puk ze třetí konfigurace volných puků je vyznačen na obr. 5.4(b).

Pole `free_puck_pick_up_sequence` tedy jednak uchovává informaci o tom, kde jsou v dané konfiguraci puků umístěny volné puky. Dále uchovává informaci o tom, v jakém pořadí je bude robot sbírat (jako první bude sbírán puk s offsetem z první trojice čísel, jako druhý puk s offsetem z druhé trojice atd.) Pole také obsahuje informaci o tom, z které strany bude robot ke kterému puku přijíždět. Protože robot sbírá puky jeden za druhým (přinejmenším první čtyři), umožňuje nastavení příjezdových úhlů ovlivnit trajektorii, po které se robot při přejezdu mezi nimi pohybuje.

[5] modul ve smyslu komponenta programu napsaného v jazyce C



(a) Indexování puků: u jednotlivých puků jsou zobrazeny hodnoty (n_x, n_y)

(b) Pořadí sbírání puků a směry příjezdu s vyznačeným příjezdovým úhlem u prvního puku

Obrázek 5.4: Ilustrace hodnot `free_puck_pick_up_sequence` pro konfiguraci puků č. 3

```
// pole definujici
//  - polohu volnych puku v jednotlivych konfiguracich
//  - poradi jejich sbirani
//  - prijezdove uhly k jednotlivym volnym pukum
const int free_puck_pick_up_sequence[][6][3] = {
  /* nx, ny, prijezdovy uhel */

  /* ... vynechane konfigurace 1., 2. */

  /* konfigurace c. 3: */
  {
    {0, 3, 125},
    {2, 3, 160},
    {1, 2, 40},
    {1, 1, 90},
    {2, 0, 135},
    {0, 0, 0},
  },

  /* ... vynechane konfigurace 4.-10. */
}
```

Druhá ze zmíněných konstant se jmenuje `preferred_acropolis_approach_angles`. Je to opět pole celočíselných hodnot, tentokrát o rozměrech 10×3. Pro každou konfiguraci volných puků obsahuje tři úhly ve stupních. Hodnoty uchované v poli označují úhly, ze kterých bude robot přijíždět k akropoli vykládat puky poté, co naloží k-tý puk. První úhel z trojice platí pro $k \in \{1, 2\}$, druhý úhel pro $k \in \{3, 4\}$ a poslední úhel platí pro $k \in \{5, 6\}$.

Způsob zavedení tohoto „příjezdového úhlu k akropoli“ je stejný jako způsob zavedení výše popsaného příjezdového úhlu k puku (viz φ na obr. 5.4(b)).

```
// definice tri prijezdovych uhlu k akropoli
// pro kazdou konfiguraci volnych puku
const int preferred_acropolis_approach_angles[][3] = {
  {180, 180, 220}, /* konfigurace 1. */
  {130, 160, 200}, /* konfigurace 2. */
  {130, 160, 200}, /* konfigurace 3. */
  /* ... prijezdove uhly pro konfigurace 4.-10. vynechany */
}
```

Učit robot s naším stavovým automatem tedy znamená ladit hodnoty v tomto poli čísel.

Popis použitých funkcí

Soutěžní verze hlavního FSM používá několik jednoduchých funkcí. Jednou z nich je funkce `timing_thread()`, tvořící časovací vlákno spouštěné po odstartování zápasu. To nastavuje po 60 sekundách hodnotu `short_time_to_end` na `true` a po 90 sekundách způsobí zastavení všech stavových automatů a zastavení robotu.

Další dvě volají funkci `robot_moveto_trans()` z pomocné knihovny *movehelper* a předtím pouze přepočítávají souřadnice, se kterými ji volají. Je to funkce pro vyvolání přesunu robotu k akropoli, `robot_goto_acropolis(int phi)` (parametr `phi` je příjezdový úhel zavedený stejným způsobem jako příjezdový úhel k pukům znázorněný na obr. 5.4(b)).

Druhá funkce vyvolává přesun robotu k určenému volnému puku z dané konfigurace. Jmenuje se `robot_goto_next_puck_in_sequence(int lot, int puck_number)`. Parametr `lot` odpovídá „indexu“ konfigurace (číslo `0..9`) a parametr `puck_number` pořadovému číslu puku (1.–6. puk: hodnoty `0..5`).

Popis stavového automatu

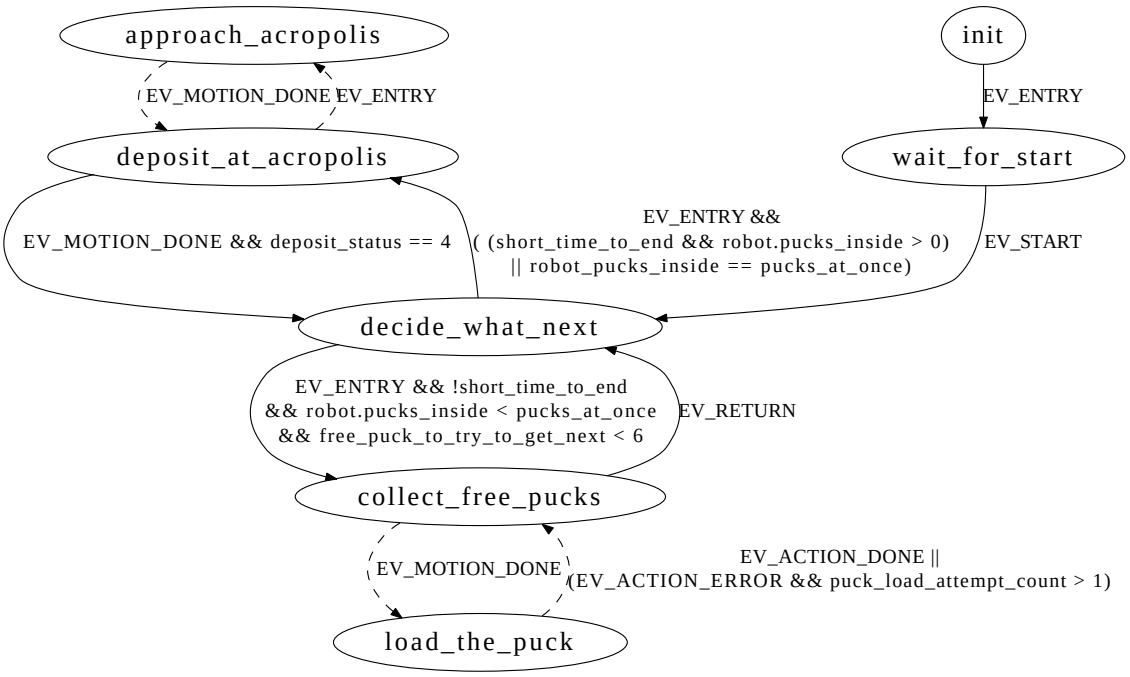
Výchozím stavem soutěžního hlavního stavového automatu je stav opět pojmenovaný `init`. Při vstupu do tohoto stavu je publikováním příslušné zprávy na ORTE (voláním funkce z knihovny *actlib*) zajištěno, že začne pracovat program *rozpuk*, který běží rovněž na hlavním palubním počítači, obsluhuje kameru a klasifikuje získané obrázky za účelem rozpoznání vylosované konfigurace puků. Protože po odstartování zápasu již program *rozpuk* není užitečný a zároveň je poměrně náročný na výpočetní výkon, je později po startu, kdy už není potřeba, jeho činnost pozastavena, opět voláním příslušné funkce z knihovny *actlib*.

Rovněž jsou při vstupu do stavu `init` inicializovány proměnné

```
short_time_to_end = false
free_puck_to_try_to_get_next = 0
```

a rovnou je vyvolán přechod do následujícího stavu `wait_for_start`.

Ve stavu `wait_for_start` se čeká na příchod události `EV_START` (vytažení startovacího lanka z konektoru). Po jejím příchodu je pozastavena činnost programu `rozpuk`, je vytvořeno časovací vlákno, které mimo jiné po uplynutí šedesáti sekund nastaví hodnotu `short_time_to_end` na `true` a je vyvolán přechod do stavu `decide_what_next`.



Obrázek 5.5: Stavy soutěžní verze stavového automatu *main fsm*

Rozhodování Stav `decide_what_next` je jakýmsi centrálním stavem, do něhož se stavový automat vrací „pokaždé“, když provede nějakou akci. Rozhoduje se v něm, co bude další akcí robotu. Tou může být buď sbírání *volných puků* nebo vykládání puků na *akropoli* (kde je to bodově nejvýhodnější). Toto rozhodování se provádí ihned při vstupu do stavu (ošetřením vstupní události `EV_ENTRY`) a je založeno na hodnotách `free_puck_to_try_to_get_next`, `short_time_to_end` a `robot.pucks_inside`.

V případě, kdy má `short_time_to_end` hodnotu `true` (což není případ okamžiku, kdy zápas právě začal) se nejprve ověří hodnota proměnné `robot.pucks_inside`. Je-li kladná (robot má naložen nejméně jeden puk), automat přejde do stavu `deposit_at_acropolis` (robot vyrazí pokusit se vyložit náklad na akropoli). Je-li `robot.pucks_inside == 0` (nejsou naloženy žádné puky) a zbývá-li na hřišti nějaký volný puk (tzn. hodnota proměnné `free_puck_to_try_to_get_next < 6`), přejde automat do stavu `collect_free_pucks`.

Má-li proměnná `short_time_to_end` hodnotu `false`, rozhoduje se *main fsm* takto. Je-li robot plně naložen (`robot.pucks_inside == 4`), vyvolá přechod do stavu pro vykládání puků `deposit_at_acropolis`. V opačném případě se vydává sbírat další puky, jsou-li ještě na hřišti nějaké (FSM přechází do stavu `collect_free_pucks`). V případě že nejsou a v robotu je alespoň jeden puk, přechází FSM do stavu `deposit_at_acropolis`.

Nakládání volných puků je obstaráno dvěma stavy: stavem `collect_free_pucks` a stavem `load_the_puck`. Druhý z nich je zároveň subautomatem (z toho důvodu, aby mohl být případně v nezměněné podobě „volán“ z jiného stavu než z `collect_free_pucks`, kdyby byla herní strategie rozšířena). Stav `collect_free_pucks` obstarává přesun k dalšímu volnému

puku v pořadí a „volá“ subautomat tvořený jediným stavem `load_the_puck`, aby zajistil naložení puku do robotu.

Ve stavu `collect_free_pucks` je vstupní událost `EV_ENTRY` ošetřena voláním funkce `robot_goto_next_puck_in_sequence`, která způsobí přesun robotu k dalšímu volnému puku v pořadí (pořadové číslo puku je určeno parametrem `free_puck_to_try_to_get_next`, který je funkci předán spolu s parametrem `robot.game_conf`). Poté co robot pohyb úspěšně dokončí, vygeneruje *move fsm* pro hlavní FSM událost `EV_MOTION_DONE`. Ta je v popisovaném stavu ošetřena přechodem do stavu (subautomatu) `load_the_puck`, v němž se učiní pokus o naložení puku. Po naložení puku (případně po druhém neúspěšném pokusu o jeho naložení) se *main fsm* opět přesune do stavu `collect_free_pucks` a v něm je generována událost návratu z „volaného“ subautomatu, událost `EV_RETURN`. Při příchodu této události je inkrementována hodnota proměnné `free_puck_to_try_to_get_next` a je vyvolán přechod nazpět do „rozhodovacího“ stavu `decide_what_next`.

V předchozím odstavci byl přeskočen popis stavu `load_the_puck`, který obstarává komunikaci se stavovým automatem mechanismů (*act fsm*) a ve kterém robot činí pokus o naložení puku. V tomto stavu je využita lokální proměnná `puck_load_attempt_count` uchovávající údaj, ke kolika pokusům o naložení došlo (v rámci momentálního pobytu ve stavu `load_the_puck`). Při vstupu do stavu je tato proměnná nastavena na `0`, automatu mechanismů je poslán signál `EV_SCRABBLE` a je nastaven časovač na krátkou časovou hodnotu 200 ms. Po příchodu události `EV_TIMER` od tohoto časovače je volána funkce `robot_move_by` knihovny *movehelper* s parametry `0.02` (přesun o dva centimetry vpřed), `NO_TURN()` (pohyb beze změny směru) a odkazem na předdefinovou `TrajectoryConstraints` strukturu `&tcSlow` (robot se bude pohybovat pomalu).[6] Dokončení krátkého přesunu robotu je signalizováno příchodem události `EV_MOTION_DONE` od automatu *move fsm*. Událost `EV_MOTION_DONE` je ošetřena zasláním události `EV_LOAD_THE_PUCK` automatu *act fsm*, tedy je požadováno naložení puku. Automat mechanismů se pokusí nabrat puk a v případě úspěchu posílá hlavnímu FSM událost `EV_ACTION_DONE`. V tom případě je rovnou vyvolán návrat do „volajícího“ automatu, tedy návrat do stavu `collect_free_pucks`. Pokud se mechanismům nepodaří puk naložit, signalizují to událostí `EV_ACTION_ERROR`. V případě příchodu této události ve stavu `load_the_puck` automat *main fsm* nejprve inkrementuje hodnotu lokální proměnné `puck_load_attempt_count`. Pokud se to stalo poprvé (`puck_load_attempt_count == 1`), je opět voláním funkce `robot_move_by()` požadován posun robotu vpřed ve víře, že puk se podaří naložit o něco dále. Pokud ovšem událost `EV_ACTION_ERROR` přišla podruhé (tedy po její inkrementaci je `puck_load_attempt_count++ > 1`), je podnícen návrat do volajícího automatu.

Vykládání puků na akropoli obstarávají v hlavním FSM stavy `deposit_at_acropolis` a `approach_acropolis`. Stav `approach_acropolis` opět tvoří subautomat o jediném stavu a dochází v něm pouze k přesunu robotu k akropoli. Akropole je kruhová a směr, ze kterého se k ní robot po nasbírání určitého počtu volných puků bude přibližovat, je dán hodnotou uloženou v poli `preferred_acropolis_approach_angles` pro danou konfiguraci puků a pro daný puk. Při vstupu do stavu `approach_acropolis` je pouze volána funkce

[6] Význam struktury `TrajectoryConstraints` při plánování trajekotrie viz odstavec 3.2 o knihovně *movehelper*.

`robot_goto_acropolis()` a čeká se na příchod události `EV_MOTION_DONE`. Jakmile událost přijde, je vyvolán návrat do volajícího automatu.

Vykládání je celé v režii stavu `deposit_at_acropolis`. Robot nejprve přijede před akropoli, hlavní automat vyšle automatu *act fsm* signál `EV_PREPARE_THE_UNLOAD` a ten připraví naložené puky na lžíci a lžíci s puky přesune do výšky nad úroveň akropole (podobně jako vysokozdvižný vozík při vykládání nákladu). Poté robot popojede dopředu, lžící nad akropoli. Hlavní automat vyšle automatu mechanismů signál `EV_UNLOAD_PUCKS` a ten zajistí, že lžíce sjede o kousek níže a vytlačovač vytlačí puky ven z robotu.[7] Aby se zajistilo, že nejnižší vyložený puk leží celou svou plochou na akropoli, provede se ještě následující manévr: robot zacouvá a velmi pomalu opět popojede dopředu, aby posunul lžící puky blíže ke středu akropole. Lžíce z hliníkového plechu je pružná, takže její hrana je po vyložení puků nad dolní hranou nejnižšího puku a je možné lžící puk tlačit beze změny polohy výtahu. Poté se vykládací procedura dokončí. Robot vycouvá tak, že lžíce je zcela mimo prostor akropole, a *main fsm* vydá automatu *act fsm* signál `EV_FREE_SPACE`, což znamená, že *act fsm* může lžíci přesunout do transportní polohy. Ke koordinaci je ve zmíněném stavu použita lokální proměnná typu `int deposit_status`. Pro srovnání uvádím definici stavu:

```
FSM_STATE(deposit_at_acropolis)
{
  static int deposit_status;
  switch (FSM_EVENT) {
    case EV_ENTRY:
      deposit_status = 0;
      SUBFSM_TRANSITION(approach_acropolis, NULL);
      break;
    case EV_RETURN: {
        int floor = 2;
        FSM_SIGNAL(ACT, EV_PREPARE_THE_UNLOAD, (void*)floor);
      }
      break;
    case EV_ACTION_DONE:
      switch(deposit_status) {
      case 0:
        // udalost prisla v reakci na EV_PREPARE_THE_UNLOAD
        robot_move_by(0.11, NO_TURN(), &tcSlow);
        break;
      case 1:
        // udalost prisla v reakci na EV_UNLOAD_PUCKS
        robot_move_by(-0.07, NO_TURN(), &tcSlow);
        break;
      }
      deposit_status++;
      break;
    case EV_MOTION_DONE:
      switch(deposit_status) {
```

[7] Podrobný popis mechanismů je v kapitole 5.2.1.

```
      case 1:
        FSM_SIGNAL(ACT, EV_UNLOAD_PUCKS, NULL);
        break;
      case 2:
        robot_move_by(0.08, NO_TURN(), &tcVerySlow);
        deposit_status++;
        break;
      case 3:
        robot_move_by(-0.15, NO_TURN(), &tcSlow);
        deposit_status++;
        break;
      case 4:
        FSM_SIGNAL(ACT, EV_FREE_SPACE, NULL);
        FSM_TRANSITION(decide_what_next);
        deposit_status++;
      }
      break;
    case EV_TIMER:
    case EV_LASER_POWER:
    case EV_STACK_FULL:
    case EV_ACTION_ERROR:
    case EV_START:
      DBG_PRINT_EVENT("unhandled event");
      break;
    case EV_EXIT:
      break;
  }
}
```

Při vstupu do stavu `deposit_at_acropolis` je nejdříve nastavena pomocná lokální proměnná `deposit_status` na hodnotu `0` a je volán subautomat `approach_acropolis`. Poté co tento subautomat skončí svoji práci, je v rámci ošetření události `EV_RETURN` ve stavu `deposit_at_acropolis` vyslán automatu *act fsm* signál `EV_PREPARE_THE_UNLOAD` s parametrem `2` (akropole má výšku dvou puků). Automat mechanismů poté provede přípravu na vykládání puků ve druhém „patře“ (lžíci s naloženými puky přesune do patřičné výšky) a po ukončení této procedury pošle hlavnímu automatu událost `EV_ACTION_DONE`. Při zpracování této události se *main fsm* řídí hodnotou proměnné `deposit_status`, která je v tomto okamžiku stále rovna nule. Proto volá funkci `robot_move_by()` s parametry odpovídajícími dopřednému přesunu a také je inkrementována hodnota `deposit_status`. Po dokončení pohybu přijde automatu *main fsm* událost `EV_MOTION_DONE`. Chování hlavního automatu ve stavu `deposit_at_acropolis` při příchodu této události je opět závislé na hodnotě `deposit_status`. V případě že událost `EV_MOTION_DONE` přichází v tomto stavu poprvé (`deposit_status == 1`), je vyslán automatu *act fsm* signál `EV_UNLOAD_PUCKS`. Po fyzickém vyložení puků opět přichází událost `EV_ACTION_DONE` (a platí `deposit_status == 1`), a proto je vyžádán pohyb zpět a proměnná `deposit_status` je inkrementována. Po příchodu události `EV_MOTION_DONE` je opět vyžadován pohyb, tentokrát pomalý dopředný (zatlačení

puků), zvýšena hodnota `deposit_status`. Příchod události `EV_MOTION_DONE` se opakuje, následuje poslední požadavek na pohyb (vycouvání – odjezd) a zvýšení hodnoty proměnné `deposit_status`. V momentě, kdy událost `EV_MOTION_DONE` přijde a hodnota proměnné `deposit_status` je rovna čtyřem, vyšle se automatu mechanismů událost `EV_FREE_SPACE` a vyvolá se přechod do centrálního stavu hlavního automatu, do stavu `decide_what_next`.

5.2 FSM pro obsluhu nakládacích mechanismů

Úkolem stavového automatu pro obsluhu nakládacích mechanismů robotu (dále *act fsm*) je řídit sekvenční proces nakládání a vykládání sloupcových herních elementů (puků) a umožnit tak „klientovi“ (tedy hlavnímu stavovému automatu) řídit nakládání a vykládání puků pouze zasíláním povelů na úrovni *nalož puk* nebo *připrav se na vyložení nákladu do úrovně druhého patra*.

V následující části je popsána mechanika robotu, aby bylo zřejmé, jak mechanismy vypadají a jak vypadá proces, který má *act fsm* řídit. V části 5.2.2 je popsáno rozhraní, jehož prostřednictvím automat mechanismů komunikuje s okolím, a v části 5.2.3 je popsán samotný stavový automat *act fsm* ve verzi použité v soutěžní variantě. Spolu s homologační verzí hlavního stavového automatu byla použita verze automatu *act fsm* obsahující jednoúčelovou funkcionalitu popsanou v části 5.2.4.

5.2.1 Mechanické uspořádání a popis činnosti

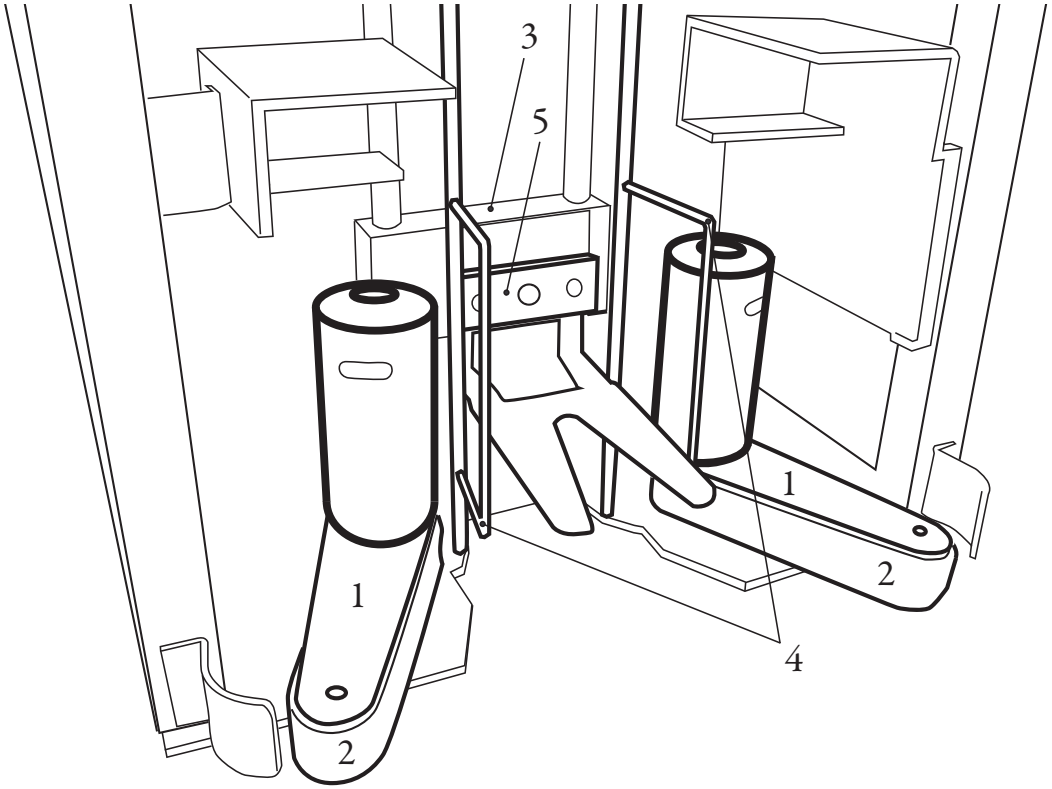
Uspořádání mechanismů pro manipulaci s puky je znázorněno na obr. 5.6. Jsou tvořeny párem *čelistí* (1), které se otáčejí okolo svislé osy a stisknou puk ležící před robotem na zemi a svými *pásy* (2) ho natáhnou na *lžíci* připevněnou k *výtahu* (3), který lžící pohybuje nahoru a dolů. Zadní část lžíce obsahuje otvor, kterým při dolní poloze výtahu „vykukuje“ ramínko koncového spínače použitého k detekci přítomnosti puku při jeho „nasávání“. Původně byl na místě tohoto mikrospínače infračervený dálkoměrný senzor Sharp, který umožňoval navíc rozpoznávat puky blízko před robotem (a proto bylo možné vytvářet pro *main fsm* událost `EV_PUCK_REACHABLE` zmiňovanou v sekci 5.1.1). Nepřesné řízení polohy výtahu a s ním spojené problémy s odrazy paprsku od lžíce nás ale donutily použít mechanické řešení (mikrospínač).

Důležitou součástí mechanismů je držák puků (4) tvořený dvojicí drátěných rámečků přiletovaných ke svým osičkám, okolo kterých se rámečky otáčí. Tento „párový orgán“ byl pojmenován jako *dvířka*. Naložené puky jsou poté, co je výtah vyveze do potřebné výšky, sevřeny ve dvířkách. Po otevření dvířek puky spadnou na lžíci. Posledním pohyblivým elementem je *vytlačovač* (5), který se účastní procesu vykládání puků. Je umístěn na výtahu spolu se lžící (a za ní) a pohybuje se vertikálně spolu s ní, jeho horizontální pohyb (vytlačování) je na lžíci nezávislý. Je tvořen kvádříkem, který doléhá na nejnižší puk naložený na lžíci a při vykládání věže puků vytlačuje (vysouvá) věž po lžíci ven z robotu. Pohyby mechanismů jsou shrnuty v tab. 5.3.

Při jízdě robotu po hřišti je vhodné, aby byla lžíce mírně zvednuta nad zem, aby se zabránilo jejímu zaseknutí o nerovnosti hřiště (*transportní poloha výtahu*). Před nakládáním je naopak třeba lžíci položit na zem, aby bylo možné na ni nasunout puk (*dolní poloha výtahu*). V následujících bodech je stručně popsán postup nakládání puků a vykládání naložené věže puků z robotu. Vykládání věže je parametrizováno výškou („patrem“), na které má být náklad vyložen.

Popis procedury nakládání puků

1. **Příprava před samotným nakládáním:** přesun „výtahu“ (nakládací lžíce) z transportní polohy do dolní polohy.



Obrázek 5.6: Uspořádání mechanismů, popisky podle tab. 5.3

Číslo	Název části	Pohyb	Typ pohonu
1.	čelisti	sevření, rozevření	serva
2.	pásy na čelistech	dovnitř, ven	motory
3.	výtah (pohyb lžíce)	nahoru, dolu	motor
4.	drátěná dvířka	otevřít, zavřít	servo
5.	vytlačovač puků	vysunout, zasunout	motor

Tabulka 5.3: Přehled pohybů jednotlivých částí mechanismů

2. **Naložení puku:** čelisti se sevřou a pohyb pásů na čelistech vtáhne puk na nakládací lžíci robotu. Úspěšné vtažení sloupcového herního elementu (puku) do robotu je zjištěno mikrospínačem.[8]

3. **„Uskladnění“ puku:** nakládá-li se již čtvrtý[9] puk, spočívá jeho uskladnění v přesunutí výtahu do transportní polohy a sevření čelistí (aby puk při jízdě nevypadl). V případě nakládání prvních tří puků jsou dvířka otevřena (dříve uskladněné puky spadnou na puk právě nakládaný), výtah nakládaný puk (včetně těch případně již naložených) vyzdvihne do potřebné výšky (nad dolní drátěnou hranu držáku puků dvířek), dvířka se zavřou, čímž se puky zafixují, a nakonec se výtah přesune do transportní polohy.

Popis procedury vykládání puků

1. **Příprava výtahu:** V případě že nejsou naloženy více než tři puky, vyjede nejprve výtah pod úroveň dvířek, aby po jejich otevření nepadaly puky na lžíci ze zbytečně velké výšky. V případě že je robot plně naložen, se tento krok neprovádí.

2. **Otevření drátěného držáku puků (otevření dvířek):** puky spadnou na lžíci, a jsou tak připravené k vyložení.

3. **Přesun výtahu do správné polohy:** výtah (a tedy lžíce s puky) vyjede do takové výšky, aby byla lžíce v bezpečné vzdálenosti nad úrovní, do které se budou puky vykládat (přízemí, první, nebo druhé patro), a nemohla tak zavadit o vykládací plochu. (Výška patra odpovídá výšce puku, tedy 30 mm.)

4. **Vyložení nákladu:** poté co se robot přiblíží k místu, na kterém má vyložit svůj náklad tak, že přední hrana lžíce je nad plochou určenou k vyložení puků, sjede výtah o trochu níže, aby se lžíce dotýkala plochy, na kterou budou puky vyloženy, a vytlačovač vytlačí puky ven z robotu (ze lžíce). Vytlačovač v krajní poloze vysune puky tak, že jsou z větší části na cílové ploše, ovšem z části jsou ještě opřené o lžíci. Robot tedy musí následně vycouvat.

5. **Dokončení procesu a přesun výtahu do transportní polohy:** poté co lžíce opustí prostor nad cílovou (pravděpodobně vyvýšenou) plochou, dostane stavový automat signál a výtah se přesune do základní polohy (což při vykládání do jiného patra, než do „přízemí“ znamená, že sjede dolů).

5.2.2 Události přijímané a generované automatem a jeho vazby na okolí

Jak již bylo naznačeno, automat mechanismů je řízen hlavním stavovým automatem. Spolupráce mezi nimi je založená na výměně zpráv (událostí). Automat mechanismů po většinu času setrvává v „ústředním“ stavu `wait_for_command` a čeká na příchod jedné z událostí

[8] Původním záměrem bylo využití infračerveného dálkoměrného senzoru. Pro problémy s odrazy paprsku od hliníkových dílů a zejména pro problémy s přesností regulace polohy výtahu jsme ale od jeho využití byli nuceni upustit.

[9] Čtvrtý puk je pukem posledním, v robotu je místo pouze pro čtyři puky, tři upevněné v drátěném držáku (dvířkách) a jeden položený na lžíci.

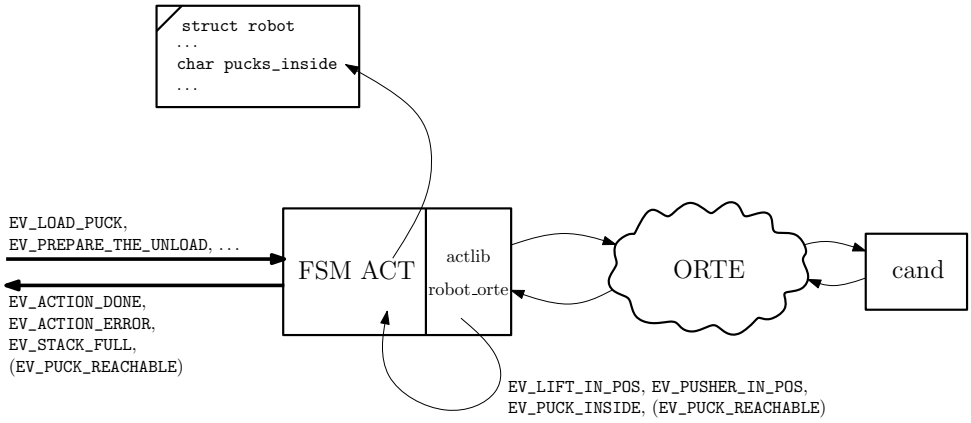
EV_SCRABBLE, EV_LOAD_THE_PUCK resp. EV_PREPARE_THE_UNLOAD, které znamenají po pořádku požadavek na „hrábnutí chapadly“, naložení puku, resp. přípravu na vyložení puků v zadaném patře (událost EV_PREPARE_THE_UNLOAD s sebou nese informaci o čísle patra, do kterého se má náklad vyložit).

Přehled událostí, na něž *act fsm* reaguje, je vypsán v tab. 5.4. Jako zpětnou vazbu *act fsm* hlavnímu stavovému automatu posílá v případě úspěchu událost EV_ACTION_DONE a v případě chyby (např. pokud se nepodařilo puk „nasát“ na lžíci) událost EV_ACTION_ERROR (tab. 5.5).

Komunikace *act fsm* s řízenými mechanismy probíhá prostřednictvím ORTE a fyzicky dále po sběrnici CAN. Vzájemný převod zpráv ORTE a zpráv na sběrnici CAN obstarává program *cand* (CAN daemon). Schematicky je to znázorněno na obr. 5.7. Napojení celého řídicího programu robotu (jehož jsou stavové automaty součástí) na komunikační middleware ORTE obstarává modul robot_orte.[10]

Automat mechanismů prostřednictvím knihovny *actlib* určuje, jaké zprávy pro jednotlivé aktuátory má ORTE publikovat. Serva a motory pásů jsou takto ovládány „v otevřené smyčce“. Při „nasávání“ puků automat potřebuje zpětnou vazbu od koncového spínače a při řízení výtahu a vytlačovače využije informaci o tom, že motor dojel na žádanou polohu. Zprávy o těchto okolnostech přijímá modul robot_orte a vytváří na jejich základě pro *act fsm* příslušné události. Konkrétně události EV_PUCK_INSIDE, EV_LIFT_IN_POS a EV_PUSHER_IN_POS.

Posledním prostředkem komunikace stavového automatu *act fsm* s okolním světem je globální proměnná uložená v centrální struktuře robot, uchovávající počet puků naložených do robotu. Hodnotu robot.pucks_inside aktualizuje pouze *act fsm*. Čte ji ovšem i hlavní stavový automat a displej.



Obrázek 5.7: Spolupráce *act fsm* s okolím

[10]Modul ve smyslu komponenty programu napsaného v jazyce C. Zdrojový kód modulu robot_orte je v souboru robot_orte.c a rozhraní modulu v souboru robot_orte.h.

č.	Název události	Původce	Význam
1.	`EV_SCRABBLE`	*fsm main*	Hlavní automat signalizuje, že je třeba „zašátrat“ chapadly, *act fsm* po příchodu tohoto signálu sjede výtahem do dolní polohy a zmenší úhel sevření chapadel, čímž učiní pokus o nahrnutí případného nepřesně umístěného puku před lžíci robotu, poté čeká na příchod události `EV_LOAD_THE_PUCK`.
2.	`EV_LOAD_THE_PUCK`	*fsm main*	Signál pro *act fsm*, že má naložit puk.
3.	`EV_PREPARE_THE_UNLOAD`	*fsm main*	Signalizuje automatu *act fsm*, že se má připravit pro vykládání naložených puků do patra daného hodnotou předanou spolu s událostí; automat umístí naložené puky na lžíci a vyjede s ní kousek nad požadované patro, poté čeká na příchod události `EV_UNLOAD_PUCKS`.
4.	`EV_UNLOAD_PUCKS`	*fsm main*	Pokyn pro dokončení vykládací procedury, po příchodu události výtah sjede o něco níže, aby dolehl na plochu, na níž bude náklad vykládán a vytlačí puky ven z robotu.
5.	`EV_FREE_SPACE`	*fsm main*	Dává najevo, že pod lžící již není překážka a je možné výtah přesunout do transportní polohy.
6.	`EV_PUCK_INSIDE`	ORTE	Puk byl detekován koncovým spínačem (pásům se podařilo ho nasunout na lžíci).
7.	`EV_LIFT_IN_POS`	ORTE	Signál od řídicí elektroniky, že výtah dojel na požadovanou polohu.
8.	`EV_PUSHER_IN_POS`	ORTE	Vytlačovač puků dojel na požadovanou polohu.

Tabulka 5.4: Přehled událostí přijímaných v různých stavech automatem *act fsm*; jejich původce z hlediska FSM a jejich význam

Název události	Význam
`EV_ACTION_DONE`	Akce vykonána v pořádku.
`EV_ACTION_ERROR`	Při pokusu o vykonání akce došlo k chybě.

Tabulka 5.5: Přehled událostí, které posílá *act fsm* hlavnímu stavovému automatu

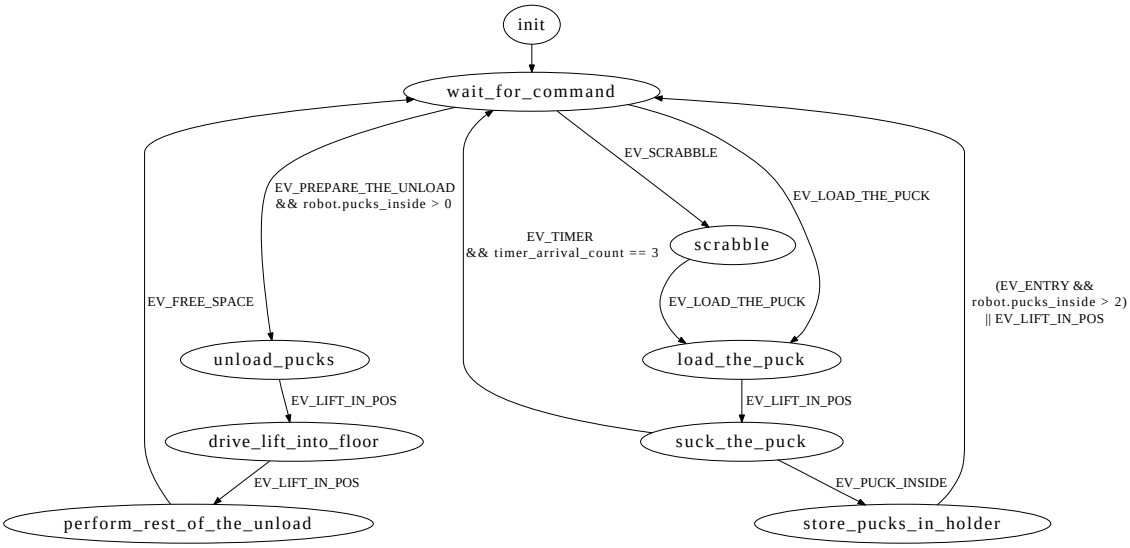
Název proměnné	Umístění	Význam
`robot.pucks_inside`	struktura `robot`	Počet puků naložených do robotu (0–4).
`floor_to_unload`	globální proměnná na úrovni *act fsm*	Číslo patra, do kterého se mají vyložit naložené puky; hodnota je nastavována v ústředním stavu `wait_for_command` po příchodu události `EV_PREPARE_THE_UNLOAD`.

Tabulka 5.6: Globální proměnné, s nimiž *act fsm* pracuje

5.2.3 Popis samotného stavového automatu

Počátečním stavem navrženého stavového automatu *act fsm* je (pseudo)stav `init`, ve kterém je pouze nastavena hodnota počtu uložených puků na nulu (`robot.pucks_inside`), a ihned je vyvolán přechod do „ústředního“ stavu `wait_for_command`.

V ústředním stavu `wait_for_command` tráví *act fsm* většinu času. Při vstupu do tohoto stavu voláním příslušných funkcí z knihovny *actlib* zajistí, že jsou po ORTE publikovány „transportní“ polohy všech mechanismů (vytlačovač zasunut, výtah v transportní poloze, pásy zastaveny, dvířka zavřena). Na základě počtu naložených puků je nastavena volná (otevřená) poloha chapadel, jsou-li naloženy méně než čtyři puky. Je-li již robot plně naložen všemi čtyřmi puky, jsou chapadla stisknuta, aby sevřela čtvrtý naložený puk umístěný na lžíci a zabránila tak, aby za jízdy vypadl. Ve stavu `wait_for_command` se čeká na příchod jedné z událostí `EV_SCRABBLE`, `EV_LOAD_THE_PUCK` nebo `EV_PREPARE_THE_UNLOAD`.



Obrázek 5.8: Stavy stavového automatu *act fsm* použité při nakládání puků a vykládání věží

Zbylé stavy automatu *act fsm* patří do jedné ze dvou větví. První z nich je určena pro nakládání puku a druhá pro vykládání věže. První odpovídá popisu procedury nakládání

puků z části 5.2.1 a druhá odpovídá sekvenci vykládací, popsané tamtéž. Stavový diagram automatu *act fsm* je znázorněn na obr. 5.8.

Nakládání puků Před pokusem o naložení puku může být vhodné „hrábnout“ chapadly a potom robotem o kousek popojet dopředu. Požadavek na „zahrabání“ dává hlavní automat najevo zasláním události `EV_SCRABBLE`. Automat *act fsm* ve stavu `wait_for_command` reaguje na příchod této události přechodem do stavu `scrabble`. V tomto stavu se přesune výtah do dolní polohy a více se sevřou klepeta, aby případně nahrnula puk na lžíci. Poté se čeká na příchod události `EV_LOAD_THE_PUCK`, zaslané hlavním automatem, která vyvolá přechod do stavu `load_the_puck`.

Do stavu `load_the_puck` mohl automat přejít buď ze stavu `scrabble`, nebo přímo ze stavu `wait_for_command`. Pro pořádek se při přechodu do tohoto stavu nastaví definované polohy vytlačovače (zasunut), klepet (otevřeny) a rychlosti pásů (vypnuty). Požadovaná poloha výtahu se nastaví na nejnižší. Automat v tomto stavu čeká na událost `EV_LIFT_IN_POS`, po jejímž příchodu je vyvolán přechod do následujícího stavu, jehož definice následuje (`suck_the_puck`):

```
FSM_STATE(suck_the_puck)
{
  static int timer_arrival_count;
  switch (FSM_EVENT) {
  case EV_ENTRY:  // prechod (vstup) do stavu
    // we expect lift is on the ground
    timer_arrival_count = 0;
    printf("suck_the_puck state entered\n");
    FSM_TIMER(2000);            // nastaveni casovace
    act_chelae(CHELA_TIGHT, CHELA_TIGHT); // celisti
    act_belts(BELTS_IN, BELTS_IN);        // pasy
    break;
  case EV_PUCK_INSIDE: // udalost od koncoveho spinace
    FSM_TRANSITION(store_pucks_in_holder);
    break;
  case EV_TIMER:       // vyprseni casovace
    timer_arrival_count++;
    switch (timer_arrival_count) {
    case 1:
      act_belts(BELTS_OUT, BELTS_OUT); // pasy
      FSM_TIMER(600);  // nastaveni casovace
      break;
    case 2:
      act_belts(BELTS_IN, BELTS_IN);   // pasy
      FSM_TIMER(2000);
      break;
    case 3:
      FSM_SIGNAL(MAIN, EV_ACTION_ERROR, NULL);
      FSM_TRANSITION(wait_for_command);
```

```
      break;
    }
    break;
  case EV_RETURN:
  case EV_LOAD_THE_PUCK:
  case EV_LIFT_IN_POS:
  case EV_PUSHER_IN_POS:
  case EV_PREPARE_THE_UNLOAD:
  case EV_UNLOAD_PUCKS:
  case EV_FREE_SPACE:
  case EV_SCRABBLE:
    DBG_PRINT_EVENT("unhandled event");
    break;
  case EV_EXIT: // opusteni stavu
    act_belts(BELTS_OFF, BELTS_OFF);     // pasy
    act_chelae(CHELA_OPEN, CHELA_OPEN); // celisti
    break;
  }
}
```

Tímto dalším stavem je `suck_the_puck`. Úkolem automatu v tomto stavu je zajistit nasátí puku do robotu pomocí pásů. Opět se jedná o jakýsi pseudostav, ve kterém je nejméně jednou volán časovač použitím makra `FSM_TIMER()` a ve kterém je použita lokální celočíselná proměnná `timer_arrival_count`, počítající, kolikrát přišla událost od časovače. Při přechodu do tohoto stavu (tj. při příchodu události `EV_ENTRY`) je hodnota proměnné `timer_arrival_count` nastavena na `0`, je požadováno sevření čelistí a pohyb pásů ve směru do robotu a je nastaven časovač (2 sekundy).

Automat ve stavu `suck_the_puck` čeká na příchod události `EV_PUCK_INSIDE` signalizující úspěšné nasátí puku do útrob robotu. Pokud se pokus o nasátí puku obejde bez komplikací, tato událost přijde a je vyvolán přechod do posledního „nakládacího“ stavu, jímž je `store_pucks_in_holder`. Pokud událost během dvou sekund nedorazí, vypršení časovače vytvoří událost `EV_TIMER`. Obsluha této události spočívá:

1. ve zvýšení hodnoty proměnné `timer_arrival_count`,
2. v provedení následujícího v závislosti na její nové hodnotě.
   - Pokud časovač vypršel poprvé, je nastaven pohyb pásů směrem ven z robotu a znovu odstartován časovač.
   - Jestliže časovač vypršel podruhé, je iniciován pohyb pásů směrem do robotu a opět odstartován časovač.
   - Vypršel-li časovač potřetí, znamená to, že se puk ani na druhý pokus nepodařilo úspěšně nasát a *act fsm* zastavuje pásy, posílá hlavnímu stavovému automatu signál `EV_ACTION_ERROR` a přechází do ústředního stavu `wait_for_command`.

Ve stavu `store_pucks_in_holder` se dokončuje nakládání puků. Jsou-li již naloženy tři puky (čtvrtý se do držáku/dvířek nevejde), je pouze inkrementována hodnota globální proměnné `robot.pucks_inside`, hlavnímu FSM je vyslána událost `EV_ACTION_DONE` a je iniciován přechod do „hlavního“ stavu `wait_for_command`. Naložený puk je uskladněn na lžíci v transportní poloze.

Je-li nakládaný puk prvním, druhým, nebo třetím pukem v robotu, jsou při přechodu do stavu `store_pucks_in_holder` otevřena dvířka (případné již naložené puky spadnou na puk nakládaný) a je vyslán povel pro přesun výtahu do úrovně kousek nad hranu dolní části dvířek. Poté automat v tomto stavu čeká na příchod události `EV_LIFT_IN_POS`.

Příchod události `EV_LIFT_IN_POS` znamená, že je možné „uchopit puky držákem“, tedy vydat povel k uzavření „dvířek“, zvýšit hodnotu `robot.pucks_inside`, vyslat automatu *main fsm* signál `EV_ACTION_DONE` a přejít do stavu `wait_for_command`.

Vykládání puků je implementováno ve třech stavech pojmenovaných `unload_pucks`, `drive_lift_into_floor` a `perform_rest_of_the_unload`. Komunikace mezi hlavním automatem a automatem mechanismů je při vykládání nákladu čilejší. Oba automaty musí spolupracovat, protože (jak bylo popsáno v oddíle o mechanice 5.2.1) robot musí při této akci popojíždět.

Vykládací sekvence je započata přesunem z centrálního stavu `wait_for_command` do stavu `unload_pucks` na základě příchodu události `EV_PREPARE_THE_UNLOAD`. Při příchodu této události ve stavu `wait_for_command` je ovšem nejprve zkontrolován počet naložených puků (`robot.pucks_inside`). Je-li nulový, je hlavnímu automatu poslána „chybová“ událost `EV_ACTION_ERROR` a *act fsm* zůstává ve stavu `wait_for_command`. V opačném případě je hodnota získaná spolu s událostí `EV_PREPARE_THE_UNLOAD` přetypována a uložena do proměnné `floor_to_unload` a je vyvolán přechod do stavu `unload_pucks`.

Při vstupu do stavu `unload_pucks` je nejprve vydán pokyn pro úplné otevření chapadel (v případě, že byly naloženy všechny čtyři puky, byla chapadla dosud sevřena). Pokud jsou naloženy všechny čtyři puky, je rovnou vyvolán přechod do stavu `drive_lift_into_floor`. Jsou-li naloženy méně než čtyři puky, pošle automat výtah (lžíci) pod hranu držáku puků. V popisovaném stavu potom čeká na událost `EV_LIFT_IN_POS`, jejíž příchod ošetřuje přechodem do stavu `drive_lift_into_floor`.

Ve stavu `drive_lift_into_floor` je výtah poslán do výšky odvozené z hodnoty proměnné `floor_to_unload`. Dokončení tohoto pohybu je signalizováno příchodem události `EV_LIFT_IN_POS`. Reakcí na její příchod je vyslání zprávy `EV_ACTION_DONE` hlavnímu automatu (čímž *act fsm* sděluje, že lžíce naložená puky je připravená ve výšce nad požadovaným patrem) a přechod do stavu `perform_rest_of_the_unload`.

V posledním „vykládacím“ stavu automat čeká na událost `EV_UNLOAD_PUCKS` od hlavního FSM. Její příchod znamená, že může dokončit vykládání, a první reakcí je vyslání výtahu do poněkud nižší polohy, ve které se lžíce dotkne vykládané plochy. Ukončení tohoto přesunu signalizuje událost `EV_LIFT_IN_POS`. Po jejím příchodu následuje úplné vysunutí vytlačovače (spolu s pokynem k tomuto pohybu je vynulována hodnota `robot.pucks_inside`). Dokončení pohybu vytlačovače signalizuje příchod události `EV_PUSHER_IN_POS`. Poté co automat obdrží tuto zprávu, vyšle událost `EV_ACTION_DONE` pro hlavní FSM. Nato stále zůstává ve stavu `perform_rest_of_the_unload` a čeká na příchod události `EV_FREE_SPACE`

od hlavního automatu, signalizující, že pod lžící je již volno (robot vycouval od akropole) a automat může vyvolat přechod do stavu `wait_for_command`.

Příchod do stavu `wait_for_command` znamená zároveň přesun všech mechanismů do transportní polohy.

5.2.4 Homologační stavy automatu *act fsm*

Homologační verze hlavního stavového automatu využívá modifikované verze automatu *act fsm*, který v centrálním stavu `wait_for_command` reaguje na událost `EV_GRASP_THE_PUCK` a pro účely homologace definuje dva stavy: `grasp_the_puck` a `hold_the_puck`.

V tomto modifikovaném *act fsm* způsobí ve stavu `wait_for_command` příchod události `EV_GRASP_THE_PUCK` přechod do stavu `grasp_the_puck` pro „uchopení“ puku. Při vstupu do stavu `grasp_the_puck` se sevřou chapadla, pásy se roztočí směrem do robota a nastaví se časovač. Nepočítá se s událostí od koncového mikrospínače detekujícího puk, ale čeká se na událost `EV_TIMER` od časovače. V okamžiku, kdy tato událost přijde, zastaví se pásy, hlavnímu FSM se předá událost `EV_ACTION_DONE` a vyvolá se přechod do stavu `hold_the_puck`.

Ve stavu `hold_the_puck` se čeká na příchod události `EV_UNLOAD_PUCKS` od hlavního stavového automatu. Po jejím příchodu je puk „vyplivnut“: pásy se roztočí směrem ven a nastaví se časovač. Po vypršení časovače příjde událost `EV_TIMER`, na kterou *act fsm* ve stavu `hold_the_puck` reaguje zastavením pohybu pásů a otevřením klepet. Zároveň pošle automatu *main fsm* signál `EV_ACTION_DONE` a vyvolá přechod do stavu `wait_for_command`.

Kapitola 6

Závěr

Cílem práce byla implementace stavových automatů pro robotickou soutěž Eurobot 2009. V této práci jsem popsal stavový automat pro řízení herní strategie (hlavní stavový automat) a rovněž jemu podřízený stavový automat pro ovládání mechanismů robotu, použitých pro nakládání a vykládání herních prvků – dřevěných puků. Rovněž jsem se pokusil objasnit způsob spolupráce těchto automatů s ostatním softwarem robotu.

Práce na stavových automatech před samotnou soutěží byla poněkud nepříznivě ovlivněna stavem mechanické stránky robotu a postupem prací na mechanismech před soutěží. Byly tu problémy s přesností řízení polohy výtahu (bohužel byl použit jednokanálový IRC senzor) a se senzorem pro detekci puků, které nás stály velké množství času.

Automat pro sbírání a vykládání herních elementů se v těch chvílích, kdy byla mechanika robotu provozuschopná, osvědčil. Ohodnotit vytvořený stavový automat pro řízení strategie je složitější. Jistě by bylo možné realizovanou strategii rozšířit. Před soutěží se to stihnout nepodařilo. Ani po soutěži pro to nebyly příznivé podmínky, protože mechanika dobře nefungovala. Vyvíjet a ladit stavové automaty pro robot našeho týmu bez robotu samotného je teoreticky možné, ale je to složitější.

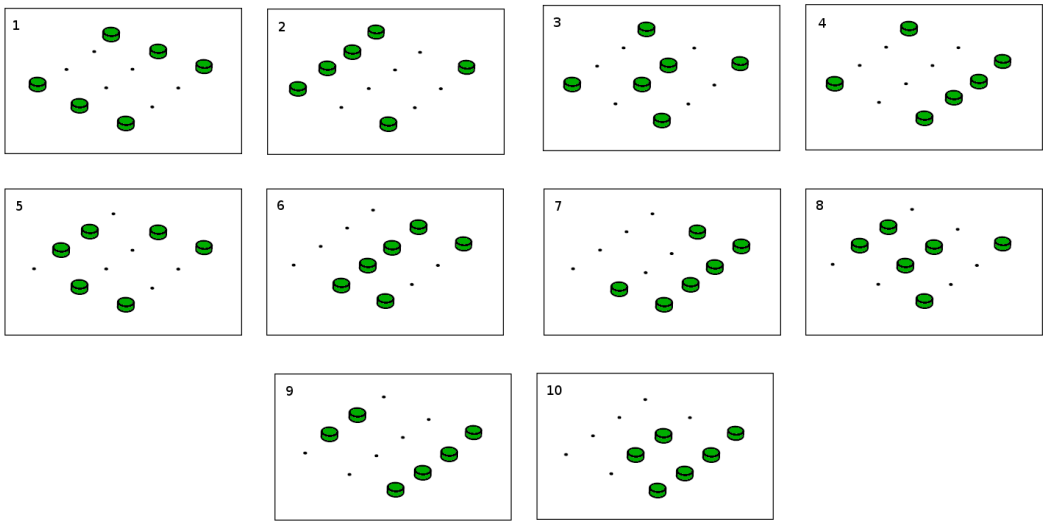
Pořadí a způsob sbírání puků a jejich vykládání na akropoli pro jednotlivé konfigurace volných puků je definován ve dvou konstantních polích. Ne pro všechny konfigurace se podařilo hodnoty v těchto polích dostatečně otestovat a odladit.

V případě, že by se podařilo použít rangefinder Hokuyo k detekci věží postavených protivníkem, bylo by vhodné hlavní stavový automat rozšířit o schopnost poradit si se situací, kdy na jím zvoleném místě již stojí věž protivníka. Další možnost vylepšení (a zrychlení hry) by se nabízela v rozsáhlejším využití schopnosti stavových automatů (hlavního automatu a automatu mechanismů) fungovat paralelně; vzhledem k sekvenční povaze mechanismů použitých v letošním ročníku by ale zrychlení práce robotu nebylo zásadní.

Zajímavým rozšířením této práce by mohl být návrh a implementace nástroje pro automatické generování diagramů stavových automatů podobných těm, které jsou uvedeny v kapitole 5, na základě zdrojového kódu. Autorovi stavového automatu napsaného s užitím knihovny *FSM* by takový nástroj usnadnil hledání chyb. Všem programátorům, kteří s hotovým stavovým automatem musí pracovat by velmi usnadnil orientaci v programu.

Dodatek A

Konfigurace volných puků



Obrázek A.1: Kombinace volných puků, jak jsou uvedeny v Pravidlech [1]; detail puků zeleného hráče (směr pohledu jako na obr. 2.1)

Dodatek B

Obsah přiloženého CD

`/pdf`	Elektronická podoba této práce ve formátu PDF.
`/doc`	Použitá nepublikovaná literatura.
`/src`	Zdrojové kódy části řídicího programu robota. Stavové automaty popsané v této práci jsou k nalezení v souborech `competition.cc` (soutěžní verze hlavního stavového automatu), `homologation.cc` (homologační verze) a `fsmact.c` (stavový automat řídící mechanismy). Definice událostí pro tyto automaty potom v souboru `roboevent.py`.
`/pics`	Fotografie robota.
`/video`	Video (záznam jízdy robotu a nabírání puků).

Literatura

[1] Eurobot 2009 – Chrámy Atlantidy – Pravidla. Na webu, 2008, http://www.eurobot.cz/2009/E2009_Rules-CZ-v1.pdf.

[2] Robotický tým CTU Dragons: Eurobot Software Documentation. Dokumentace na webu generovaná ze zdrojových kódů. http://rtime.felk.cvut.cz/dragons/doc/.

[3] Smolík, P.; Píša, P.: ORTE: The Open Real Time Ethernet. Na webu, Czech Technical University in Prague, Department of Control Engineering http://smoliku.cz/orte/lib/exe/fetch.php?media=wiki:rtn08_orte.pdf.

[4] Sojka, M.: Mobilní robot pro soutěž Eurobot. Na webu, 2009, prezentace pro Kolokvium CAK. http://rtime.felk.cvut.cz/kolokvium/2009/program.php.