Fakulta elektrotechnická
Katedra aplikované elektroniky a telekomunikací

Diplomová práce

Softwarové vybavení autonomního prostředku

Autor práce: Bc. Filip Zvonař
Vedoucí práce: Ing. Kamil Kosturik, Ph.D.

Plzeň 2020

ZÁPADOČESKÁ UNIVERZITA V PLZNI
Fakulta elektrotechnická
Akademický rok: 2019/2020

ZADÁNÍ DIPLOMOVÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Filip ZVONAŘ**
Osobní číslo:	**E18N0033P**
Studijní program:	**N2612 Elektrotechnika a informatika**
Studijní obor:	**Elektronika a aplikovaná informatika**
Téma práce:	**Softwarové vybavení autonomního prostředku**
Zadávající katedra:	**Katedra aplikované elektroniky a telekomunikací**

Zásady pro vypracování

Vytvořte řídicí aplikaci pro autonomní pásovou platformu.

1. Zanalyzujte současný stav pásové platformy. Zaměřte se zejména na softwarové vybavení.
2. Navrhněte úpravy softwaru rozšiřující současné možnosti autonomního pohybu. Zaměřte se zejména na zpracování informací z obrazových senzorů.
3. Navržené řešení realizujte. K programování používejte programovací jazyk C/C#.

Rozsah diplomové práce:	**40 – 60 stran**
Rozsah grafických prací:	**podle doporučení vedoucího**
Forma zpracování diplomové práce:	**tištěná/elektronická**

Seznam doporučené literatury:

Jürgen Bayer: C# 2005. Computer Press, ISBN: 978-80-251-1620-3

Vedoucí diplomové práce:	**Ing. Kamil Kosturik, Ph.D.** Katedra aplikované elektroniky a telekomunikací
Datum zadání diplomové práce:	**4. října 2019**
Termín odevzdání diplomové práce:	**28. května 2020**

Západočeská univerzita v Plzni
Fakulta elektrotechnická

L.S.

Prof. Ing. Zdeněk Peroutka, Ph.D.
děkan

Doc. Dr. Ing. Vjačeslav Georgiev
vedoucí katedry

V Plzni dne 4. října 2019

Abstrakt

Tato diplomová práce se zabývá vývojem hardware a nového software pro řídící počítač autonomní pásové platformy, která se nachází na Fakultě elektrotechnické Západočeské Univerzity v Plzni. V první části této práce je rozebírán její současný stav a úpravy periferií systému jako nové senzory otáček, úpravy firmware v jazyce C a regulace otáček motorů. K platformě je navržena a vyrobena pohyblivá věž pro umožnění pohybu senzorem MS Kinect. Ve druhé části práce se věnujeme principům programování v jazyce C# a vývoji řídící aplikace, implementaci sledování postav a řízení pohybu platformy.

Klíčová slova

Kinect, CAN, PID, autonomní řízení

Abstract

Zvonař, Filip. *Software equipment of an autonomous tracked platform* [*Softwarové vybavení autonomního prostředku*]. Pilsen, 2020. Master thesis (in Czech). University of West Bohemia. Faculty of Electrical Engineering. Department of Applied Electronics and Telecommunications. Supervisor: Kamil Kosturik

---

This master thesis is about the design of hardware and new software for a PC controlled autonomous platform which is at the Faculty of Electrical Engineering of the University of West Bohemia. The platform features and changes of its periferies are described, including new speed encoders, firmware updates in language C and the solution of speed regulation. A turret for the mounting of a Kinect sensor is designed and manufactured. In the second part of this thesis, principles of C# programming are described and the development of a control software, the implementation of skeleton tracking and the control of the platform's movement is explained.

Keywords

Kinect, CAN, PID, autonomous control

Prohlášení

Předkládám tímto k posouzení a obhajobě diplomovou práci, zpracovanou na závěr studia na Fakultě elektrotechnické Západočeské univerzity v Plzni.

Prohlašuji, že jsem svou závěrečnou práci vypracoval samostatně pod vedením vedoucího diplomové práce a s použitím odborné literatury a dalších informačních zdrojů, které jsou všechny citovány v práci a uvedeny v seznamu literatury na konci práce. Jako autor uvedené diplomové práce dále prohlašuji, že v souvislosti s vytvořením této závěrečné práce jsem neporušil autorská práva třetích osob, zejména jsem nezasáhl nedovoleným způsobem do cizích autorských práv osobnostních a jsem si plně vědom následků porušení ustanovení § 11 a následujících autorského zákona č. 121/2000 Sb., včetně možných trestněprávních důsledků vyplývajících z ustanovení § 270 trestního zákona č. 40/2009 Sb.

Také prohlašuji, že veškerý software, použitý při řešení této diplomové práce, je legální.

V Plzni dne 19. června 2020

Bc. Filip Zvonař

.....

Podpis

Obsah

Seznam obrázků

Seznam symbolů a zkratek

CAN	Controller Area Network
CIL	Common Intermediate Language
CLR	Common Language Runtime
DC	Direct Current, stejnosměrný
ECMA	European Computer Manufacturers Association, Evropská asociace výrobců počítačů
GPOS	General Purpose Operating System, běžný operační systém
GPS	Global Positioning System
I2C	Inter-Integrated Circuit
IR	Infrared Radiation
LCD	Liquid Crystal Display
LED	Light Emitting Diode, svítivá dioda
MS	Microsoft
MSI	Micro-Star International
OS	Operační systém
PC	Personal Computer, osobní počítač
PHP	Hypertext Preprocessor
PID	Proporcionální, integrační a derivační
PLA	Polylactic Acid
PVC	Polyvinylchlorid
PWM	Pulse Width Modulation, pulzně šířková modulace
RAM	Random Access Memory, paměť s libovolným přístupem
RGB	Red Green Blue, červená-zelená-modrá
RTOS	Real Time Operating System
RTR	Remote Transmission Request, pořadavek o přenost dat
SDK	Software Development Kit
SPI	Serial Peripheral Interface, sériové periferní rozhraní
SSD	Solid-state drive
USB	Universal Serial Bus, univerzální sériová sběrnice
VGA	Video Graphics Array

1

Úvod

Na úvod této práce zmíním motivaci jejího vzniku. Vybírat si tu práci, která člověka baví, je sice už velké klišé, ale to nic nemění na tom, že i mě některé věci fascinují více než jiné. Věřím, že pro většinu lidí je praxe zajímavější než teorie. S tímto přístupem když budeme vybírat mezi vývojem software a vývojem hardware elektroniky, brzy zjistíme, že k praxi má daleko blíže vývoj software. Dnešní elektronická zařízení se už neobejdou bez chytrých aplikací a to nemluvím jen o internetu věcí. Naprostá většina moderní elektroniky je závislá na svém software a při svém vzniku ožívá právě až v okamžiku spuštění své řídící aplikace. Takovému oživení je vždy nádherné přihlížet a stalo se i motivací této práce.

Pásová platforma je na Katedře aplikované elektroniky a telekomunikací na 5. patře Fakulty elektrotechnické ZČU v Plzni již přes 10 let. Jejím základem je asi 75 cm dlouhý podvozek pro model tanku, díky kterému si vysloužila také zažitou přezdívku "tank". Tento tank sice není nijak militantní, ale přesto budí respekt svou zdánlivě složitou elektronickou výbavou ve své otevřené konstrukci pod plexisklem. V jeho jádru se točí ventilátor osobního počítače od kterého vedou sběrnice k množství senzorů a čidel. V jeho zádi je vidět dvojice převodovek se stejnosměrnými motory poháněnými celou platformu.

Její současný stav je výsledkem již mnoha diplomových prací. V současnou podobu má jeho výbava především z prací z roku 2011 [3], kdy byl založen modulární koncept elektroniky platformy s využitím sběrnice CAN, a také z práce z roku 2015 [1], kdy byl připojen chytrý senzor pro rozpoznávání lidské postavy MS Kinect a kdy byly dokončeny firmware jednotek senzorů a první verze řídící aplikace implementující veškerý instalovaný hardware.

Platforma se využívala a nadále využívá při prezentacích univerzity na veřejných akcích i mimo prostory univerzity díky své relativně skladné velikosti. Zároveň je dostatečně prostorově výrazná pro upoutání pozornosti a umožňuje prezentovat práce studentů univerzity dalším potenciálním zájemcům o studium, a to zábavnou formou.

Hlavním účelem této práce je vytvořit novou řídící aplikaci, která umožní platformě spolehlivěji fungovat autonomně a tím lépe uplatnit schopnosti jejího hardware. Tím bude možné lépe prezentovat i výsledky předchozích studentských prací. Díky zařízení MS Kinect, které umožňuje rozpoznávání lidské postavy a gest, může platforma interagovat

nonverbální komunikací i s lidmi, kteří jinak nemají o technologie velký zájem.

V první polovině této práce je popsán celkový stav platformy a hlavně úpravy hardware, které byly nutné pro umožnění použití plného potenciálu senzoru MS Kinect. Hlavními limity pro vývoj responzivní aplikace bylo nevhodné umístění MS Kinect a velmi pomalé reakce regulátoru motorů, která se pohybovaly v řádu vteřin. Pro umožnění vývoje aplikace je tedy první část této práce věnována úpravám firmware a přidanému hardware na jednotce motorů a také otočné věži, která umožňuje směrování kamer nezávisle na podvozku.

V druhé polovině práce se věnujeme vývoji řídící aplikace, který nastíníme úvodem o programování v jazyce C#. Pokračujeme rozborem struktury aplikace a vlastních řídících algoritmů, grafického rozhraní a rozborem vlastního užití aplikace.

2

Popis systému mobilní platformy

Pásová platforma se nachází na katedře aplikované elektroniky a telekomunikací. Svým vzhledem může platforma připomínat tank nebo robota z Číslo pět žije. Jejím základem je pevná ocelová konstrukce v červené barvě s pásovým podvozkem. Podvozek je 75 cm dlouhý a celá platforma váží asi 25 Kg.

Jádrem platformy je počítač s operačním systémem Windows 7, který je uvnitř platformy zabudovaný a na kterém běží jeho řídící Windows aplikace. Platforma je dále osazena několika druhy senzorů, včetně zařízení MS Kinect na vyvýšené věži. Obsahuje GPS a s okolím při provozu komunikuje pomocí reproduktoru, malého LCD displaye nebo přes vzdálenou plochu.

V této kapitole shrneme přehled všech modulů v platformě pro snadné porozumění funkci celého systému. Větší důraz bude kladen na bloky které byly během této práce pozměněny a u ostatních bude více odkazováno na předešlé práce.

Obr. 2.1: Platforma v terénu 2015 [2]

2.1 Blokové funkční schéma

Při základním návrhu platformy byl zvolen modulární přístup ke konstrukci jejího elektronického vybavení. Právě díky tomu je také možné navázat na předchozí práce provedené na platformě.

Pomocí schématu je snadno pochopitelné propojení a návaznost jednotlivých komponent. V centru je vidět počítač, ke kterému jsou po sběrnici USB připojené periferie a další převodníky. Přes převodník na sériový port je připojený modul GPS. Převodník USB/CAN zajišťuje komunikaci s páteřní sběrnicí platformy, na které jsou pak další moduly. Sběrnice CAN umožňuje přístup na sběrnici dle priority zprávy, což umožňuje zajistit určitou bezpečnost systému. Po sběrnici CAN je také možné systém dál rozšiřovat o další moduly s minimálním zásahem do celého systému.

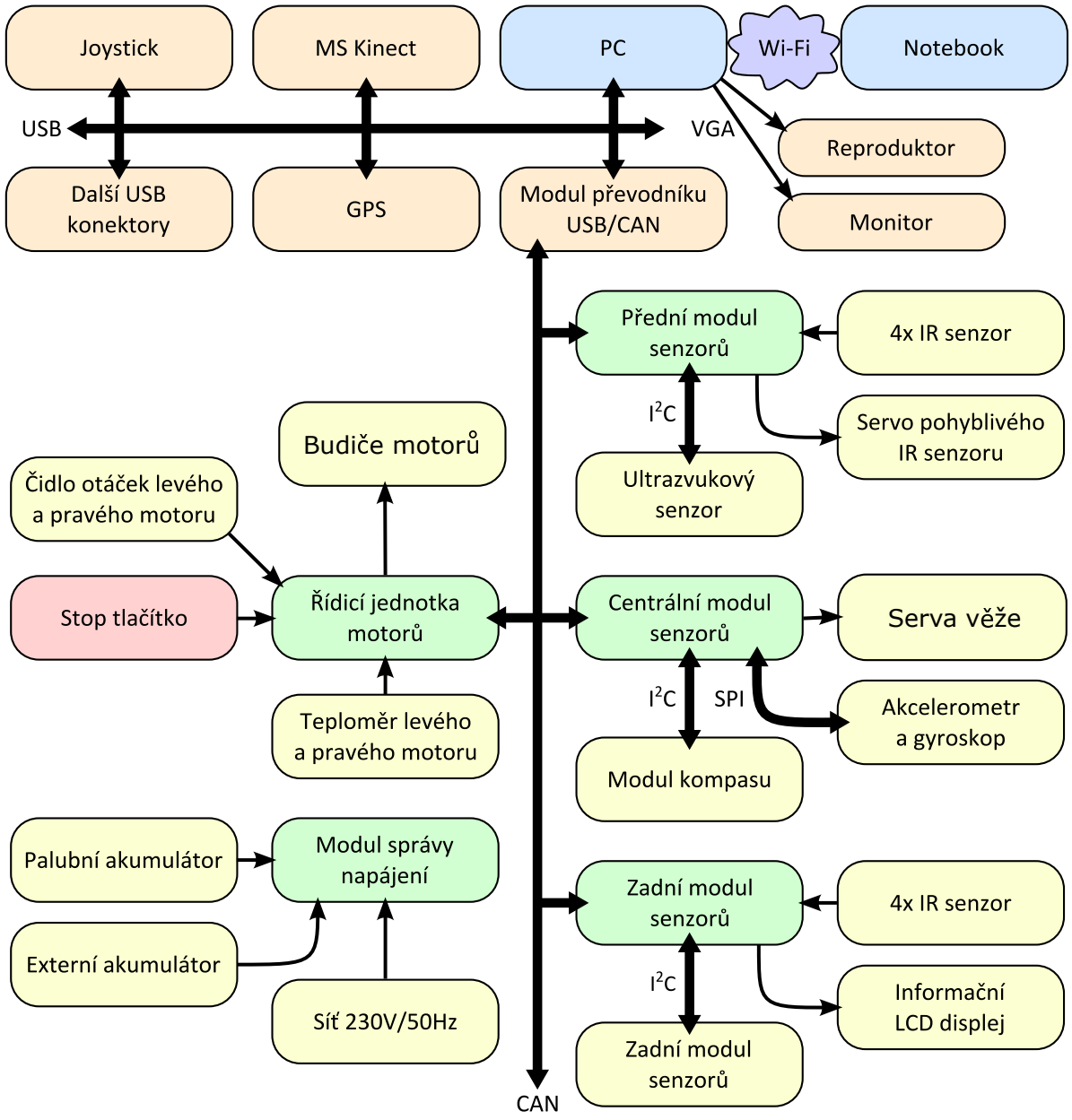
Přímo po USB jsou připojená zařízení, která jsou typická pro použití s počítačem, jako MS Kinect a herní ovladač Gamepad s joysticky. Na bok platformy jsou vyvedeny také dva USB konektory pro připojení dalšího příslušenství.

Na sběrnici CAN jsou připojeny moduly s vlastními mikrokontroléry, na kterých běží real-time aplikace zajišťující regulaci motorů, ovládání serv, obsluhu senzorů a LCD displeje. Se sběrnicí CAN a real-time aplikacemi v modulech je tak možné zajistit vyšší bezpečnost provozu a další rozšiřování systému. Jedním takovým rozšířením do budoucna bude přidání modulu ke správě napájení.

Serva, budiče motorů, budiče laseru a LED světla věže jsou ovládány pomocí modelářského PWM protokolu a připojeny charakteristickými třížilovými plochými kabely. I když mají motory své vlastní budiče s vnitřními regulátory napětí, jsou regulované modulem motorů, který využívá měření otáček přímo na hnací hřídeli pásů. Stop tlačítko znázorněné na schématu je současně implementované přímo na budičích motorů, které již jsou vybavené hardwarovými vypínači.

Ke komunikaci uživatele s platformou je v zadní části platformy na boku malý LCD display, který je součástí zadního modulu senzorů. Platforma je také vybavena reproduktorem a pro hlubší diagnostiku je využíváno připojení na vzdálenou plochu. Pokud není platforma v pohybu, je také možné připojit monitor s klávesnicí a myší přímo přes VGA a USB porty na platformě. V terénu je možné ke komunikaci využít také vizuální komunikaci, jako rozsvícení světel nebo programově nastavený pohyb - gesto. podobně jako platforma je schopna rozpoznávat gesta lidské postavy.

Celá platforma je napájena tříčlánkovou li-ion baterií. Baterie má původní kapacitu 108 Ah a nominální napětí 11,1 V. V nabitém stavu má napětí až 12,6 V. Při vybití na 11 V se automaticky vypíná napájecí zdroj centrálního počítače, ale teoreticky by bylo možné akumulátor vybíjet daleko víc. Li-ion články běžně snesou vybití až na 3 V. To by umožnilo vybíjet baterii i pod 10 V, podle typu článků ze kterých je vyrobena [1]. Při nabíjení z 11,0 V na 12,3 V byla naměřena kapacita přibližně 33 Ah.

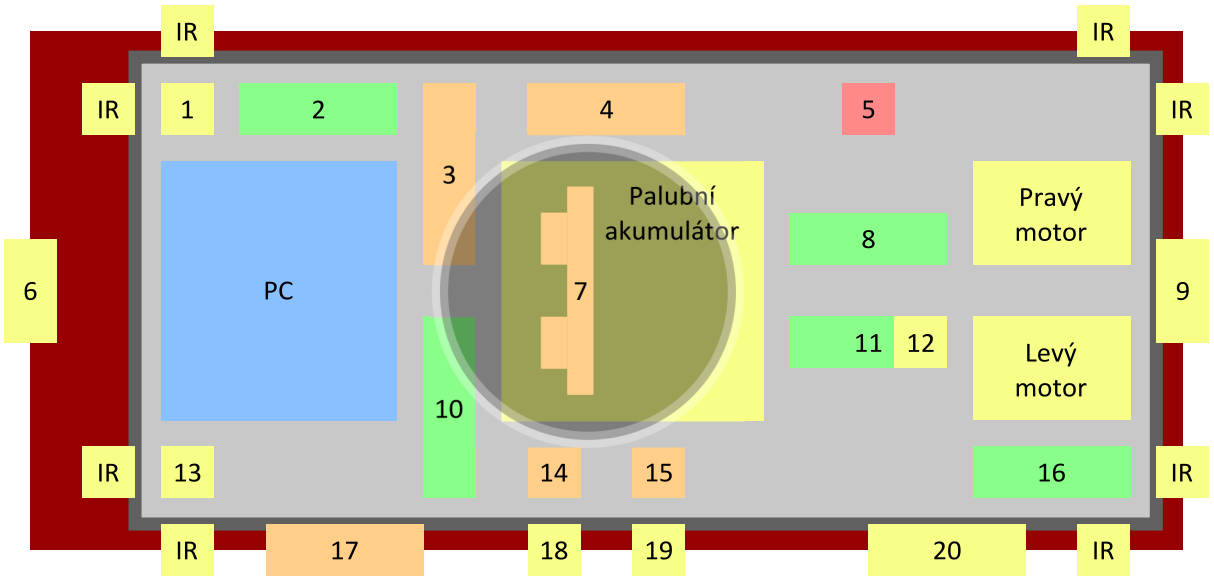


Obr. 2.2: Blokové schéma platformy [1]

2.2 Fyzické rozložení komponent

Fyzické rozložení jednotlivých modulů a komponent je znázorněno schématem 2.3. Barvy jsou použity stejné, jako v blokovém schématu platformy. Otočná věž je znázorněna šedou barvou a fyzicky je možné ji povolením 4 šroubů a odpojením konektorů z platformy sejmout pro transport.

Pod věží se nachází centrální baterie, kolem které se nachází ostatní komponenty. Kabeláž je organizovaná do svazků pomocí spirálových chrániček. Kritická místa pro organizaci kabelů jsou kolem otevřené převodovky platformy a ventilátor počítače.



1. Servo s otočným IR senzorem
2. Přední modul senzorů
3. Převodník USB/CAN
4. GPS modul
5. Stop tlačítko
6. Přední ultrazvukový senzor
7. MS Kinect
8. Řídicí jednotka motorů
9. Zadní ultrazvukový senzor
10. Modul správy napájení
11. Centrální modul senzorů
12. Akcelerometr a gyroskop
13. Kompas
14. Reproduktor
15. Wi-Fi adaptér
16. Zadní modul senzorů
17. Další USB konektory
18. Vypínač
19. Konektor pro síťové napájení
20. LCD displej

Obr. 2.3: Schéma rozložení komponent platformy [1]

2.3 Centrální počítač

Počítač MSI MS-7677 s operačním systémem Windows 7 Professional, na kterém běží řídící aplikace, slouží jako mozek celé platformy. Má dvoujádrový procesor Intel Pentium G630T 2,3 GHz, 4 GB RAM a integrovanou grafickou kartu Intel HD Graphics 2000. Používá 32 GB SSD disk pro dostatečnou odolnost vůči otřesům. Řídící aplikace se spouští automaticky po zapnutí PC spolu s platformou nebo ji je možné spustit ručně po připojení klávesnice, myši a monitoru, nebo pomocí vzdálené plochy.

Ze základové desky je vyveden VGA port do boku platformy a také dva rozšiřující porty USB snadno dostupné zvenčí. Pomocí 3,5 mm jacku je k počítači připojený zabudovaný reproduktor se zesilovačem, napájeným 12 V ze zdroje počítače.

Přímo k základové desce jsou pomocí USB připojena následující zařízení:

- `Wifi modul`
- `GPS modul`
- `Přijímač Gamepad joysticku`

- `Převodník na CAN bus`
- `MS Kinect`

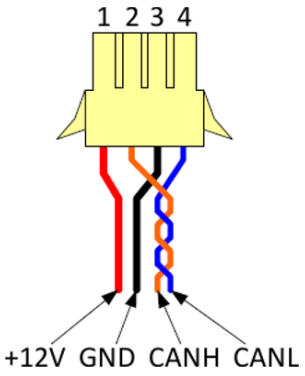
Velkou výhodou počítače přímo v platformě je jeho využití k vývoji řídící aplikace a k programování jednotlivých modulů. Na počítači jsou nainstalovány knihovny a ovladače pro běh platformy a také vývojová prostředí Visual Studio a CodeWarrior s projekty všech verzí software. Na disku je také možné najít další podklady a dokumentaci k tanku. Při potřebě více USB konektorů, například při programování modulů pomocí USB programátoru, je vhodné využít externí USB hub.

2.4 Sběrnice CAN

Pro komunikaci mezi jednotlivými moduly platformy je použita sběrnice CAN podle standardního formátu CAN 2.0 A. Původně byla vyvinuta pro automobilový průmysl, kde byly kladeny nároky na spolehlivost a bezpečnost. Sběrnice je vysoce odolná vůči rušení a také umožňuje arbitráž zpráv podle priority.

2.4.1 Fyzické provedení sběrnice CAN

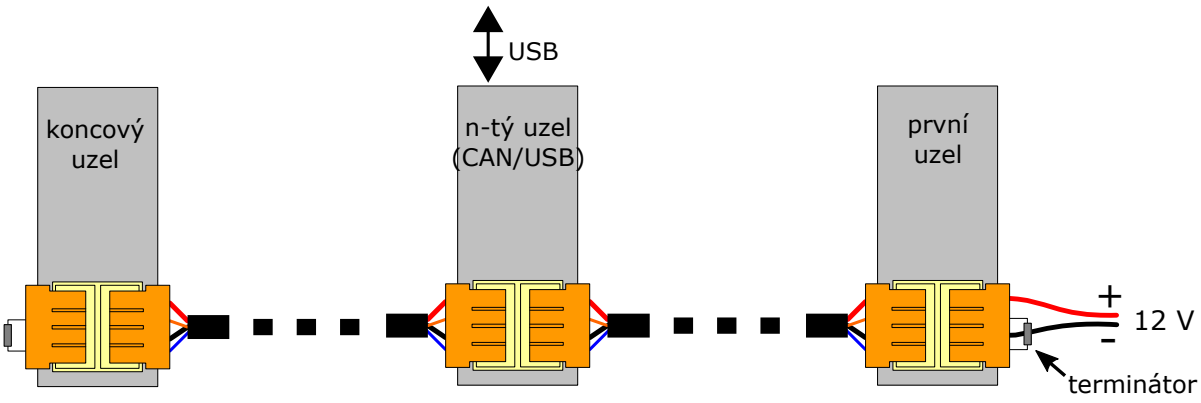
Sběrnice CAN je rozvedena pomocí kabelů se dvěma páry vodičů, jednoho páru pro napájení a jednoho (CANH a CANL) pro přenos diferenčního signálu. Každý uzel na sběrnici je osazen dvojitým konektorem VAGO 734-404, viz obrázek 2.4, což umožňuje snadno rozšiřovat sběrnici o další moduly.



Obr. 2.4: Konektor kabelů sběrnice CAN [3]

Topologie sběrnice je znázorněna na obrázku 2.5. U této topologie jsou všechny uzly průchozí, včetně koncových, ke kterým jsou připojeny ukončovací odpory, tzv. terminátory. Proto není možné jednoduché odpojení jednoho z uzlů bez přepojování kabelů pro zachování sběrnice. Komunikace s centrálním počítačem je zajištěna pomocí převodníku USB/CAN který je zařazen jako jeden z uzlů. Napájení je zavedeno od hlavního vypínače

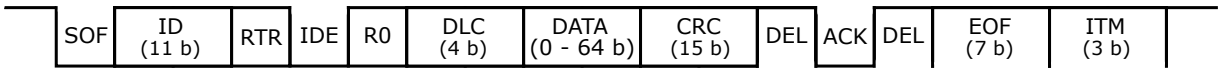
přímo z baterie platformy a slouží pro napájení všech připojených modulů. Jen pro napájení výkonových obvodů, jako jsou budiče motorů, jsou vedeny extra vodiče o větším průřezu.



Obr. 2.5: Topologie sběrnice CAN [3]

2.4.2 Protokol sběrnice CAN

Formát zpráv na sběrnici CAN 2.0 A umožňuje přenést kromě svého 11-bitového identifikátoru (ID) také až 8 datových bajtů. Vysílat může jakýkoliv uzel a informace o prioritě, obsahu, původu a účelu zprávy jsou obsaženy v jejím ID, případně také v RTR bitu. Struktura zprávy je znázorněna na obrázku 2.6.

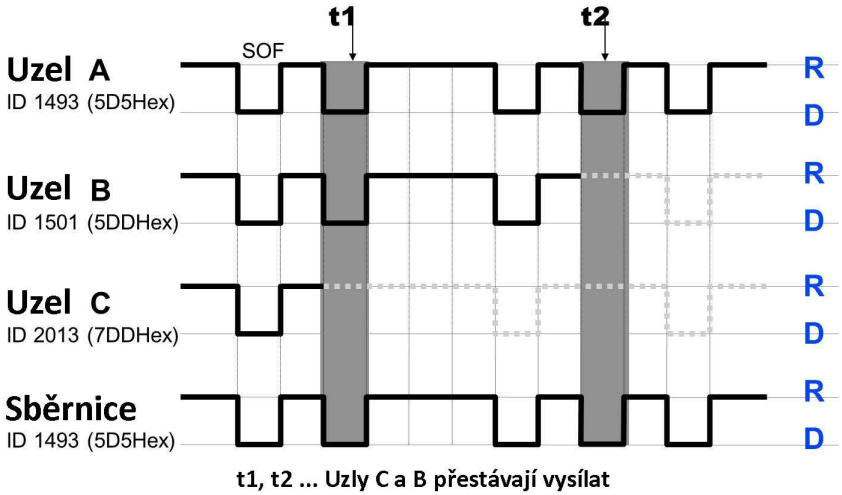


Obr. 2.6: Zpráva protokolu CAN[4]

Pro zamezení situacím, kdy by vysílaly dva uzly současně, je využito jednoduché arbitráže. Ze specifikace ve standardech vyplývá jakým způsobem tato arbitráž probíhá. Je provedena vždy na začátku vysílání porovnáním prvních bitů (ID) zpráv. Nižší ID vždy znamená vyšší prioritu. To funguje díky dominantní povaze bitů '0', které přepíší recesivní bity '1', viz obrázek 2.7.

Sběrnice je na platformě provozována na rychlosti 100 kbit/s. Při tomto kmitočtu může jedna zpráva trvat 0,5 ms až 1,24 ms, podle její délky a počtu "bit stuffing" bitů. Z toho vyplývá minimální odezva komunikace po této sběrnici v řádu jednotek ms.

Jak již bylo dříve zmíněno, priorita zpráv je dána jejím ID. Nižší ID mají vyšší prioritu a byly v platformě přiřazeny jednotlivým zprávám během předchozí práce. Nejnižší hodnotu ID a tedy nejvyšší prioritu mají na sběrnici zprávy týkající se ovládání motorů. Zpracování zpráv při komunikaci s jednotkou motorů a s jednotkami senzorů je vysvětleno v následujících kapitolách a více do detailu popsáno v předchozí práci [1].

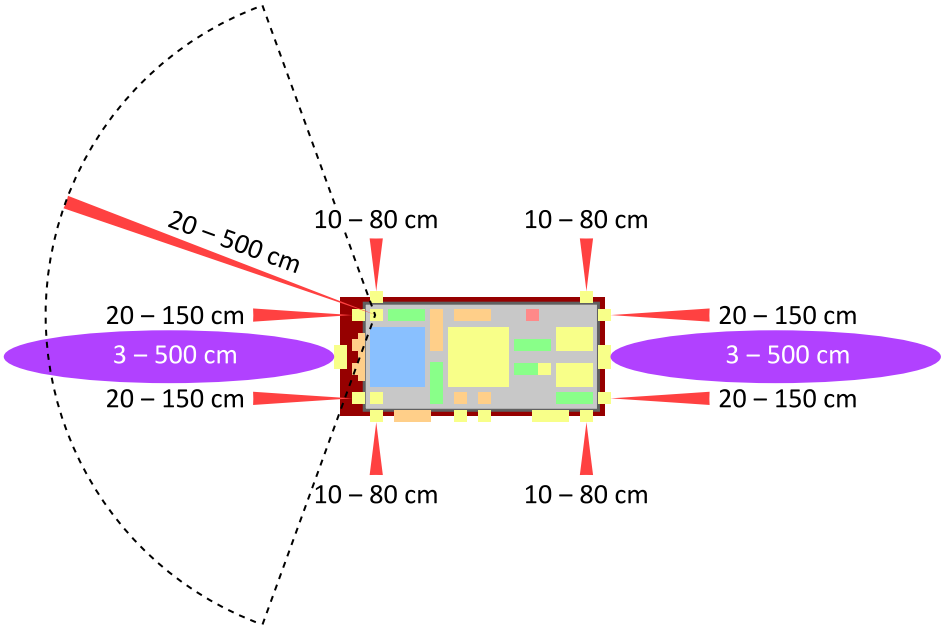


Obr. 2.7: Kolize zpráv s různým ID [5]

2.5 Moduly senzorů

Platforma je vybavena třemi moduly senzorů, které provádí sběr dat z několika druhů senzorů, a ty posílají tyto informace po sběrnici CAN k dalšímu zpracování. Všechny tři moduly používají stejný firmware a i z hlediska HW mají stejný základ.

Obsluhují dva druhy senzorů přiblížení, znázorněné na obrázku 2.8. Vpředu a vzadu platformy jsou osazeny ultrazvukové senzory, vyznačené fialově. Tyto senzory mají relativně vysoký rozptyl a detekují překážku kdekoliv ve svém úhlu dosahu - to vyplývá z jejich akustické podstaty. Směrovější jsou pak senzory infračervené, vyznačené červeně, které jsou na tanku osazené ve třech verzích dosahů. Boční senzory mají menší dosah, na rozdíl od předních.



Obr. 2.8: Znázornění dosahů infračervených a ultrazvukových senzorů [1]

Přední modul obsluhuje infračervené senzory v přední polovině platformy. Obsluhuje také ultrazvukový senzor a servo otáčející infračerveným senzorem na přídi tanku. Podporuje také automatický režim otáčení servem.

Střední modul obsluhuje akcelerometr a gyroskop a také kompas namontovaný na přídi tanku. Ovládá také serva otáčející věží. Napájení těchto serv však pro jejich vysoký odběr zprostředkovávají stejnosměrné měniče na 5 V integrované v budičích motorů platformy.

Zadní modul obsluhuje infračervené senzory v zadní polovině platformy. Obsluhuje také LCD display na boku platformy a serva ramen věže.

Firmware modulů senzorů ovládajících serva byl optimalizován zrychlením frekvence modelářské PWM z běžných 50 Hz na 100 Hz.

2.6 Jednotka motorů

Během této práce prodělala jednotka motorů instalovaná během jedné z prvních prací výrazné změny. Kvůli přetrvávajícím problémům s přehříváním její výkonové části byla tato část kompletně odstřižena od napájení a byly přidány modelářské budiče DC motorů QuicRun 1060 V2 s psaným trvalým proudovým zatížením 60 A, viz obrázek 2.9.

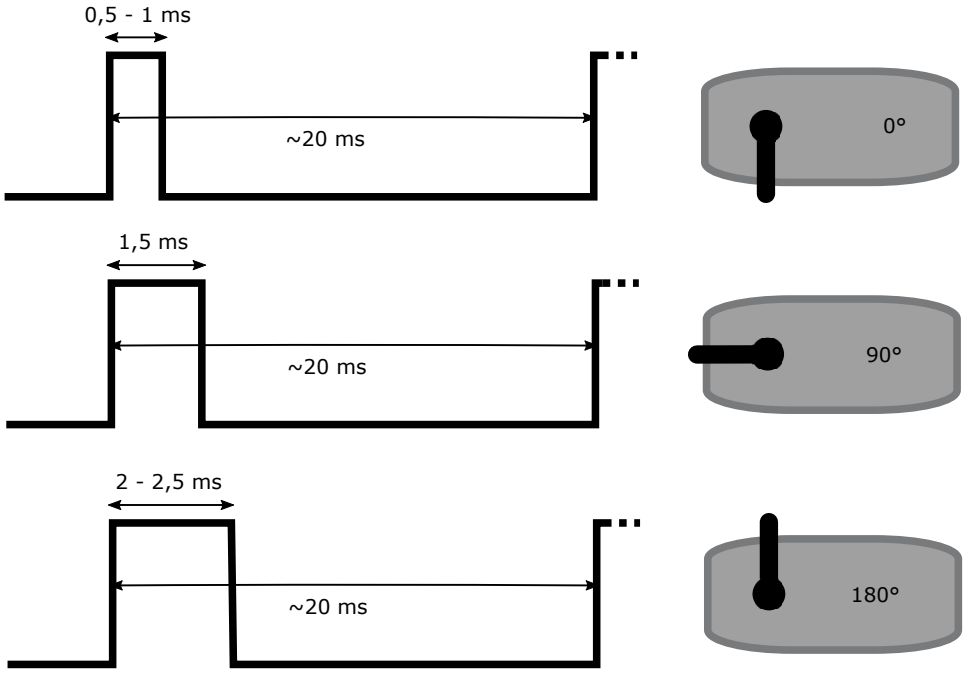
2.6.1 Budiče motorů

Obr. 2.9: Budiče motorů QuicRun [6]

Budiče byly pomocí jumperů nakonfigurovány tak, aby nebyly v běžném režimu vpřed - brzda - vzad, jako se používá u RC modelů aut. Místo toho jsou v režimu vpřed - vzad, což je vhodné pro diferenční řízení jako u pásových vozidel, kde je zapotřebí častých změn směru otáčení motorů pro manévrování v terénu.

Pro ovládání modelářských budičů, stejně jako všech serv na platformě, se používá signál typu PWM. Údaj přenášený tímto druhem PWM signálu nezávisí přímo na jeho střídě ani na periodě, ale pouze na šířce samotného pulzu, jak je vykresleno na obrázku

2.10. Šířka tohoto pulzu se pohybuje v mezích od 1 ms do 2 ms, někdy serva podporují i rozsah 0,5 ms - 2,5 ms. Perioda tohoto signálu se běžně udává 20 ms, ale dekodéry modelářských zařízení podporují bez problému i periody kratší a delší. [7] Toho bylo na platformě využito a perioda byla u jednotky motorů, podobně jako u modulů senzorů ovládajících serva věže, zkrácena na 10 ms pro dosažení rychlejší odezvy systému.



Obr. 2.10: Využití signálu PWM pro ovládání serv a budičů motorů

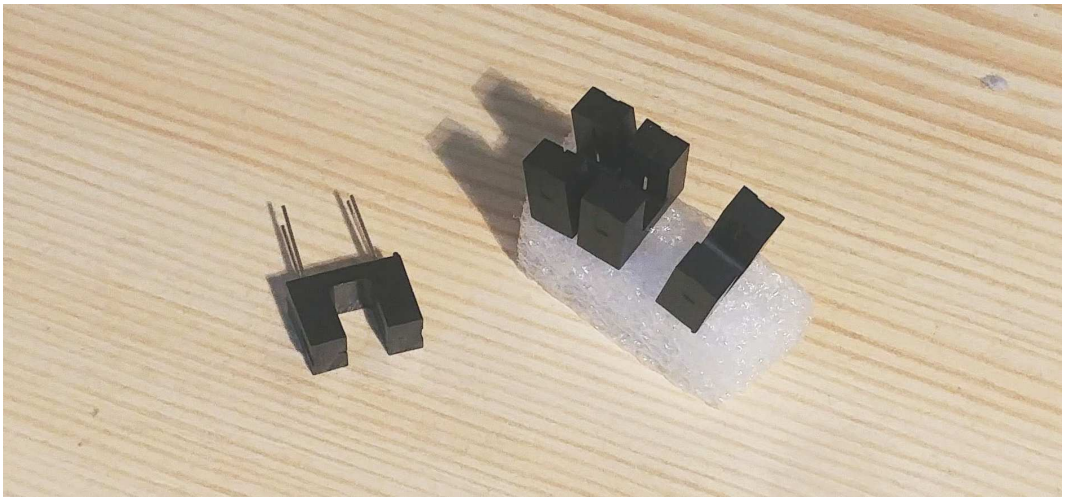
Piny mikrokontroléru jednotky motorů, které sloužily k ovládání H-můstku v její výkonové části byly využity k ovládání nových budičů.

2.6.2 Senzory otáček

Další nutnou změnou na jednotce motorů bylo přidání hardwarové detekce směru otáčení pásů. V předchozí verzi jednotka detekovala pouze rychlost otáčení a směr otáčení pouze odhadovala softwarově pomocí několika podmínek v programu. To naprosto znemožňovalo regulaci otáček (tedy v jejím pravém slova smyslu) v oblasti rychlostí kolem nuly. Výsledkem toho byly velmi pomalé změny směru otáčení pásů nutné pro zajištění stability. Kvůli setrvačnosti pásů a nepředvídatelnosti terénu bylo softwarové řešení v principu nedostatečné a bylo proto v rámci této práce nahrazeno hardwarovým.

Původní senzory otáček s jednou optickou závorou, která mohla na děrovaném terčíku určit pouze rychlost, byla nahrazena senzory s dvojicemi optických závor, viz obrázek 2.11. Byly vyrobeny nové desky plošných spojů na fréze ZČU a osazeny a byly instalovány místo původních čidel. Děrné štítky na hřídeli byly ponechány původní, i přes své nepřesnosti. Pro správnou funkci bylo nutné senzory otáček dobře vyladit.

Jelikož byly původní senzory otáček původně připojeny k externímu čítači v integrovaném obvodu, byly výstupy nových optických závor zapojeny na další piny mikrokontroléru

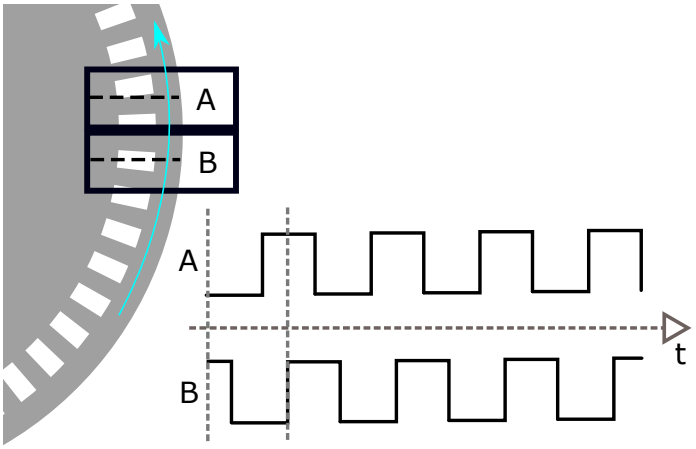


Obr. 2.11: Dvojice nových optických závor

uvolněné odpojením výkonové části jednotky. Tím bylo umožněno dekódovat vytvořený kvadraturní dekodér s adekvátním časováním.

2.6.2.1 Princip funkce senzorů otáček - kvadraturní dekodér

Zdvojením optických závor na děrném štítku došlo k vytvoření senzoru fungujícího na principu kvadraturního dekodéru. Princip kvadraturního dekodéru je ilustrován na obrázku 2.12. Spočívá v porovnávání sledu pulsů přicházejících z fázově posunutých optických závor. V ideálním případě, při dosažení fázového posunu 90° je poloha náběžných a doběžných hran přesně po čtvrtinách periody. Směr se pak pozná podle toho, ze které optické závory přichází náběžná hrana první. Problémy mohou nastat tehdy, pokud by fázový posun překračoval 0° nebo 180°.



Obr. 2.12: Funkce kvadraturního dekodéru otáček platformy

Časování tohoto senzoru otáček (kvadraturního dekodéru) je kritické pro jeho správnou funkci. Náběžné hrany musí být od sebe vzdáleny přibližně 90°, tedy 1/4 periody, a to po celém obvodu děrného štítku a také pro všechny rychlosti. Ladění tohoto fázového

posunu bylo v našem případě řešeno posunem dvojice optických závor blíže či dále od středu děrného štítku, čímž se měnil rozestup drážek vůči rozestupu optických závor. Tím se tedy měnila i vzájemná vzdálenost náběžných hran. Během tohoto ladění se nám několikrát stalo, že fázový posun překročil polovinu periody. To se projevilo jako kladná zpětná vazba a tedy nestabilita a oscilace regulátoru otáček v jednotce motorů.

2.6.3 Firmware jednotky motorů

Firmware jednotky byl upraven co se týče jeho regulačního algoritmu. Původní fuzzy regulátor byl nahrazen klasickým PID regulátorem. Byla také vyřazena softwarová detekce směru a tedy i omezení v pásmech u rychlostí kolem nuly. Hlavní nevýhodou fuzzy regulátoru bylo jeho složité ladění, které vyžadovalo změny mnoha konstant pro změnu jeho charakteristiky, obzvláště při jeho aplikaci v jazyce C.

PID regulátor poskytuje ověřené řešení a relativně snadné ladění systému. V kódu je také srozumitelnější pro každého programátora, jelikož jde o daleko častěji používané řešení. K jeho ladění se využívá tří konstant, proporcionální, integrační a derivační. Jeho vstupní hodnota se nazývá chyba a jde o rozdíl požadovaných a aktuálních otáček za minutu na hnací hřídeli pásů.

Byl přidán také kód pro snímání otáček pomocí přerušení, a jednoduché rozpoznání směru podle sledu náběžných hran z každého senzoru otáček. Nový PID regulátor spolu s čidlem otáček výrazně zkrátil odezvu systému, přibližně o vteřinu.

2.6.3.1 PID regulátor v jazyce C

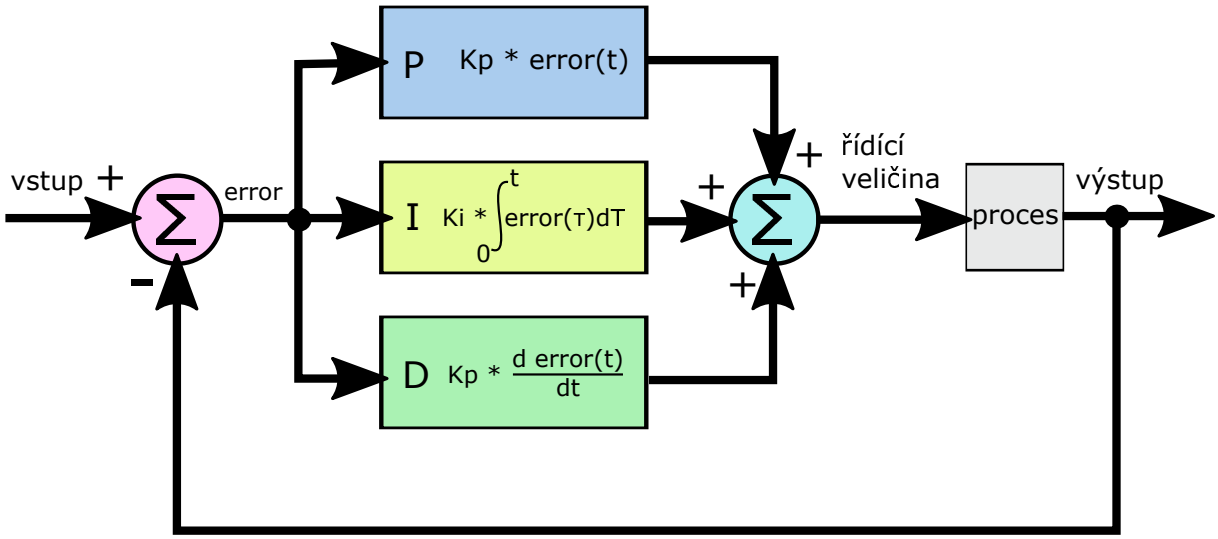
Princip PID regulátoru se běžně definuje pro spojitý průběh s ideálním integračním a derivačním článkem, viz obrázek 2.13. Pro aplikaci v programu mikrokontroléru jednotky motorů bylo však nutné tento regulátor použít v diskrétní formě a ideální integrační a derivační články nahradit vhodnými ekvivalenty.

Regulátor se v diskrétní formě v kódu spouští v pravidelných intervalech ve smyčce. Nejprve se pro získání hodnoty chyby (error) odečítají požadované otáčky od naměřených reálných otáček na hnací hřídeli pásů.

Řídící veličina je rovna součtu tří složek, jejichž velikost se odvíjí od hodnoty a průběhu chyby. Proporční složka se vypočítá jednoduše vynásobením aktuální chyby proporční konstantou.

Integrační složka se získá vynásobením sumy všech předchozích chyb integrační konstantou. Tato suma je diskrétní náhradou za spojitý integrátor. Během stabilního provozu se pohybuje kolem nuly díky střídání kladných a záporných hodnot chyby. Přesto ale musí být ošetřena proti přetečení.

Derivace chyby se aproximuje odečtením předchozí hodnoty chyby od její současné hodnoty, tedy z její diference. Vynásobením diference chyby s derivační konstantou získáme derivační složku.



Obr. 2.13: Princip PID spojitého regulátoru

Pro správnou funkci regulátoru je nutné dodržet spouštění smyčky vždy ve stejných časových intervalech. Po součtu všech tří složek je také nutné ošetřit řídící veličinu proti překročení maximální nebo minimální hodnoty.

```
// Ukázka kódu PID regulace

regValue = 0;                                // nulování řídící veličiny

error = GET_MOTOR1_RPMW - leftRPM;           // výpočet chyby

sumRpmError += error;                        // inkrementace sumy chyb
sumRpmError += (sumRpmError > 0) ? -1 : 1;// snižování hodnoty sumy o 1
if(sumRpmError < -sumRpmErrorMax)            // ošetření přetečení sumy
  { sumRpmError = -sumRpmErrorMax;  }
if(sumRpmError > sumRpmErrorMax)
  { sumRpmError = sumRpmErrorMax;  }

regValue += Pconst * error;                  // přičtení proporční složky
regValue += sumRpmError / Iconst;            // přičtení integrační složky
regValue += Dconst * (lastError - error); // přičtení derivační složky

lastError = error;                           // uložení poslední chyby

if(regValue < -regValueMax)                  // omezení řídící veličiny
  { regValue = -regValueMax;  }
if(regValue > regValueMax)
  { regValue = regValueMax;  }

regValue += 3000;                            // centrování řídící veličiny
BRIDGE_PWM_WIDTH(regValue);                  // uložení výstupu
```

Snižování hodnoty sumy chyby je implementováno proto, aby se po zastavení platformy

vrátila řídící veličina do nuly a nezůstávala v nízkých hodnotách, které sice pásy neotočí, ale způsobují nepříjemné pískání budičů motorů.

2.7 MS Kinect

Platforma je osazena zařízením Kinect for Xbox 360, která vydala firma Microsoft v roce 2010. Zařízení z důvodu svého stáří již nemá podporu a postrádá online dokumentaci od Microsoftu, ale z předchozích prací [1] a ze zdrojů třetích stran [8] známe jeho parametry. Disponuje barevnou kamerou s rozlišením 1280x960 pixelů a hloubkovou kamerou s rozlišením 640x480 pixelů. Má sadu čtyř mikrofonů a také trojosý akcelerometr.

Hloubková kamera využívá k měření také třetí objektiv, ve kterém je rozmítaný infračervený laser. Laser vysílá pulsy do jednotlivých pixelů a kamera měří čas a tedy i vzdálenost jednotlivých pixelů v prostoru. Vzdálenost, na kterou je hloubková kamera schopna měřit, je přibližně 0,7 - 6 metrů.

Úhel záběru obou kamer je 43° vertikálně a 57° horizontálně [9]. To znamená, že postava vysoká 180 cm se do záběru může teoreticky vejít celá od vzdálenosti 228 cm, pokud bude přesně uprostřed záběru a pokud bude kamera ve výšce 90 cm.

Obr. 2.14: MS Kinect bez plastového krytu[10]

Zařízení Kinect komunikuje s centrálním počítačem pomocí USB. Využívá navíc externí napájení 5 V přímo ze zdroje počítače - při napájení z USB by hrozilo přetížení USB. Dále využívá knihovny od Microsoft KinectSDK v1.8, které umožňují další zpracování dat, o kterém se zmíníme v dalších kapitolách.

2.8 Otočná věž

V rámci této práce byla k platformě přidána otočná věž, primárně pro zvýšení polohy umístění zařízení MS Kinect. V minulosti byla jeho funkce značně omezena jeho umístěním blízko země a pevným úhlem jeho natočení. Kvůli jeho poloze nebylo možné rozpoznat lidskou postavu v blízkosti zařízení jednoduše proto, že postavu nebylo možné dostat do záběru. Proto byla navržena a vyrobena otočná věž, která vyvyšuje MS Kinect o cca 45 cm výše oproti jeho původní poloze a umožňuje jeho natáčení ve dvou osách. K otáčení celé věže, a nejen její horní části, bylo přistoupeno po otestování pevnosti kloubu a po uvážení výhod plynoucích z možnosti otáčet celou věží.

Obr. 2.15: Kloub věže tanku [11]

Návrh konstrukce věže se odvíjel od samotného otočného kloubu, který je tvořen točnou vyráběnou pod televizory a pro otočné nástavby stolů s nosností až 100 kg, viz obrázek 2.15 . Pro zajištění naprosto minimální vůle v ložisku byla točna navíc sevřena šroubem ve středu otáčení. Jako základna věže byla použita 10 mm deska z PVC (medur), která je instalována nad krycím plexisklem platformy na vyvyšujících 50 mm šroubech - pro zajištění přístupu pod točnu pro spojování konektorů věže.

2.8.1 Návrh otočné věže

Věž byla navržena v programu Autodesk Fusion 360. Design byl záměrně zvolen takový, aby nebudil dojem vojenského útočného tanku a aby byl dodržen výukový záměr platformy - "aby bylo vidět dovnitř". Byla zohledněna také pevnost a hmotnost věže, proto bylo využito žebrování a širší základna. Celkové rozměry věže byly omezeny pracovním prostorem 3D tiskárny KAE ZČU, a jsou nakonec 297 mm na 195 mm. Věž bylo tak možné vytisknout z jednoho dílu a nebylo zapotřebí lepení. Věž se tiskla z PLA plastu se

100% výplní přes 30 hodin. Do návrhu věže byly zahrnuty otvory pro šrouby pro uchycení kabeláže a také dvě plochy s otvory pro malá devítigramová serva. Tyto plochy mohou být použity pro uchycení ramen věže či jiného příslušenství. Ve spodní části je možné na místo současně umístěného potenciometru přímo připevnit standardní servo. Tak tomu bylo v první verzi otočného mechanismu věže. Na vrcholu věže je připevněné servo s držáky pro náklon MS Kinect.

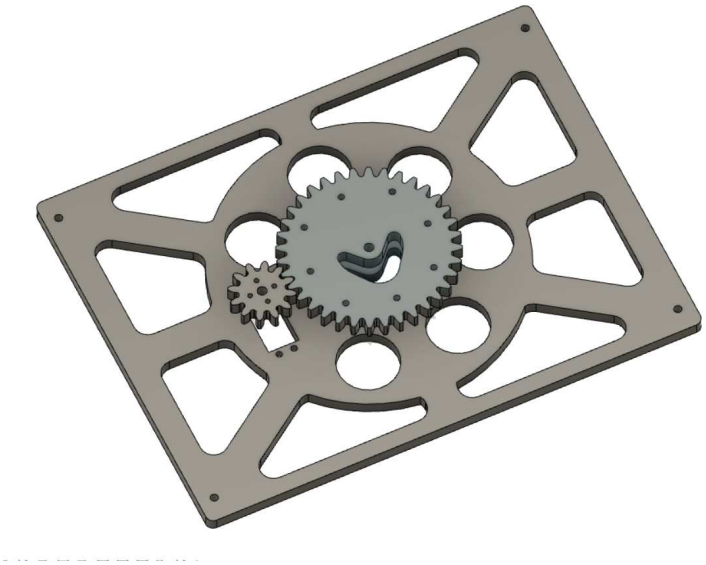
Obr. 2.16: Návrh věže v Autodesk Fusion 360

Obě hlavní serva pro otáčení věže a pro náklon MS Kinect jsou ovládána ze střední jednotky senzorů. Napájení 5 V berou ze step-down měničů v budičích motorů platformy, které jsou na jejich výkon stavěné. Při použití 5 V napájení z jednotky senzorů docházelo k jejímu přetěžování při jakémkoliv rychlejším pohybu. Jejich špičkový příkon může snadno překročit i 1 A.

2.8.2 Servo otáčení věže

Při prvních experimentech se servem umístěným přímo v ose otáčení, na místě nyní umístěného potenciometru, se ukázalo, že taková zátěž na servu způsobuje drobné kmitání věže ve směru rotace. Příčinou je naladění jeho vnitřní regulace na nižší setrvačnost zátěže. Vzniklé oscilace se přenášely do kamery na vrcholu věže a působily problémy se snímáním obrazu při otáčení věží. Protože vnitřní regulátor serva není možné ladit, byla zvolena úprava celého mechanismu otáčení. Zátěž na servu byla upravena přidáním pře-

vodu z ozubených kol. Ta byla vyrobena ze stejného PVC materiálu jako deska základny pomocí laserové vypalovačky. Ozubená kola jsou v poměru 3:1 a servo je tedy zatíženo jen třetinovým momentem. Výsledkem je sice pomalejší, ale výrazně plynulejší otáčení věže bez kmitání. Ozubená kola také působí lépe esteticky a reprezentativně, viz obrázek 2.17.



Obr. 2.17: Ozubená kola v základně věže

Pro získání dostatečného rozsahu otáčení, bylo servo modifikováno pro kontinuální rotaci. Z převodovky serva byly odstraněny koncové zarážky a byl odpojen vnitřní potenciometr. Z místa potenciometru byly následně vyvedeny kabely pro připojení externího potenciometru pro uzavření zpětnovazební smyčky k vnitřnímu regulátoru serva, viz obrázek 2.18. Potenciometr s rozsahem 270° a odporem 5 kiloohmů byl osazen ve středu věže. Pro případ odpojení nebo jiného selhání zpětné vazby z potenciometru bylo velké ozubené kolo opatřeno koncovou zarážkou, aby nemohlo dojít k překroucení kabelů vedoucích do věže.

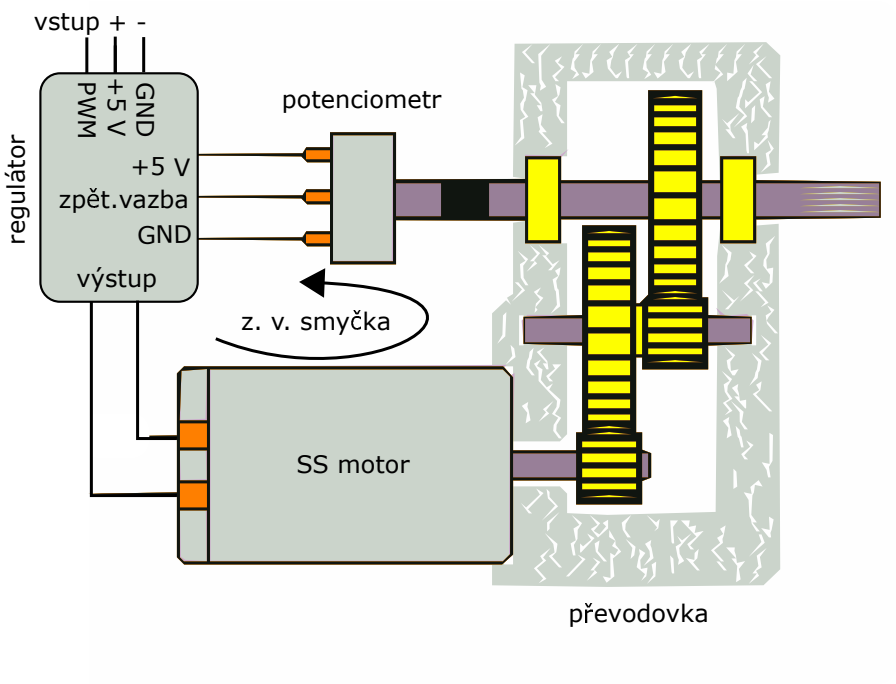
2.8.3 Ramena věže

Po stranách věže byla přidána malá serva pro vytvoření ramen. Ramena jsou zatím jen velmi jednoduchá, ale umožňují platformě vizuálně signalizovat svůj stav. Řídící aplikace může tato ramena také využívat například pro napodobování lidské gestikulace.

Levé rameno bylo osazeno praporkem a na pravé rameno byly instalovány 5 V LED světlo a 5 V laserová dioda napodobující laserové ukazovátko. Toto příslušenství je ovládáno pomocí budičů získaných z modelářských serv.

2.8.3.1 Použití serva jako budiče

Každé klasické servo se skládá ze čtyř základních částí, viz obrázek 2.18:



Obr. 2.18: Vnitřní diagram serva [22]

- `Regulátor` - dekóduje vstupní PWM signál, snímá polohu potencimetru a pomocí H-můstku budí motor napětím, které je úměrné rozdílu zpětné vazby a vstupního signálu
- `Motor` - stejnosměrný motor s malým kroutícím momentem
- `Převodovka` - konvertuje relativně vysoké otáčky motoru na vyšší kroutící moment
- `Potenciometr` - snímá úhel natočení hlavní osy serva

Jak vyplývá z vnitřní konstrukce serva, není složité jeho komponenty použít i pro jiné účely. Fixací potenciometru v jeho střední poloze (přerušením mechanické zpětné vazby mezi potenciometrem a převodovkou) a odstraněním koncových zarážek získáme servo pro kontinuální rotaci. Pokud zpětnovazební signál zůstává neutrální, je pak výstup, který je roven rozdílu vstupního a zpětnovazebního signálu, přímo roven vstupu. Místo motoru s převodovkou můžeme pak na regulátor připojit i jinou zátěž a využít tak vnitřní komponenty serva jako budič.

Výhodou takového budiče je jeho malá velikost, která se blíží tloušťce samotného kabelu. Modelářská serva jsou také nejdostupnějšími ze zařízení schopných dekódovat tento typ PWM signálu. Nevýhodou může být nízká výkonová zatížitelnost. Tu poskytují několikanásobně dražší stejnosměrné budiče, jako např. ty použité u jednotky motorů.

3

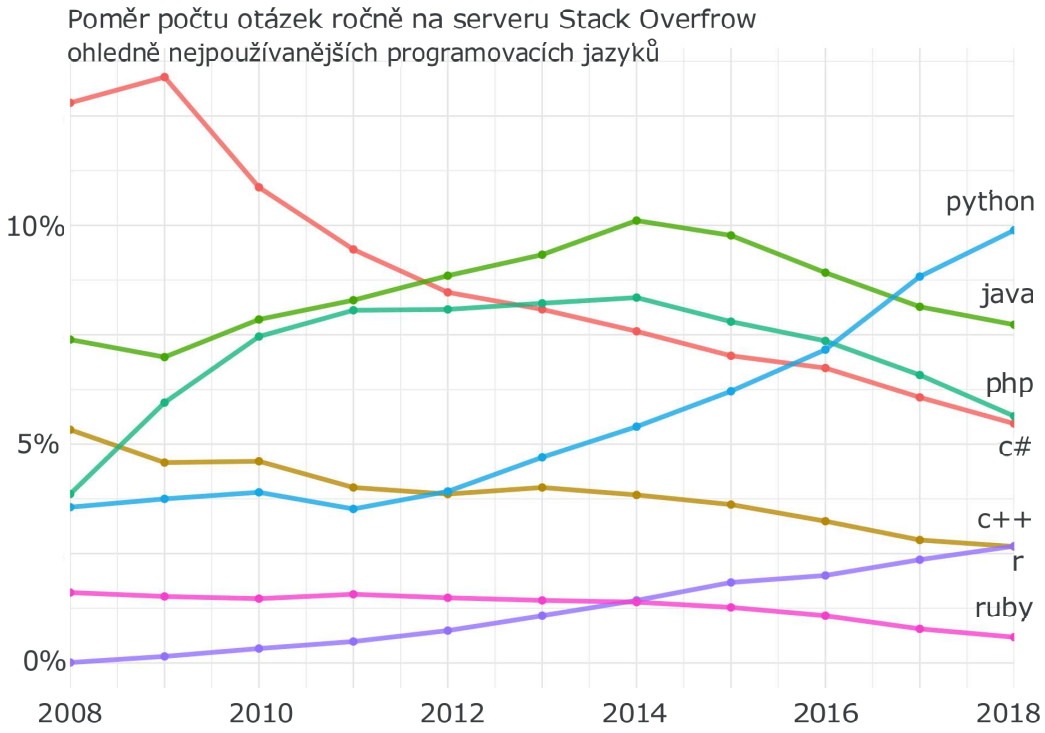
Programování v jazyce C#

V této kapitole se budeme zabývat základem a principy programování v jazyce C# pro umožnění správného postupu při vývoji aplikace pro autonomní platformu. Každý jazyk má svá specifika, která je potřeba znát, aby bylo možné využít jeho potenciál. V rámci této práce byly v první části řešeny úpravy firmware v jazyce C, který je velmi oblíbený v real-time aplikacích zajišťujících běh základních funkcí systémů, jako je v našem případě regulace motorů nebo obsluha senzorů. Je ale výrazný rozdíl mezi programováním pro integrovaná řešení v jazyce C a programováním aplikací pro PC.

C# je objektově orientovaný jazyk, který vznikl na základech ostatních jazyků C, především C++, a také byl inspirován jazykem java. Vyvinut byl od firmy Microsoft spolu s platformou .NET framework, pro kterou byl vytvořen. Jazyk byl následně standardizován organizací Ecma International jako ECMA-334. První verze 1.0 tohoto jazyka spolu s první verzí .NET framework a vývojového studia Visual Studio byla vydána roku 2002 [**?**]. V dnešní době je již dostupná i stabilní verze 8.0. Díky neustálému vývoji ze strany Microsoft si C# stále drží přední příčku v oblíbenosti u vývojářů. Je celkem logické, že vysokoúrovňový jazyk vyvinutý od stejné společnosti, která vyvinula i samotný operační systém, bude mít u jeho uživatelů vždy podstatnou výhodu.

Pro vývoj aplikace v jazyce C# bylo přistoupeno z velmi praktických důvodů. Jde o jeden z nejpoužívanějších jazyků pro vývoj aplikací pro OS Windows. V době vydání zařízení MS Kinect (2010) byl zdaleka nejpopulárnější, viz obrázek 3.1, na kterém můžeme vidět statistiky z frekvence dotazů týkajících se tohoto jazyka na oblíbeném serveru Stack Overflow [13]. V tomto jazyce jsou pochopitelně dostupné i pro nás potřebné knihovny.

Na grafu je vidět, že špička popularity jazyka C# na programátorském fóru Stack Overflow byla kolem let 2008 až 2010 a v následujících letech je vidět mírný pokles jeho popularity. Zároveň je vidět nárůst popularity dalších jazyků - nejdříve jazyka java a následně i skriptovacího jazyka python, které se v dnešní době také využívají pro vývoj aplikací pro OS Windows. Pro jejich použití by ale bylo obtížné zprovoznit knihovny již implementované na platformě a knihovny vydané firmou Microsoft specificky pro jazyk C# .



Obr. 3.1: Vývoj popularity jazyků 2008 - 2018 [13]

3.1 Specifikace jazyka C#

Čím se tento jazyk vymezuje a v čem je podobný jiným jazykům si můžeme shrnout následujícím přehledem.

3.1.1 Jazyk s virtuálním strojem

Pro pochopení, jak funguje jazyk s virtuálním strojem, jakým je právě jazyk C#, je nutné vysvětlit podstatu virtuálního stroje.

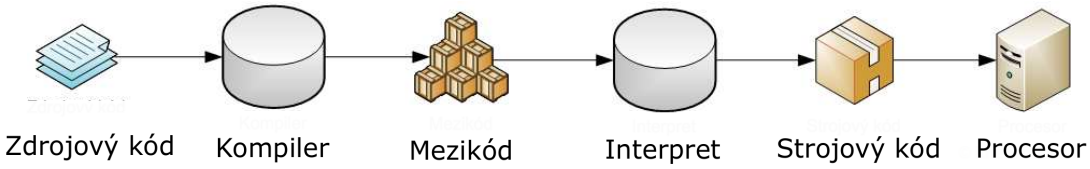
Každý procesor potřebuje ke svému běhu strojový kód, který vypadá jako jedničky a nuly. Ty jsou ale k programování naprosto nepřehledné a pro vývoj aplikací v dnešní době zcela předpotopní. Proto se používají takzvaní interpreti, nebo kompilery, které umí generovat strojový kód ze zdrojového kódu, který je pro člověka čitelnější. Účel tohoto procesu je umožnit psaní programu v kódu, který je pro programátora srozumitelnější, přehlednější, bezpečnější, pohodlnější a vůbec výhodnější.

Kompiler, neboli překladač, bere zdrojový kód od programátora a ten zkompiluje (sestaví) do strojového kódu. Uživatel pak může strojový kód spustit, ale nemá už možnost nahlížet do zdrojového kódu. Zpětný překlad pro získání původního kódu ze strojového kódu není možný. To je užitečné pro uchování duševního vlastnictví autora. Kompiler také pomáhá odhalit chyby a hlásí autorovi, z jakého důvodu nelze kód přeložit, což usnadňuje vývoj. Nevýhodou takto vytvářených programů může být nemožnost úprav po

kompilaci. To může brzdit vývoj u složitějších aplikací, u kterých kompilace trvá delší dobu. Obzvláště v případech, kdy se opakovaně projevují chyby, které kompiler nemá možnost odhalit. Jazyky používající kompilery jsou např. jazyk C nebo C++.

Tzv. interpret na druhou stranu umožňuje zdrojový kód spustit ihned, protože ho překládá do strojového kódu v reálném čase. Vždy překládá tu část kódu, kterou je zapotřebí provést. Rozdíl mezi kompilerem a interpretem se dá přirovnat k rozdílu mezi překladatelem knih a tlumočníkem. Výhodou simultánního překladu je snazší odhalení chyb kódu po překladu, které se projevují ihned. Díky tomu je u takového jazyka snazší vývoj. Další výhodou je také přenositelnost programu, program lze spustit na jakémkoliv zařízení s vhodným interpretem. Velkou nevýhodou interpretovaných jazyků je ale jejich pomalejší běh plynoucí ze zdlouhavé interpretace a také žádná ochrana duševního vlastnictví. Příkladem takového jazyka je například jazyk PHP.

Jazyk s virtuálním strojem využívá jak kompiler, tak interpreta. Účelem je získání výhod obou systému a minimalizaci jejich nevýhod. Kompiler v tomto případě nevytváří přímo strojový kód, ale generuje tzv. mezikód, který je předkládán interpretovi. Interpret slouží jako tzv. virtuální stroj a překládá mezikód. Ten je již v binárním formátu, proto překlad do strojového kódu pro procesor probíhá daleko rychleji, viz obrázek 3.2. Mezikódu pro platformu .NET se říká CIL neboli Common Intermediate Language. Ten slouží jako mezistupeň mezi zdrojovým kódem psaným vysokoúrovňovým programovacím jazykem a úplně základním strojovým kódem. Stále umožňuje určitou detekci chyb při běhu, ochranu duševního vlastnictví a také relativně vysokou rychlost díky své zjednodušené sadě instrukcí [14].



Obr. 3.2: Princip překladu jazyka s virtuálním strojem [14]

3.1.2 Objektově orientované programování

Objektově orientované programování, oproti běžnému programování s jazyky nižších úrovní, jako například C, přináší široké spektrum nástrojů pro zjednodušení návrhu. Umožňuje organizaci kódu (proměnných, dat, metod...) do objektů, skrze které se můžeme odkazovat na prvky uvnitř. Principy jsou velmi podobné těm u struktur v jazyce C, ale dále rozšířené pro použití v daleko širším smyslu. Hlavní principy jsou následující [16]:

- `Zapouzdření` - je možné zahrnout skupinu vlastností, metod a jiných objektů do jednoho objektu a přistupovat k němu jako k jednomu objektu

- `Dědičnost` - je možné použít existující třídu k vytváření nových tříd a ty budou mít stejné vlastnosti jako třída původní
- `Polymorfismus` - různé třídy mohou obsahovat stejně pojmenované metody a vlastnosti, jejichž použití a volání probíhá stejně, ale různé třídy se projeví různými výstupy

3.1.2.1 Třídy

Zmíněné třídy jsou v podstatě typy objektů a vlastní objekty jsou vždy vytvořené jako zástupci některé z tříd. Třída slouží pro definování, jak bude objekt vypadat, ale není možné s ní pracovat dokud vlastní objekt dané třídy nevytvoříme.

Uvnitř třídy může být mnoho různých členů tříd, z nichž nejčastěji používané jsou následující [16] :

- `Vlastnosti a pole (proměnné)` - reprezentují informace uvnitř třídy, vlastnostem je možné upravovat přístup zvenčí pomocí jejich vlastností get a set
- `Metody` - funkce uvnitř tříd
- `Konstruktory` - funkce, které se spustí vždy a pouze při vytvoření objektu toho typu, používá se například k inicializaci proměnných
- `Finalizační metody` - metoda, která se používá k vyčištění paměti (pokud je to nutné) před samotnou destrukcí svého objektu
- `Události` - slouží k odchycení události z jiné třídy nebo k vyvolání delegáta události, který může byt odchycen jinde, kde je přihlášen odběr
- `Vnořené třídy` - vnitřní třídy jsou ve výchozím nastavení soukromé

3.1.2.2 Dědičnost

Díky dědičnosti je možné vytvořit novou třídu s použitím již existující třídy jako jejího základu a její vlastnosti upravit dle potřeby. Veškeré typy, i ty nejzákladnější jako jsou bool nebo int, jsou upravenými verzemi původní třídy object. Na tomto principu je založen jednotný systém typů jazyka C# a umožňuje konzistentní způsob práce napříč třídami [15].

Dědičnost je běžně využívána také pro přejímání a úpravu rozsáhlých tříd pro ušetření práce s tvorbou celé třídy od základu. Definovat novou třídu na základě existující, předefinovat nebo přidat pár jejích členů je často nejrychlejší postup jak získat spolehlivé řešení.

3.1.2.3 Programování orientované na komponenty

Ještě pokročilejším aspektem programování v jazyce C# je jeho podpora a orientace na komponenty. Pro vývoj naprosté většiny moderního software je použití hotových komponent nevyhnutelné. Při návrhu se spoléhá na hotové knihovny, moduly a balíčky funkcí dostupné od výrobců nebo z jiných zdrojů. Tyto komponenty obsahují i vlastní dokumentaci. Zjednodušení implementace těchto komponent a jejich vlastností včetně dokumentace je tedy jednou z klíčových vlastností vysokoúrovňových jazyků [15].

3.1.3 Ošetření rizik při programování

Vyšší jazyky jako je C# implementují funkce pro detekci některých základních problémů, které mohou nebo by mohly během běhu programu nastat. Mezi takové funkcionality se řadí například automatické uvolňování paměti po nepoužitých objektech. Dále se používá také vyvolávání a zpracovávání výjimek v případě výskytu chyby při běhu programu, což výrazně usnadňuje identifikaci zdroje problému. Výjimky jsou vyvolávány například při použití nesprávného typu, při překročení maximálních indexů polí nebo při čtení z neinicializovaných proměnných. Jazykům, které implementují opatření proti použití nesprávných typů, se říká typově bezpečné [15].

3.1.3.1 Použití výjimek

Výjimky je možné použít pro detekci a ošetření jakéhokoliv neočekávaného nebo potenciálně nebezpečného chování. Základní problémy je ošetřeny již v základu, jako například typová bezpečnost, ale je možné definovat i uživatelské výjimky. Výjimky jsou běžně také součástí softwarových komponent a zabezpečují jejich správnou implementaci.

Výjimky se v jazyce C# zpracovávají pomocí klíčových slov *try*, *catch* a *finally*. Vytváření vlastních výjimek se provádí pomocí *throw*. Pro ošetření části programu, která hrozí vyvoláním podmínky, se kód dává do bloku *try*, po kterém následuje přidružený blok *catch* s kódem, který se provede v případě, že k vyvolání výjimky dojde. Pokud je blok *catch* definován jen pro specifickou výjimku, může být použit ještě blok *finally* s kódem, který se vykoná při jakékoliv jiné výjimce.

Pro ilustraci uvedu příklad použití *try*, *catch* a *finally* přímo v programu:

```
try
{
   podezrelaFunkce(pochybnyParametr);
}
catch (ArgumentException e)
{
    Console.WriteLine("Funkce přerušena kvůli neplatnému parametru");
}
finally
{
```

```
    Console.WriteLine("Funkci se nepovedlo provést z jineho duvodu");
}

//pokračování programu
```

Pokud k vyvolání výjimky dojde mimo blok *try*, virtuální stroj, neboli CLR (angl. Common Language Runtime), se pokusí najít v zásobníku blok *catch* pro konkrétní typ výjimky. Velmi často ale takový blok nenajde, a tak dojde k přerušení běhu programu, a uživateli se zobrazí typické okno se zprávou o neočekávané výjimce [17].

To je výhoda programování s virtuálním strojem, kde kromě chyb, které se projeví již při pokusu o kompilaci (jako například chybějící středník), které odhaluje kompiler, je možné odhalovat chyby i po přeložení kódu.

3.1.4 Základní syntax jazyka C#

Syntaxe jazyka C# je blízce příbuzná jazyku C. Je podobná také jazyku java s několika rozdíly. Nejlépe si jeho vlastnosti můžeme přiblížit na typickém příkladu `Program.cs`:

```
using System;

namespace HelloWorld
{
  class Program
  {
    static void Main(string[] args)
    {
      string hello = "Hello World!";
      Console.WriteLine(hello);
    }
  }
}
```

Na tomto programu si můžeme ukázat následující charakteristiky [18] :

- `Organizace bloků kódu` - jednotlivé bloky jsou definované složenými závorkami, prázdné řádky jsou ignorovány
- `Ukončení příkazů středníkem` - každý příkaz je ukončen středníkem
- `namespace` - pojmenování bloků kódu; používá se k organizaci kódu, může obsahovat třídy (class) či další *namespace*
- `using` - slouží ke umožnění přístupu k dalším blokům kódu (*namespace*) mimo náš soubor Program.cs, umožňuje přístup ke knihovnám a dalším komponentům, v tomto konkrétním případě umožňuje přístup k funkci *Console.WriteLine()*
- `class` - třída obsahuje naši funkcionalitu, v našem případě program, který spouštíme

- `Main` - hlavní metoda, podle konvence slouží jako vstupní bod programu
- `proměnné` - každá proměnná musí mít definovaný typ a být před použitím inicializována

Oproti jazykům, jako je například python, je C# ve své syntaxi lehce konzervativnější co se týče používání závorek a středníků. Na druhou stranu je výhodné, že byly ponechány stejné operandy a operátory jako se používají v jazyce C.

3.2 Aplikace v operačním systému Windows

V této části si přiblížíme několik důležitých vlastností systému Windows, které definují chování aplikace vyvíjené v rámci této práce. Jak již bylo zmíněno, centrální počítač je vybaven operačním systémem Windows 7 Professional. Jde o normální operační systém někdy označován jako GPOS (anglicky General Purpose Operating System). Stejného typu jsou také OS Linux nebo Mac OS.

OS Windows byl navržen především okolo svého grafického rozhraní, jak už napovídá i jeho název - "okna". Umožňuje spouštění množství náročných aplikací se složitou grafikou a jejich organizaci za účelem dosažení co nejlepší uživatelské zkušenosti. O efektivní běh všech spuštěných aplikací a procesů uvnitř počítače se stará jádro operačního systému neboli kernel. [19]

3.2.1 Porovnání Windows a RTOS

RTOS (anglicky Real Time Operating System) neboli Operační systém reálného času je typ operačního systému který se nejčastěji používá v integrovaných řešeních. Oproti GPOS, jako je například Windows, jsou RTOS navrženy kolem spolehlivého a přesného dodržení časování. To je nutné v systémech, kde na čase závisí třeba bezpečnost nebo spolehlivost. Mezi takové systémy by se počítala i naše mobilní platforma, pokud by mělo jít o komerční produkt.

3.2.1.1 Plánovač - priority

Plánovač, anglicky task scheduler, je zodpovědný za přepínání mezi jednotlivými souběžnými procesy. U GPOS je navržený tak, aby bylo dosaženo co nejvyšší efektivity využití výkonu počítače. I díky tomu se mohou priority, které určují, co bude zpracováno nejdříve, u systému Windows měnit. Například procesy, které běží na popředí, automaticky získávají prioritní zvýhodnění oproti procesům na pozadí. To je výhodné právě kvůli zaměření na uživatelskou zkušenost. Procesy se stejnou prioritou jsou brány jako sobě rovné a přepínání mezi nimi, které svým trváním ubírá na efektivitě, může být plánovačem oddalováno.

Naopak RTOS dává hlavní důraz na dodržení pevně daných priorit, které jsou definované i přímo v aplikacích a je jich na výběr zpravidla více. U takového systému nehrozí, že

by nevýznamný proces mohl shodou různých pravidel získat jednorázové navýšení priority, čímž by přerušil běh hlavní aplikace. [20]

3.2.1.2 Časování - systémový čas

Možná nejpodstatnějším důvodem, proč Windows nelze použít jako RTOS, je jeho nepřesné časování.

Rozlišení systémových hodin u systému Windows 7 je přibližně 15 milisekund. To znamená, že cokoliv bychom od aplikace chtěli dělat s vyšší frekvencí, se nám stejně vykoná nejdříve po 15 milisekundách. To je nepřijatelně dlouhá doba pro jakýkoliv RTOS.[21]

Systém Windows tedy bez podstatných úprav kernelu jako RTOS používat nelze a na tato omezení je při vývoji řídící aplikace nutné brát ohled.

4

Řídící aplikace v jazyce C#

Účelem řídící aplikace vyvíjené v rámci této diplomové práce je vylepšit autonomní schopnosti platformy obzvláště s využitím jeho obrazových senzorů. Aplikace běží na centrálním PC s operačním systémem Windows a řídí celkovou operaci platformy. Shromažďuje data ze všech modulů platformy a komunikuje s nimi. Aplikace není real-time jako jsou aplikace v jednotlivých modulech, a tak nezajišťuje například regulaci otáček, ale slouží pro řízení komplexnějších operací, jako např. celkového chování platformy.

Aplikace zajišťuje také uživatelské rozhraní jako běžná Windows aplikace. Slouží pro ovládání platformy, monitorování jejího stavu, ladění a konfiguraci běhu programu.

Jelikož hlavním účelem naší platformy je prezentace studentských prací fakulty, je také důležité aby aplikace plnila reprezentativní funkci. K tomu poslouží například vizualizace dat ze systému, zobrazení výstupu senzorů a kamer a responsivita uživatelského rozhraní. Pro reprezentativní účely je důležité také zajištění dostatečné míry bezpečnosti a spolehlivosti při interakci s vnějším prostředím.

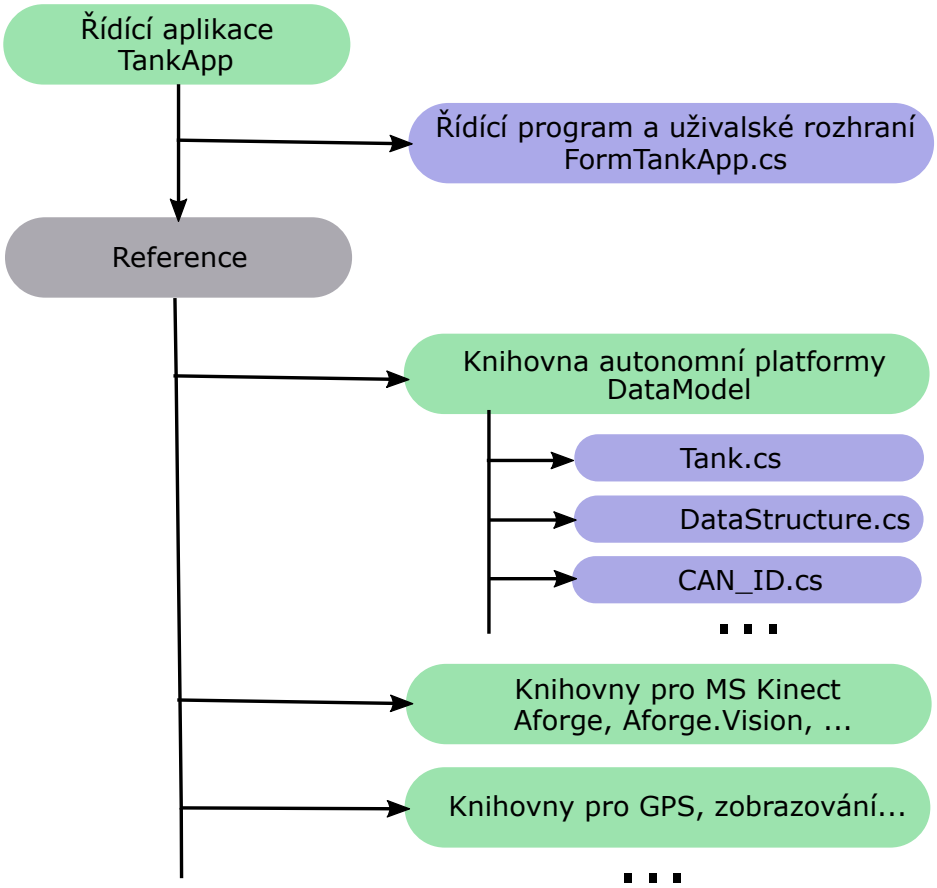
4.1 Vývojové prostředí Visual studio 2010

Řídící aplikace je vyvíjena ve vývojovém prostředí Microsoft Visual studio 2010 jako aplikace typu Windows Forms. Tato varianta umožňuje velmi rychlý návrh uživatelského rozhraní a dobrou kompatibilitu s veškerým softwarovým vybavením od firmy Microsoft. Vývojové prostředí je nainstalováno přímo na centrálním počítači a vyvíjenou aplikaci je tedy možné testovat v debugging módu přímo v provozu.

4.2 Struktura aplikace

Do samotné řídící aplikace jsou importované další knihovny jako moduly v jazyce C# zajišťující potřebné funkce pro obsluhu jednotlivých zařízení připojených k centrálnímu PC. Mezi tyto patří například knihovny pro MS Kinect nebo pro komunikaci s Gamepad joystickem.

Stejně je do aplikace importována také knihovna s funkcemi a proměnnými zajišťující ovládání a obsluhu HW tanku vyvinutého v rámci studentských prací. Tato knihovna je na rozdíl od ostatních knihoven neustále vyvíjena spolu s autonomní platformou.



Obr. 4.1: Struktura řídící aplikace

4.3 Použité knihovny

Pro ilustraci uvedu seznam použitých jmenných prostorů ve vývojové verzi aplikace s příklady jmen, která v projektu zajišťují:

- `System` - Int16, bool, Double
- `System.Collections.Generic` - list
- `System.Drawing` - color, Graphics
- `System.Drawing.Imaging` - PixelFormat, BitmapData
- `System.Linq` - using direktiva
- `System.Windows.Forms` - Form

- `System.Threading.Tasks` - Task
- `System.Globalization` - CultureInfo
- `AForge.Controls` - Joystick
- `Microsoft.Kinect` - KinectSensor, Skeleton
- `DataModel.DataModel` - Tank

4.3.1 Knihovny pro MS Kinect

K využití MS Kinect byl nainstalován kit Kinect For Windows SDK (Software Developement Kit) ve verzi 1.8, tedy nejvyšší podporovaná verze na OS Windows 7. Knihovny byly instalovány v rámci předchozí diplomové práce, ve které je také možné také najít další dokumentaci [1].

4.3.2 Ovládání HW platformy - knihovna DataModel

Nejnižší funkce a proměnné pro ovládání platformy jsou definované v knihovně *DataModel*. Byla vytvořena v rámci předchozí práce a je i nadále doplňována spolu s vývojem nové aplikace.

Nachází se zde třída Tank, ve které jsou uspořádány proměnné s daty o stavu platformy. Také se zde nachází například funkce pro obsluhu sběrnice CAN, příjem zpráv a aktualizaci dat ze senzorů.

4.3.2.1 Ovládání věže

Do knihovny byla přidána funkce pro obsluhu všech servo výstupů na jednotkách senzorů. Oproti předchozí verzi této funkce byl přidán parametr pro výběr jednotky a byl změněn formát vstupního parametru na pole. Následně byly přidány proměnné pro uchování informace o poloze věže do hlavní struktury a funkce pro aktualizaci polohy věže.

Totéž bylo uděláno i pro ovládání ramen a příslušenství namapováním příslušných proměnných do výstupů pro serva.

4.4 Optimalizace rychlosti řídící aplikace

Pro využití aplikace pro řízení procesů v reálném čase je kritické zachování dostatečné rychlosti odezvy systému. Počítač je vybaven dvoujádrovým procesorem o frekvenci 2,3 Ghz. To samo o sobě poskytuje opravdu dostatečné množství výkonu pro řízení několika výstupních veličin jako je rychlost pásů nebo pohyb věže. Výpočetní operace pro zpracování a zobrazování videa z MS Kinect ale mohou velmi snadno výpočetní výkon

spotřebovat celý. Podobně mohou chod aplikace zpomalovat i čekání na nepřerušitelné operace. Tyto problémy je při vývoji nutno adresovat příslušnými opatřeními.

Aplikace typu Windows Forms jsou stavěny na událostmi řízené architektuře, podobně je postaveno i chování celého systému Windows. V některých případech je ale klíčové využití paralelních vláken.

4.4.0.1 Využití paralelních vláken

Přestože většina úkonů v aplikaci Windows Forms je založena na zpracovávání událostí, jsou operace, které mohou být efektivněji prováděny ve vlastních vláknech. K tomu se používá modul System.Threading.

K takovým se řadí procesy čekání. Například čekání na převodník USB/CAN. Funkce obsluhující tyto procesy je výhodné vyvolávat ve vlastních vláknech. Při používání paralelních vláken je velmi důležité využití thread-safe metod. To zabrání současnému čtení nebo zápisu dat do stejné proměnné z různých vláken.

4.4.0.2 Využití událostí

Pro procesy které nevyhnutelně spotřebovávají výpočetní výkon není v našem případě, při využití systému Windows s dvoujádrovým procesorem, příliš výhodné zakládat paralelní vlákna. Systém Windows práci na procesech střídá dle vlastního uvážení a obsluhuje také své vlastní systémové procesy. Nutit aplikaci vytvářet další vlákna by mohlo mít na celkovou rychlost i negativní vliv.

Pro urychlení běhu aplikace je v tomto případě vhodné omezení počtu událostí vyvolávající výpočetně náročné procesy. Obzvláště při zpracovávání obrazu je klíčové omezit redundantní výpočty například omezením obnovovací frekvence.

Při regulování procesů po sběrnici CAN je také výhodné přizpůsobit rychlost hlavní řídící smyčky rychlosti odesílání a příjmu CAN zpráv. Jinými slovy je zbytečné analyzovat polohu sledované postavy stokrát častěji, než je možné odesílat zprávy po sběrnici CAN, k ovládání platformy.

4.5 Vývojový diagram

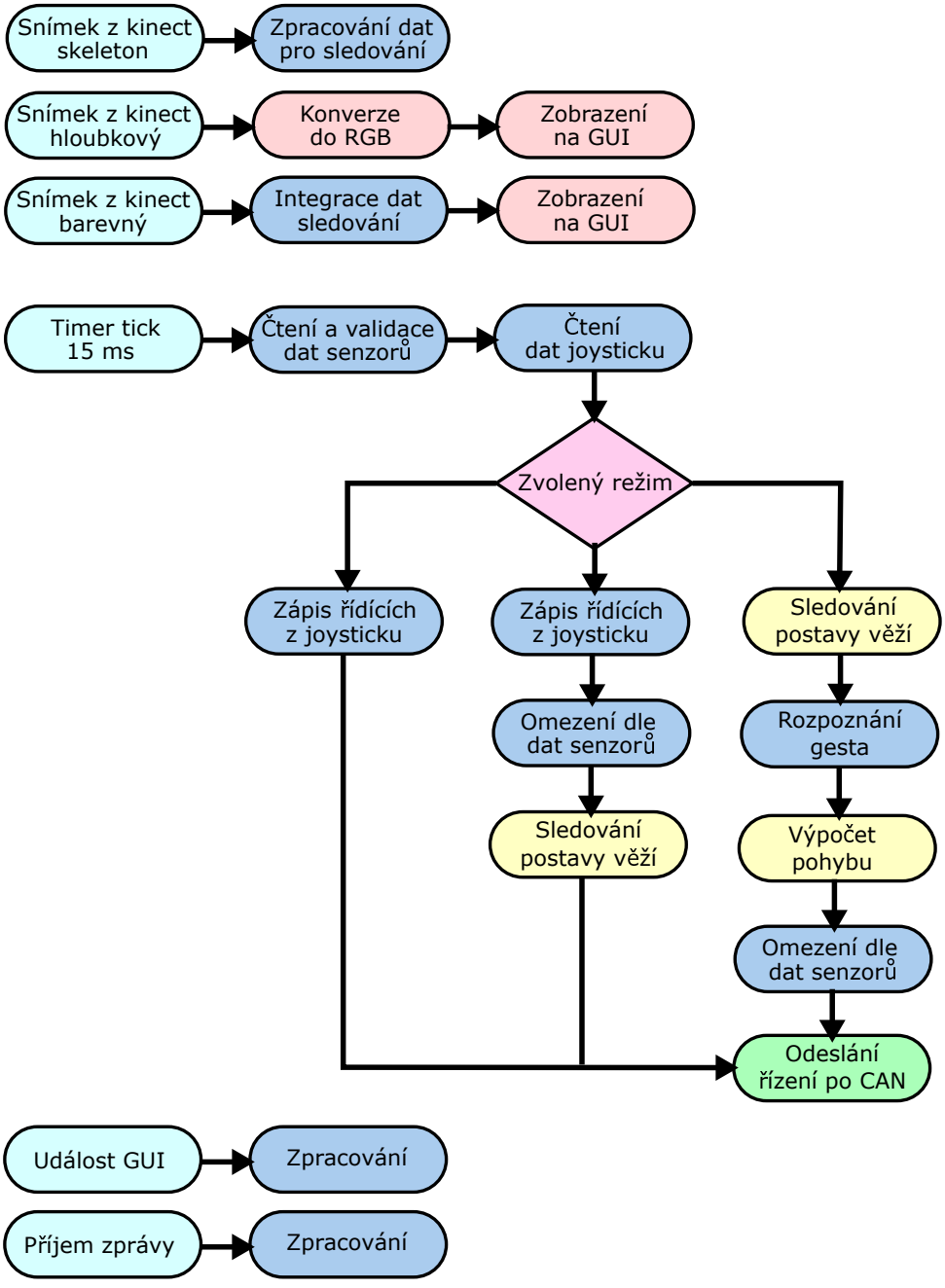
Na zjednodušeném vývojovém diagramu na obrázku jsou naznačeny všechny významné prvky řídící aplikace. Aplikace je postavena na využití událostí, které jsou vyvolávány několika různými zdroji.

Příjmem dat ze zařízení MS Kinect se spouští algoritmy pro zpracování obrazu a jeho vykreslování na grafickém rozhraní. Zde je získána informace o poloze všech sledovaných postav s identifikátory pro jejich rozlišení.

Události vyvolané příjmem zpráv ze sběrnice CAN a dalšími periferiemi jsou po konfiguraci zpracovávané knihovnou DataModel.

Události grafického rozhraní slouží například pro změnu nastavení, pro přepnutí režimu, nebo jiné úpravy chování řídícího algoritmu.

Událostí vyvolanou časovačem s intervalem 15 ms je spouštěn řídící algoritmus pro ovládání platformy. Dle zvoleného režimu se určí jak velká část řízení bude ponechána na operátorovi a co bude platforma regulovat sama. Pro sledování postavy věží nebo celou platformou je využito PID regulace a rozpoznávání gest.



Obr. 4.2: Zjednodušený vývojový diagram řídící aplikace

4.6 Softwarová bezpečnostní opatření

Protože řídící aplikace běží na systému Windows, známém svou občasnou nestabilitou, je užitečné využít všechny možnosti a data, která je z platformy možné získat, pro zajištění určité míry bezpečnosti, aby nedošlo ke škodám jako u první verze platformy, která byla po havárii silně poškozena [3].

K detekci překážek slouží senzory přiblížení a pro detekci kolize jsou použita také data z akcelerometru. Tyto metody ošetřují bezpečnost při správném běhu aplikace, ale pro případ jejím selhání je nutné zajistit bezpečnost na nižší úrovni.

4.6.1 Watchdog na sběrnici CAN

Pro kontrolu běhu řídící aplikace se používá takzvaný watchdog, neboli hlídací pes. Funguje na principu pravidelného odesílání zpráv, jejichž příjem ujišťuje příjemce o aktivitě odesílatele. Příjemce je vybaven časovým odpočtem, který je resetován při každém příjmu takové zprávy. Při výpadku zpráv dojde k vypršení tohoto časového intervalu a tehdy nastane chyba.

Toto je kritické pro jednotku motorů, která by bez informace o pádu řídící aplikace mohla nechat platformu ujet pryč rychlostí, kterou přijala po sběrnici CAN jako poslední.

4.6.2 Open-loop regulace motorů

Pro zabezpečení při případném hardwarovém selhání měření otáček u jednotky motorů je možné využít speciálního formátu CAN zprávy s požadovanou rychlostí pro řízení v otevřené smyčce. V případě chyby měření otáček, například při překlesání vodiče od senzoru, můžeme kladnou zpětnou vazbu softwarově odpojit a platformu zastavit.

V tomto režimu je možné motory i řídit, a s využitím jiné zpětné vazby, například polohou z GPS nebo MS Kinect, i teoreticky regulovat.

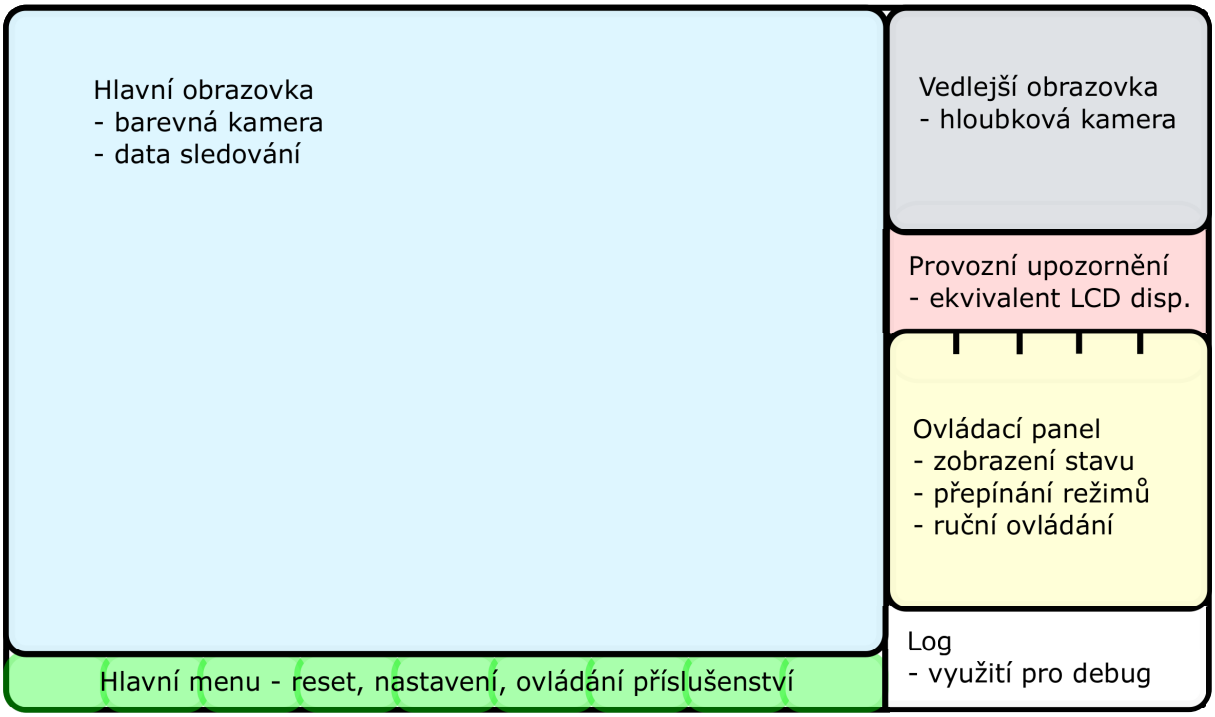
4.7 Uživatelské rozhraní aplikace

Uživatelské rozhraní aplikace zahrnuje kromě samotného grafického rozhraní také další způsoby komunikace uživatele s řídící aplikací. K těmto patří především herní konzole Gamepad s joysticky a také zařízení MS Kinect se svými schopnostmi rozpoznávání postav a gest.

4.7.1 Grafické rozhraní

Grafické rozhraní aplikace je možné používat pomocí vzdálené plochy nebo pomocí připojení monitoru a myši. Jeho účelem je přehledně zobrazit podstatné informace a umožnit intuitivní ovládání. Důležitý je také grafický návrh který by měl být při prezentacích pochopitelný na první pohled .

Nejpoutavější ze získávaných dat je výstup z kamer MS Kinect. Na druhou stranu není nutné vždy zobrazovat všechna měřená data, která se při statické prezentaci platformy téměř nemění. Jde například o data z jednotlivých os akcelerometrů a gyroskopů.



Obr. 4.3: Rozložení prvků grafického rozhraní

4.7.1.1 Návrh grafického rozhraní

Grafické rozhraní je navrženo ve Windows Forms designeru. Prostor okna je dělen na buňky s relativní velikostí. V pravé části jsou zobrazeny provozní informace a přepínání záložek s režimy provozu. Ve spodní části je lišta s hlavním menu pro vyvolání nastavení a dalších funkcí využitelných při všech režimech provozu.

Při návrhu byl přiřazen největší prostor výstupu z barevné kamery MS Kinect. V tomto obraze je zobrazován také výstup sledování postavy, jako je např. sledovaná kostra (anglicky skeleton) a identifikátory jednotlivých postav. Zobrazen je i výstup z hloubkové kamery převedený do černobílého spektra.

4.7.1.2 Nastavení

Do aplikace bylo přidáno nastavení podporované rozhraním .NET Framework, které zůstává uloženo i po ukončení aplikace. Jeho proměnné jsou definované přímo ve vlastnostech projektu pod záložkou *settings*.

Pro úpravu nastavení bylo přidáno další okno aplikace, kde jsou umístěny jeho ovládací prvky. Důležitá nastavení jsou následující:

- `PID věže` - konstanty P, I, D a celkový zisk (Gain) regulátorů pro sledování postavy pomocí věže, horizontální a vertikální směr
- `PID pásů` - konstanty P, I, D a rychlostní limity sledování postavy pomocí platformy
- `limitní hodnoty senzorů přiblížení` - minimální prostor kolem platformy pro pohyb vpřed, vzad a pro otáčení
- `limitní hodnoty akcelerometru` - hranice hodnot přetížení ve třech osách pro detekci kolize
- `Povolení senzorů` - možnost ignorovat vybraný senzor přiblížení pokud vrací nesprávné hodnoty a brání v provozu
- `Open-loop` - možnost přepnout do "nouzového" režimu řízení motorů v otevřené smyčce při selhání senzorů otáček
- `Povolení laseru` - možnost zakázat laserové ukazovátko

4.7.2 Ovládání pomocí Gamepad konzole

Během provozu platformy v terénu se k ovládání používá ovladač Gamepad joystick. Jeho dosah je v řádech metrů, což slouží i jako ujištění, že operátor bude v dosahu platformy a bude na provoz dohlížet. Aplikace je přizpůsobená tak, aby bylo možné pomocí ovladače dosáhnout na všechny podstatné ovládací prvky jako v jejím grafickém rozhraní. Samozřejmě nepočítáme množství parametrů v okně nastavení.

Indexování tlačítek na ovladači je popsáno na obrázku 4.4. Čísla odpovídají číslům ve třídě Joystick.

Obr. 4.4: Indexování ovládacích prvků konzole Gamepad joystick

- `Osy X/Y` - Ovládání pásů
- `Osy R/U` - Ovládání věže
- `1 (A), 2 (B), 3 (X), 4 (Y)` - Ovládání specifických funkcí dle aktivního režimu
- `5, 6` - Ovládání doplňků věže - světlo LED, laser
- `7 (back), 8 (start)` - Přepínání režimů manuální - semiauto - autonomní
- `M (mode)` - Přepínání mezi tlačítky X/Y a joytickem X/Y na konzoli
- `V (vibration)` - Test vibrací konzole

4.7.2.1 Přepočet joysticku na diferenciální řízení

Diferenciální řízení, jako to aplikované u platformy, obnáší ovládání rychlosti a směru pohybu pomocí dvou pásů. V předchozí aplikaci bylo ovládání pomocí Gamepad také diferenciální, použitím dvou joysticků, každého pro jeden pás. Pro uspoření jednoho joysticku na ovladači byl vybrán následující algoritmus pro přepočet dvou souřadnic joysticku na diferenciální řízení.

```
//ukázka kódu pro přepočet joysticku na diferencíální řízení

double X = joystick.State().XAxis;
double Y = -joystick.State().YAxis;

double V = (1 + Math.Abs(X)) * Y + Y;
double W = (1 + Math.Abs(Y)) * X + X;

tank.data.tracks.left = (V + W) / 2;
tank.data.tracks.right = (V - W) / 2;
```

Tyto jednoduché rovnice byly získány a upraveny z blogu jistého amerického stavitele bojových robotů [23].

V principu je pro využití joysticku nutné softwarově otočit hodnoty os XY o 45°. To by bylo možné uskutečnit také namapováním těchto souřadnic do polárních souřadnic kruhu pomocí některé z mnoha známých mapovacích metod jako například Schwarz-Christoffelova, nebo mapování pomocí eliptických sítí. Polární souřadnice by bylo možné následně otočit o potřebný úhel. Nakonec by byly namapovány zpět do čtvercových kartézských souřadnic joysticku. Takto vzniklé rovnice by měly podobný výsledek jako námi použité, ale byly by výrazně komplexnější.

4.8 Režimy provozu

Aplikace běží ve třech režimech dle stupňů autonomie.

4.8.1 Manuální režim

V manuálním režimu je operátorovi dána maximální autorita nad ovládáním platformy a jsou ignorovány jak senzory přiblížení tak detekce kolize. Pouze v manuálním režimu je také možné aktivovat laserové ukazovátko, pokud je povoleno v nastavení.

4.8.2 Semiautonomní režim

V semiautonomním režimu je operátorovi ponechána kontrola nad pásy, ale pohyb věže je převážně řízen automaticky sledováním postav. Věž sleduje první postavu, kterou detekuje, a při její ztrátě se vrátí do základní polohy, kterou je možné ovládat joystickem.

Ovládání věže zajišťuje PID regulátor, jehož zpětnou vazbou je odchylka polohy postavy od středu záběru kamery. Správné naladění tohoto regulátoru je kritické pro funkční sledování postavy. Software pro detekci postavy MS Kinect není navržený pro trasování postav pohybující se kamerou, a proto je náchylný na prudké pohyby a na rozmazání záběru kamery při oscilacích regulátoru.

4.8.2.1 PID regulátor v jazyce C#

Princip fungování regulátoru zůstává stejný jako v jednotce motorů v jazyce C. Ten jsme vysvětlili ve druhé kapitole. Oproti regulátoru v jednotce motorů je metoda univerzální pro regulování více procesů. Ve struktuře *PIDregulator* jsou uložené potřebné proměnné a také konstanty PID. Díky využití objektového programování je kód výrazně kratší:

```
// metoda pro PID regulaci
private double PIDreg(double error, ref PIDregulator PID, int min, int max)
{
    double regValue;                            // výstupní veličina

    PID.Isum += error;                          // aktualizace sumy chyby
    double IsumMax = (max - min) / PID.Iconst; // ošetření limitů
    if (PID.Isum > IsumMax) PID.Isum = IsumMax;
    if (PID.Isum < -IsumMax) PID.Isum = -IsumMax;

    regValue =  (error * PID.Pconst)            // hlavní rovnice
              + (PID.Isum * PID.Iconst)
              + ((error - PID.LastVal) * PID.Dconst);

    PID.LastVal = error;                        // uložení chyby

    regValue += (max - min) / 2;                // ošetření výstupu
    if (regValue > max) regValue = max;
    if (regValue < min) regValue = min;

    return regValue;
}
```

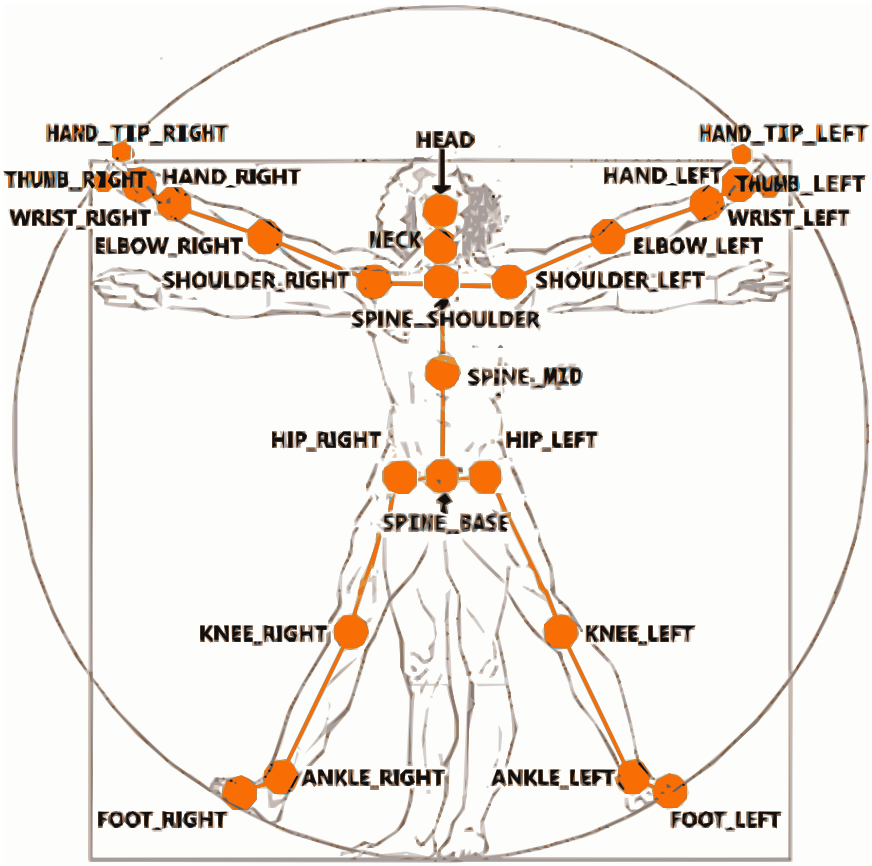
Regulátor je použitý i pro návrat věže do základní polohy po ztrátě sledované postavy. Zpětnou vazbou je odchylka od dané polohy a výsledkem je plynulejší pohyb, než jaký by nastal při skokovém centrování.

4.8.3 Autonomní režim

V autonomním režimu platforma sleduje postavu stejně jako v semiautonomním režimu a po určeném gestu zahajuje sledování. K definici gesta (zvednutí pravé ruky) je využito souřadnic jednotlivých kloubů tak, jak jsou popsány pomocí knihoven pro MS Kinect.

4.8.3.1 Rozpoznání postavy a gest pomocí MS Kinect

Rozpoznávání gest bylo implementováno již v předchozí práci [1]. Jelikož má naše verze MS Kinect již 10 let po vydání, není už bohužel na oficiálních stránkách od Microsoft podporována a dokumentaci s příklady je nutné získávat ze třetích stran. Zjednodušený popis zpracování dat o postavě se pokusím shrnout do pár odstavců.



Obr. 4.5: Popis postavy dle MS Kinect [24]

Souřadnice kloubů postavy zařízení odesílá již v metrech, přepočítané na reálný prostor. Souřadnice X odpovídá vzdálenosti vlevo / vpravo od kamery, souřadnice Y je nahoru / dolů a souřadnice Z je přímá vzdálenost od kamery. Pokud tedy postava stojí dva metry

od kamery kolmo k objektivu, který je zároveň ve výšce kolen, budou se její souřadnice pohybovat kolem [x=0, y=1, z=2].

Detekce gest spočívá v porovnání polohy jednotlivých prvků kostry postavy viz obrázek 4.5. Tato data připravuje již přímo HW KS Kinect a k jejich zpracování je vytvořena událost, ve které je možno detekovat gesto a následně povolit pohyb platformy.

4.8.3.2 Pohyb platformy

V předchozí verzi aplikace bylo řízení motorů řešeno sadou podmínek, pevných hranic a sekvencí při ztrátě sledované postavy. Díky lepšímu umístění kamery, je nyní možné se více věnovat tomuto režimu. Pro plynulejší ovládání je nutné sladit pohyb věže a pohyb platformy pro ještě nižší šanci ztráty identifikované postavy. Pro ovládání pásu byl navržen PID regulátor s laděním v nastavení, viz kapitola 4.7.1.2.

5

Závěr

Závěrem bych chtěl především shrnout dosažené výsledky a adresovat zadání práce. Následně bych rád přiblížil kontext, v jakém práce vznikala, a jaký vývoj práci ještě čeká. V neposlední řadě zmíním vlastní poznatky, které by se mohly při budoucím vývoji hodit.

Zadáním práce bylo vytvořit řídící aplikaci pro autonomní pásovou platformu, tedy vývoj software v jazycích C/C# . Velká část práce se ale věnuje také úpravám hardwaru, který významně omezoval použitelnost platformy pro účely této práce.

Během řešení prvního bodu zadání, tedy analýzy stavu platformy s ohledem na softwarové vybavení, bylo přistoupeno k řešení dvou základních problému. Prvním z nich byla velmi pomalá odezva systému na požadavek změny rychlosti. Při řešení tohoto problému bylo nejdříve umožněno napínání řetězu, což ale velký vliv nemělo. Nakonec byla zásadně upravena jednotka motorů. Její problémová výkonová část byla nahrazena výkonnými modelářskými budiči a její firmware byl převeden do projektu v novější verzi vývojového studia. Původní, nepřehledný fuzzy regulátor byl nahrazen konvenčním PID regulátorem a byly vyrobeny nové senzory otáček, které umožňují i detekci směru. Tím byla odezva systému snížena na zlomek té původní.

Druhým problémem bylo nevhodné umístění zařízení MS Kinect na platformě. To bránilo efektivnímu vývoji aplikace pro řízení pomocí obrazových senzorů, jak bylo uvedeno ve druhém bodu zadání. Proto byla navržena věž pro vyvýšení zařízení MS Kinect a pro umožnění jeho natáčení. První prototyp věže byl pro otestování serv a točny sestaven ze stavebnice Merkur. Následně byl vytvořen model věže, který byl vytisknut na 3D tiskárně a v poslední verzi doplněn vylepšeným otáčecím mechanismem. Nakonec byla přidána také jednoduchá ramena s příslušenstvím, která rozšiřují interaktivní schopnosti platformy. Věž nyní může zařízením MS Kinect otáčet v rozsahu necelých 270° horizontálně a 135° vertikálně.

V době dokončování práce propukla světová pandemie a tak velká část práce vznikala mimo zázemí univerzity a jen s omezenou mírou konzultací. Dalším problémem, který se projevil a bude nutné jej v budoucnu řešit, bylo padání počítače po vícedenním stání. Nejdříve se problém podařilo vyřešit novým ventilátorem, ale pak se problém ještě krátkodobě vracel.

Z časových důvodů nebyl zcela dokončen třetí bod zadání. Na finální podobě řídící aplikace se stále ještě pracuje. Hlavní motivací je nyní závazek vůči následující diplomové práci, kterou dokončuje příští rok můj spolužák a přítel Ondřej Malena. Ta se bude týkat dlouho postrádaného modulu správy napájení, který vyřeší neustálou potřebu externí nabíječky.

V současné verzi řídící aplikace se povedlo úspěšně použít dále rozšiřovanou knihovnu z předchozí práce [1], která slouží k ovládání jednotlivých periferií platformy. Byl vytvořen a úspěšně otestován režim pro autonomní sledování postavy pomocí věže, který k jejímu plynulému řízení využívá PID regulace. Následně bylo přidáno nastavení pro ladění potřebných konstant uvnitř aplikace. Autonomní pohyb celé platformy byl zatím implementován jen s podobným algoritmem, jaký byl použit u předchozí verze aplikace. K úplnému dokončení nové verze aplikace tedy zbývá doplnění nového algoritmu i pro autonomní pohyb, rozšíření potřebných uživatelských nastavení a finalizace podoby grafického rozhraní.

Na úplný závěr bych jen dodal, že číslo pět žije a já věřím, že s novou věží a aplikací bude platforma dobře sloužit k prezentování studentských prací v následujících letech.

Literatura

[1] Lufinka O.: *Využití ovladače Kinect k řízení mobilní platformy* Plzeň, 2015. Diplomová práce. ZČU.

[2] Lufinka O.: *Tank*[online]. Poslední změna 4. 10. 2019, 09:24 [cit. 1. 5. 2020]. Dostupné z:
http://projekty.fel.zcu.cz/index.php/Tank

[3] Pušman L.: *Řídicí systém systému mobilní platformy* Plzeň, 2011. Diplomová práce. ZČU.

[4] Zvonař F.: *Generátor DCC signálu pro modelovou železnici* Plzeň, 2018. Bakalářská práce. ZČU.

[5] Copyright Warwick Control Technologies: *CAN Access & Collisions* [online]. Poslední změna 24. 1. 2017 [cit. 10. 5. 2018]. Dostupné z:
https://manual.xanalyser.com/CAN%20Access%20&%20Collisions.html

[6] Domanský s.r.o. *Elektronický obousměrný regulátor 60 A řady QuicRun* [online]. [cit. 24. 5. 2020]. Dostupné z:
https://profimodel.cz/cs/auto-lodni/62329-quicrun-1060-v2-6938994413084.html?gclid=CjwKCAjw˙uDsBRAMEiwAaFiHa2YRzWr59hqzdZxdo˙h5jeCLv6rp4PRfYnQTwB6s4CLXSqrX8D˙4mBoCQP0QAvD˙BwE

[7] Wikipedia: *Servo control* [online]. Poslední změna 12. 5. 2018, 5:21 (UTC) [cit. 10. 5. 2020]. Dostupné z:
https://en.wikipedia.org/wiki/Servo˙control

[8] Ben Ryon *'Kinect for Xbox 360' is Official Name of Microsoft's Controller-Free Game Device* [online]. Poslední změna 13. 6. 2010 [cit. 9. 5. 2020]. Dostupné z:
https://news.microsoft.com/2010/06/13/kinect-for-xbox-360-is-official-name-of-microsofts-controller-free-game-device/

[9] Janne *Documentation: Kinect for Windows 1 / Kinect for Xbox 360 (models 1414 & 1473)* [online]. Poslední změna 27. 10. 2015 [cit. 9. 5. 2020]. Dostupné z:
https://forum.ni-mate.com/t/documentation-kinect-for-windows-1-kinect-for-xbox-360-models-1414-1473/288

[10] blogger: TheFrenchLeaf *Kinect camera* [online]. Poslední změna 5. 11. 2010 [cit. 9. 6. 2020]. Dostupné z: http://smallideasforabigworld.blogspot.com/2010/11/kinect-camera.html

[11] A.T. Shop s.r.o. *Točna pod TV 70x70 mm, nosnost 100 kg* [online]. Poslední změna 2020 [cit. 9. 6. 2020]. Dostupné z: https://www.onpira.cz/zbozi/tocna-pod-tv-70x70-mm–nosnost-100-kg/?gclid=Cj0KCQjwtLT1BRD9ARIsAMH3BtXVSCUaqHrQ27gHjz1IwvfjZJ4qd5wCrjmhR0n0ZUYqnK2RJ70Tel4aAv8nEALw˙wcB

[12] Manos Antoniou *Predicting the future popularity of programming languages* [online]. Poslední změna 14. 9. 2019 [cit. 11. 6. 2020]. Dostupné z: https://www.manosantoniou.com/post/predicting-the-future-popularity-of-programming-languages/

[13] Bikesh Srivastava *History Of C# Programming Language* [online]. Poslední změna 30. 1. 2017 [cit. 11. 6. 2020]. Dostupné z: https://www.c-sharpcorner.com/blogs/history-of-c-sharp-programming-language

[14] David Čápka *Úvod do C# a .NET frameworku* [online]. Poslední změna 8. 6. 2020, 17:30 [cit. 11. 6. 2020]. Dostupné z: https://www.itnetwork.cz/csharp/zaklady/c-sharp-tutorial-uvod-do-jazyka-a-dot-net-framework

[15] Microsoft *Průvodce jazykem C# : Úvod* [online]. Poslední změna 1. 7. 2017 [cit. 11. 6. 2020]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/csharp/language-reference/language-specification/introduction

[16] Microsoft *Objektově orientované programování (C#)* [online]. Poslední změna 13. 5. 2020 [cit. 11. 6. 2020]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/csharp/programming-guide/concepts/object-oriented-programming

[17] Microsoft *Výjimky a jejich zpracování (Průvodce programováním v C#)* [online]. Poslední změna 20. 7. 2015 [cit. 12. 6. 2020]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/csharp/programming-guide/exceptions/

[18] w3schools *C# Syntax)* [online]. Poslední změna 2020 [cit. 12. 6. 2020]. Dostupné z: https://www.w3schools.com/cs/cs˙syntax.asp

[19] circuitstoday: jojo *GPOS versus RTOS for an Embedded System)* [online]. Poslední změna 12. 6. 2012 [cit. 13. 6. 2020]. Dostupné z: https://www.circuitstoday.com/gpos-versus-rtos-for-an-embedded-system

[20] The Joseph M. Newcomer Co. *Time is the Simplest thing...)* [online]. Poslední změna 14. 5. 2011 [cit. 13. 6. 2020]. Dostupné z: http://www.flounder.com/time.htm

[21] Microsoft *Timer.Interval Vlastnost* [online]. [cit. 13. 6. 2020]. Dostupné z: https://docs.microsoft.com/cs-cz/dotnet/api/system.timers.timer.interval?view=netframework-3.5

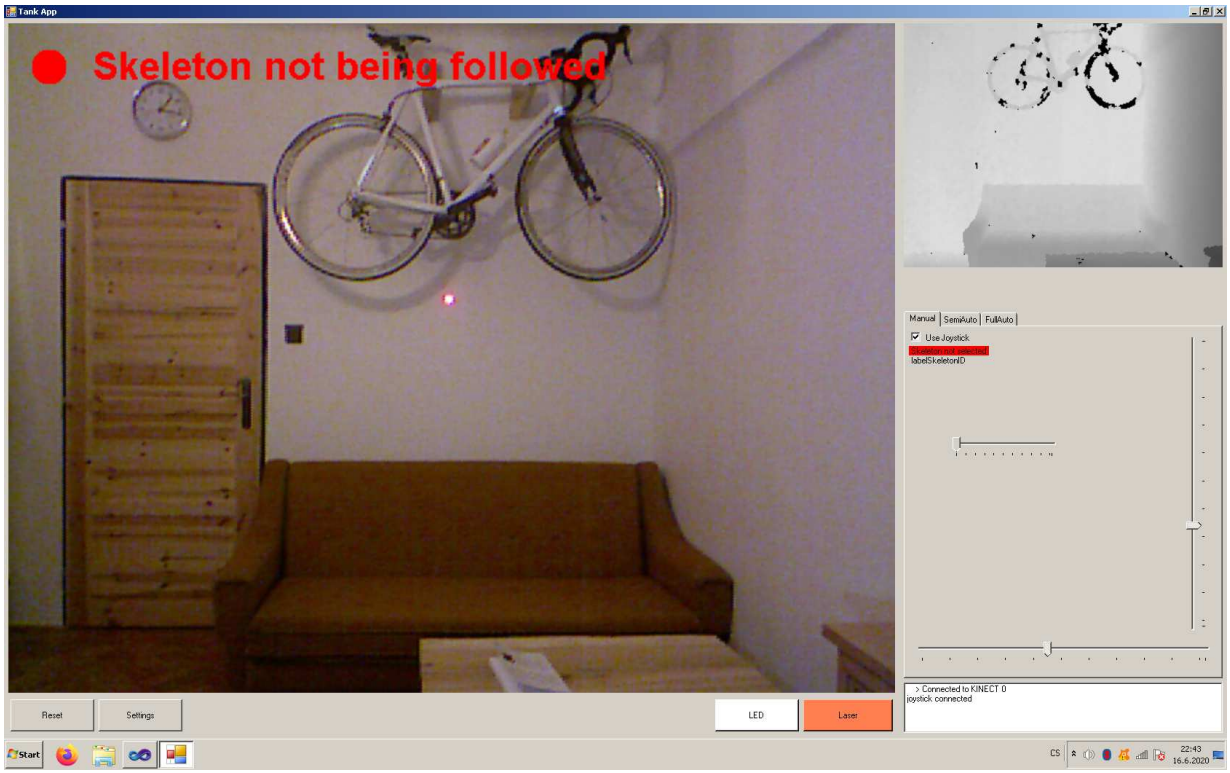
[22] Rod Elliott *Hobby Servos, ESCs And Tachometers* [online]. Poslední změna 1. 2018 [cit. 13. 6. 2020]. Dostupné z: https://sound-au.com/articles/servos.htm

[23] Mauser *Joystick* [online]. [cit. 16. 6. 2020]. Dostupné z: http://home.kendra.com/mauser/Joystick.html

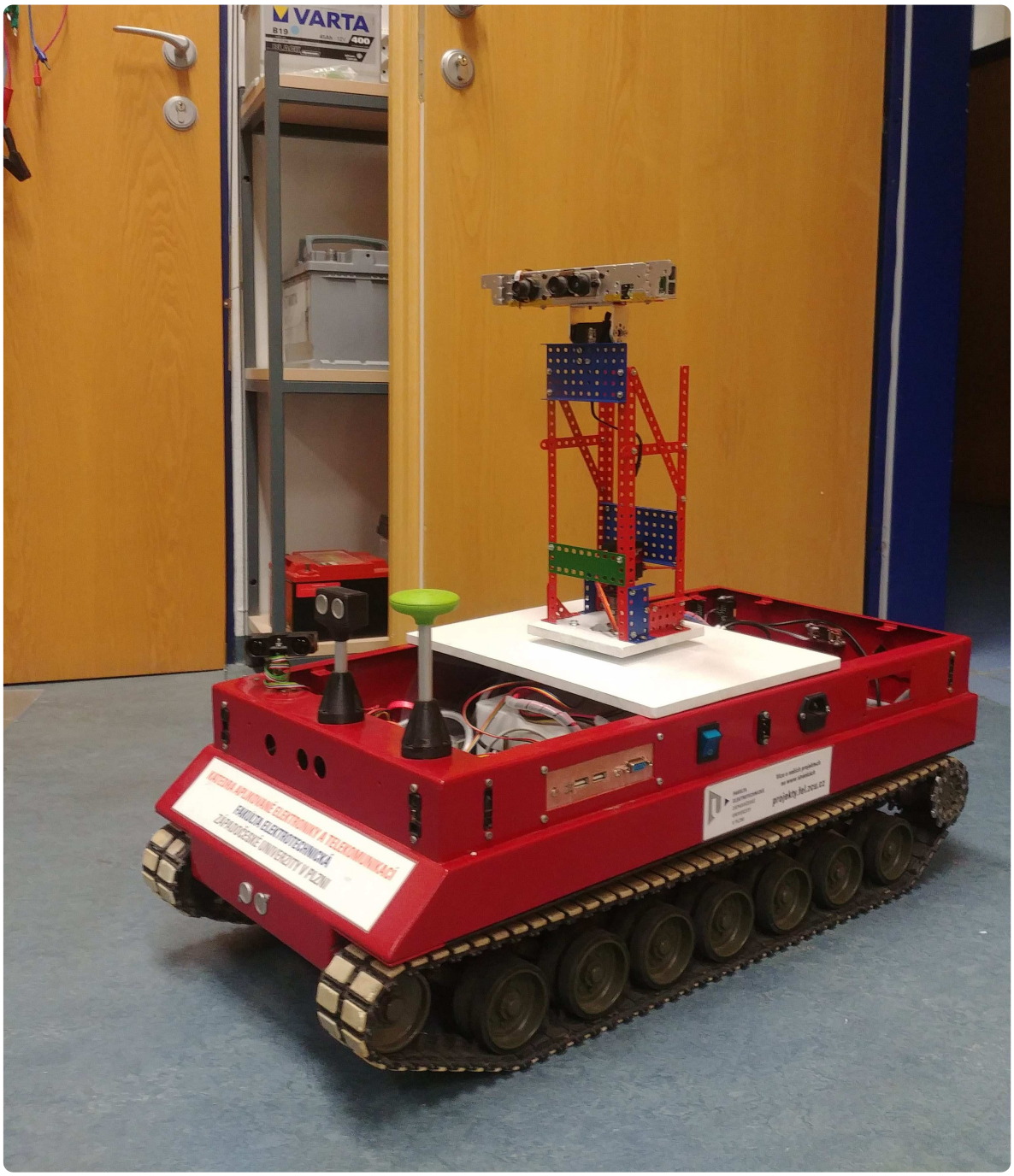
[24] Lisa Jamhoury *Understanding Kinect V2 Joints and Coordinate System* [online]. Poslední změna 23. 7. 2018 [cit. 16. 6. 2020]. Dostupné z: https://medium.com/@lisajamhoury/understanding-kinect-v2-joints-and-coordinate-system-4f4b90b9df16

Příloha A

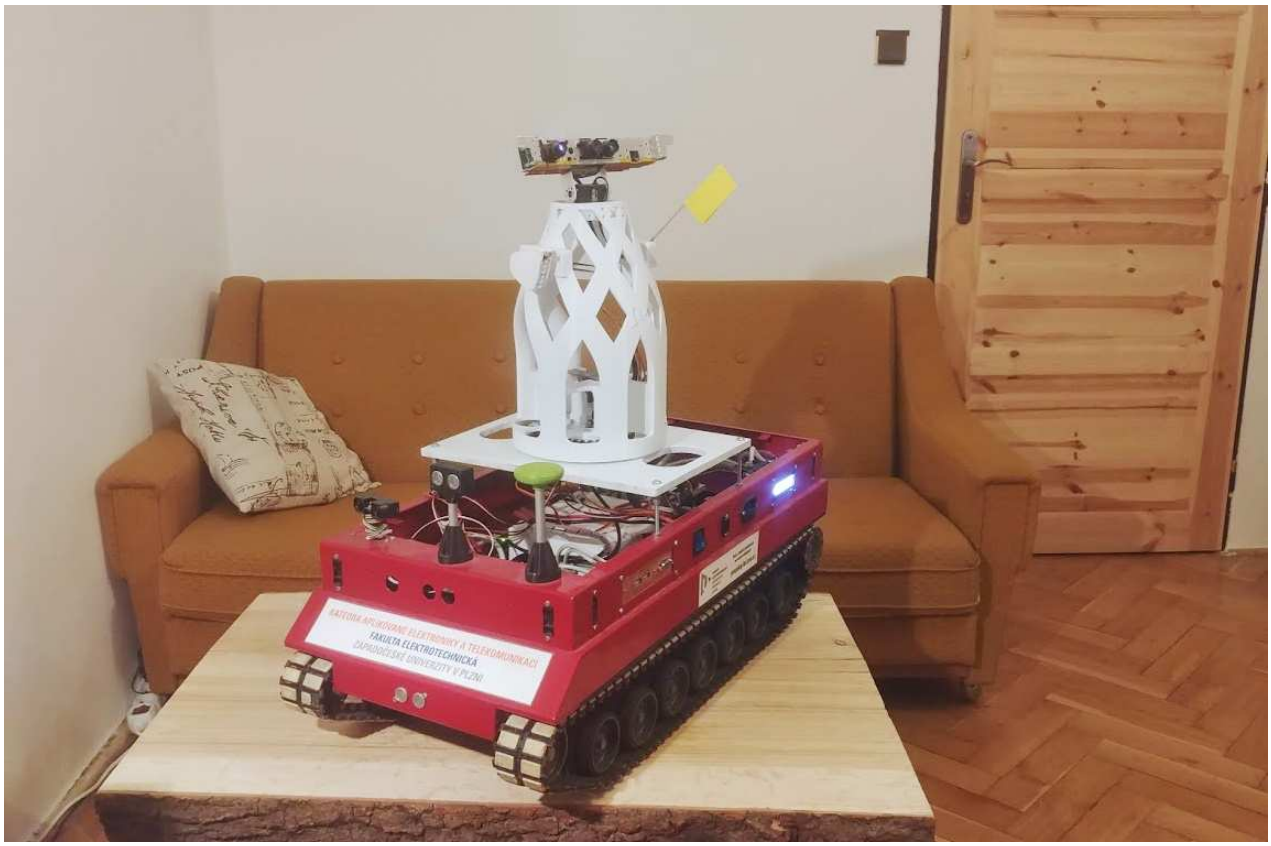
Fotografie



Obr. A.1: Snímek obrazovky vyvíjené aplikace



Obr. A.2: Platforma s prototypem věže z Merkuru



Obr. A.3: Aktuální fotka platformy