TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Komunikace s průmyslovým robotem KUKA

Bakalářská práce

Studijní program: B0714A270001 Mechatronika
Autor práce: **David Kutra**
Vedoucí práce: Ing. Tomáš Martinec, Ph.D.
Ústav mechatroniky a technické informatiky

Liberec 2022

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

Komunikace s průmyslovým robotem KUKA

Jméno a příjmení: **David Kutra**
Osobní číslo: M19000089
Studijní program: B0714A270001 Mechatronika
Zadávající katedra: Ústav mechatroniky a technické informatiky
Akademický rok: **2021/2022**

Zásady pro vypracování:

1. Seznamte se s problematikou průmyslových robotů, zaměřte se zejména na řídící systém robotů KUKA a jejich možnostmi komunikace s nějakým nadřazeným systémem.
2. Seznamte se s možnostmi programu KUKAVARPROXY a knihovny openshowvar pro komunikaci s roboty KUKA.
3. Vytvořte vlastní implementaci protokolu pro komunikaci se TCP serverem KUKAVARPROXY, vytvořte pro ní demonstrační aplikaci a otestujte možnosti a limity tohoto způsobu komunikace.
4. Navrhněte a otestujte možnosti, jak by bylo možné s využitím KUKAVARPROXY definovat a případně i ovlivňovat trajektorii robota z nadřazeného PC.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30–40 stran
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] Novotný, F., Horák, M., Konstrukce robotů, Technická univerzita v Liberci, Liberec, 2015A.
[2] KUKA System Software 8.3, Návod k obsluze a programování pro konečné uživatele, KUKA Roboter GmbH.
[3] Mesmar, M. B. Younis, T. Wruetz and R. Biesenbach, „Remote Control Package for Kuka Robots using MATLAB," *2020 17th International Multi-Conference on Systems, Signals & Devices (SSD)*, 2020, pp. 842-847, doi: 10.1109/SSD49366.2020.9364172.

Vedoucí práce:	Ing. Tomáš Martinec, Ph.D. Ústav mechatroniky a technické informatiky
Datum zadání práce:	12. října 2021
Předpokládaný termín odevzdání:	16. května 2022

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

doc. Ing. Josef Černohorský, Ph.D.
vedoucí ústavu

V Liberci dne 12. října 2021

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

15. května 2022 David Kutra

Komunikace s průmyslovým robotem KUKA

Abstrakt

Tato práce popisuje problematiku komunikace mezi externím počítačem a serverem průmyslového robota KUKA. Práce obsahuje podrobný popis programu, který tuto komunikaci zajišťuje. Porovnáním základních způsobů komunikace s robotem KUKA se práce dostává k vlastní implementaci zkoumaných metod. Práce zahrnuje detailní popis jednotlivých formulářových prvků, které jsou nezbytné pro ovládání aplikace.

Jádrem praktické části je návrh, vytvoření a testování demonstrační aplikace pro komunikaci s robotem. Práce odhaluje limity a nedostatky použitých metod v návrhu vytvořené aplikace. Zhodnocením limitů je určeno místo pro samotnou implementaci vytvořené aplikace ve výrobním řetězci. Praktická část propojuje poznatky nabyté v části teoretické s jejich implementací do vytvořené aplikace.

Klíčová slova: Robot, TCP/IP komunikace, KUKA, KRL Ethernet, KukavarProxy

Abstract

This thesis describes the issue of communication between external counter and server of the KUKA industrial robot. The work includes a detailed description of the program that provides this communication. Comparing of basic ways of communicating with the KUKA robot, thesis leads to the actual implementation of the investigated methods. The work includes detailed description of individual form elements that are necessary for controlling of the application.

The core of the practical part is the design, creation and testing of the demonstrative application for communication with the robot. The work reveals the limits and shortcomings of the methods used in the design of the created application. Evaluation of limits is used to determine the place for the actual implementation of the formed applications in the production chain. The practical part connects the experiences acquired in the theoretical part with their implementation into the created application.

Keywords: Robot, TCP/IP communication, KUKA, KRL Ethernet, KukavarProxy

Poděkování

Rád bych tímto poděkoval Ing. Tomáši Martincovi, Ph.D za vytvoření zadání, odborné vedení práce, kterou mi v průběhu zpracovaní věnoval. Jeho trpělivost a ochota vedla k úspěšnému zpracování této bakalářské práce.

Obsah

Seznam obrázků

Seznam tabulek

Seznam zkratek

ARPANET	**Advanced Research Projects Agency NETwork** - počítačová síť, jež byla zárodkem Internetu
EKI	**Ethernet KUKA Interface** - rozhraní Ethernet KRL umožnující komunikaci
FIFO	**First In First** - paměť, typu fronty
IDE	**Integrated Development Enviroment** - integrované vývojové prostředí
IP	**Internet Protocol** - protokol síťové vrstvy
ISO	**International Organization for Standardization** - Mezinárodní organizace pro normalizaci
KRL	**KUKA Robot Language** - programovací jazyk pro roboty KUKA
KUKA	**Keller und Knappich Augsburg** - výrobce robotů KUKA
LIFO	**Last In First Out** - paměť, typu zásobníku
LSB	**Less Significant Bit** - nejméně významný bit
MSB	**Most Significant Bit** - nejvýznamnější bit
NCP	**Network Control Protocol** - síťový řídicí protokol
OSI	**Open Systems Interconnection** - propojování otevřených systémů
PC	**Personal Computer** - osobní počítač
PLC	**Programmable Logic Controller** - programovatelný logický automat
RTX	**Real-time Extension** - rozšíření systému Windows o reálný čas
TCP	**Transmission Control Protocol** - nejpoužívanější protokol transportní vrstvy TCP/IP
UDP	**User Datagram Protoco** - základní protokol transportní vrstvy TCP/IP
XML	**Extensible Markup Language** - rozšířitelný značkovací jazyk

1 Úvod

Jako téma své bakalářské práce jsem si vybral práci s průmyslovým robotem KUKA. Zaměřil jsem se hlavně na možnosti komunikace nadřazeného systému s robotem. Komunikace s robotem je v průmyslu již dlouho řešeným problémem. Nejčastěji je k robotu připojeno nějaké PLC, které zajišťuje bezpečnost celé aplikace a obvykle mimo robota řídí i všechny ostatní prvky technologie. Ke komunikaci robota a PLC se využívá standardních průmyslových sběrnic. Nejrozšířenějšími komunikačními sběrnicemi u PLC a u robotů jsou ProfiBus, ProfiNet, EtherCAT nebo asynchronní sériová linka. Pomocí těchto sběrnic lze realizovat komunikaci v reálném čase bez nežádoucích prodlev.

Složitější situace ale nastává v případě, kdy nastane potřeba přímé komunikace mezi robotem a počítačem typu PC. Existuje totiž řada aplikací, kde ani robot ani případné PLC neposkytuje dostatečné nástroje nebo výkon pro nějaký konkrétní úkol v rámci výrobní linky. Může se například jednat o úlohy typu rozpoznávání obrazu nebo přístup do nějaké externí databáze. V takových úlohách není nutná komunikace v reálném čase (s běžným operačním systémem není ani možné zaručit definované odezvy při komunikaci).

Připojené externí PC, plnící nějaký z uvedených úkolů, bychom mohli k robotu připojit přes hlavní PLC linky. U jednodušších linek ale pracovní buňka s robotem nemusí nutně PLC vyžadovat a navíc může vadit časová prodleva v přenosu dat mezi robotem a PLC a následně mezi PLC a externím počítačem. Tato prodleva často nemusí vadit, ale pokud bychom chtěli přenášet větší množství dat (například požadovanou trajektorii) nebo pokud bychom chtěli sledovat např. aktuální polohu robota s co nejmenším zpožděním, narazíme na limity tohoto řešení. Výrobce robotů KUKA sice nabízí ethernetové rozšíření, které by bylo schopné toto spojení mezi externím PC a robotem realizovat, ale toto rozšíření je placené a má svoje technické limity. Proto se tato práce zabývá hledáním a otestováním alternativního

způsobu, jak tuto úlohu efektivně řešit. Praktická část práce začíná seznámením se s vývojovým prostředím pro programování průmyslových robotů KUKA. Dále je zde představeno softwarové rozšíření Ethernet KRL od firmy KUKA a jeho možnosti. Jako alternativu k Ethernet KRL tato práce představuje program KukavarProxy, který se instaluje do řídicího systému robota a který se chová jako TCP/IP server. Tento program nabízí jednoduchou možnost implementace komunikace s externím systémem.

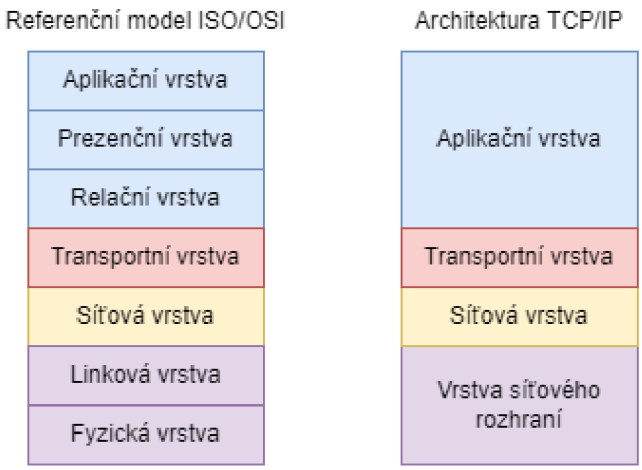
Program KukavarProxy byl vybrán pro realizaci demonstrační aplikace pro komunikaci s robotem a pro otestování dosažitelných vlastností takové komunikace.

2 Architektura TCP/IP

TCP/IP je v dnešní době nejpoužívanější rodinou protokolů. Název rodina protokolů označuje skupinu protokolů, které jsou navrženy tak, aby co nejlépe řešily danou problematiku. Protokol TCP/IP vznikl za účelem vylepšení přenosu datových paketů u první počítačové sítě ARPANET.

ARPANET z počátku používal protokol s názvem NCP, který sloužil k ověření samotné myšlenky možnosti přenosu datových paketů. Po úspěšném ověření funkce se ARPANET předal do používaní akademické sféře, ve které by však protokol NCP nesplnil po něm požadované operace, v tu dobu vznikl robustnější protokol TCP/IP.

Při vytváření protokolu TCP/IP se kladl velký důraz na finální implementovatelnost řešení přenosu. Tato skutečnost byla hlavní odlišností od konkurenčního modelu ISO/OSI. Autoři ISO/OSI usilovali při jeho vytváření o to, aby byl jejich model co nejrůznorodější k řešení různých požadavků, zatímco návrháři TCP/IP stavěli na jednoduchých problémech a postupným zdokonalováním dosáhli komplexnosti řešení. [1]

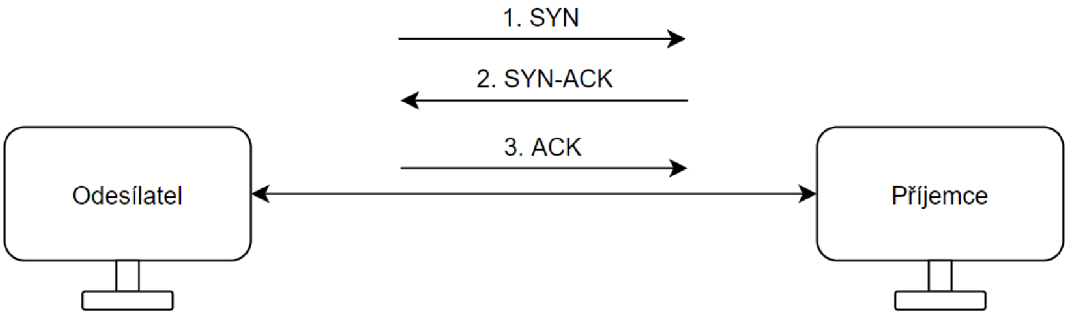


Obrázek 2.1: Porovnání blokových schémat modelu ISO/OSI a architektury TCP/IP

2.1 TCP vs UDP

Oba protokoly složí k přenosu paketů přes Internet. Protokoly pracují na principu odesílání paketů přes internetový protokol (IP), který zajistí směrování v síťové vrstvě. UDP protokol komunikuje rychleji za cenu ztráty paketů, zatímco TCP protokol se před odesláním nového paketu ujistí, že strana příjemce v pořádku přijala všechny pakety.

Odesílatel při navázání spojení protokolem TCP nejdříve pomocí synchronizační zprávy (SYN) zjistí, zda je příjemce připraven k přijímání dat. Příjemce potvrdí svoji připravenost k přijímání dat pomocí zprávy (SYN-ACK). Odesílatel potvrdí přijetí zprávou ACK, po odeslání třetí potvrzovací zprávy je vytvořeno otevřené spojení mezi dvěma body.

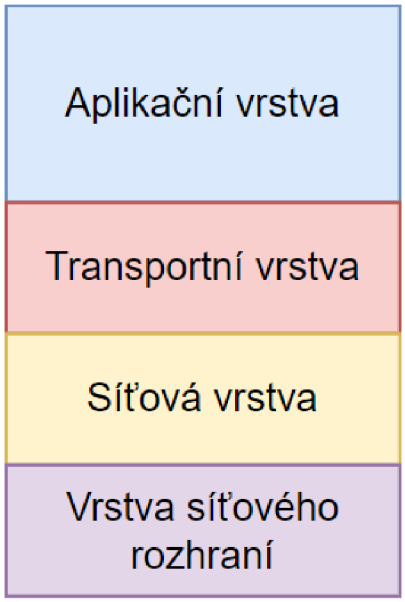


Obrázek 2.2: Průběh navázaní spojení pomocí protokolu TCP
[2]

Hlavní charakteristikou protokolu UDP je rychlost spojení a přenosu dat. Strana odesílatele a příjemce neodesílá žádné zprávy o navázaní spojení nebo potvrzení příjmu. [3]

2.2 Vrstvy TCP/IP

Vrstvy protokolu TCP/IP mají každá svoji danou úlohu a skládají se z mnoha dalších protokolů. Rozdíl od modelu ISO/OSI je určen počtem vrstev, kde ISO/OSI disponuje sedmi vrstvami a TCP/IP obsahuje pouze vrstvy čtyři. Autoři protokolu TCP/IP pouze sloučili vrstvy modelu ISO/OSI a vylepšili jejich vlastnosti.



Obrázek 2.3: Blokové schéma vrstev protokolu TCP/IP

Aplikační vrstva

V aplikační vrstvě protokolu jsou zahrnuty všechny protokoly, které umožňují uživateli konkrétní aplikaci přenosu dat po síti. Aplikační vrstva je rodinou protokolů, které zajišťují přenos dat přes Internet. Některé protokoly jsou přímo spojeny s transportním protokolem. Protokoly, které přímo vyžadují protokol TCP, se nazývají HTTP (Hypertext Transfer Protocol), FTP (File Transfer Protocol) a Telnet. Autoři postupovali při vytváření aplikační vrstvy nejjednodušším způsobem a rozhodli se ponechat implementaci tzv. podpůrných služeb na tvůrci aplikace, který síťový protokol implementuje.

Transportní vrstva

Transportní vrstva byla navržena podle modelu ISO/OSI. Uživatel u architektury TCP/IP má možnost vybrat si, zda upřednostní rychlost přenosu nebo spolehlivost přenosu. Autoři architektury navrhli dva transportní protokoly TCP a UDP. Protokol TCP zajišťuje spolehlivost a správné doručení dat, zatímco protokol UDP upřednostňuje rychlost před ztrátou dat.

Síťová vrstva

Autoři architektury dali při vyvíjení síťové vrstvy přednost rychlosti před spolehlivostí. Spolehlivost přesnosti přenosu připadá na vrstvy vyšší. Pro síťovou vrstvu byl zvolen internetový protokol (IP). Protokol IP se vyznačuje svým nespojovaným charakterem, který nedbá na to, zda bylo navázáno spojení mezi klientem a serverem a odesílá data do prázdna. Tato skutečnost přispívá k celkové robustnosti celé architektury.

Vrstva síťového rozhraní

Ve vrstvě síťového rozhraní nejsou blíže specifikované protokoly. Vrstva zajišťuje přístup k fyzickému přenosovému médiu. Z důvodů různých přenosových technologií (Ethernet, Token Ring, X.25) zajišťuje jednotné prostředí a služby pro všechny technologie.[4]

3 KukavarProxy

KukavarProxy je opensource rozhraní, které umožňuje komunikaci s průmyslovým robotem KUKA. KukavarProxy zprostředkovává komunikaci mezi běžným jádrem Windows a řídicí jednotkou robota. K této komunikaci KukavarProxy využívá služeb IntervalZero RTX64, která zajišťuje komunikaci mezi procesy reálného času v robotovi a procesy, které jsou spuštěny na operačním systému Windows. Pomocí KukavarProxy lze do robota zapisovat nebo číst proměnné, se kterými robot pracuje.[5]

Platforma IntervalZero RTX64

IntervalZero RTX64 je softwarové rozšíření, které z běžného operačního systému Windows vytvoří operační systém reálného času (RTOS). S pomocí platformy lze dosáhnout determinismu nebo operací pevného reálného času bez nutnosti zasahovat do hardwarové vrstvy operačního systému Windows. Plánovač RTX64 umožňuje vestavěným aplikacím přístup k fyzické paměti Windows.

Plánovač obsažený v RTX je nezávislý na integrovaném plánovači Windows a je schopný plánovat procesy v řádech mikrosekund. Pro potřebu zvýšení výkonu je RTX64 schopno nastavit počet jader, které bude pro operace reálného času používat. Výhodou procesu RTX je jeden izolovaný a kontrolovatelný subsystém, který je schopen zaručit veškeré vlastnosti a ovládání RTX.

Subsystém reálného času nastavuje jeho aplikacím vyšší priority než běžným aplikacím a funkcím systému Windows, zároveň je schopen vyhradit většinu jader procesoru pouze pro rozšíření RTX, čímž docílí výborné latence a odezvy systému na všechny požadavky.[6]

3.1 Vznik a funkce KukavarProxy

KukavarProxy byl vyvinut firmou IMTS s.l.r., která se zabývá vývojem technologií, jež umožňují koncovému zákazníkovi implementovat automatické sledovací systémy pro monitorování různých produktů ve výrobním řetězci. Server KukavarProxy je spuštěn na pozadí Windows části kontroléru a umožňuje tak připojení více klientů. V jeden okamžik se k serveru může připojit až deset klientů. Bezztrátový přenos dat je zajištěn protokolem TCP/IP. [7]

Při připojování aplikace na server musí uživatel respektovat tyto faktory:

- IP adresa robota
- port, na kterém server naslouchá (port 7000)
- správnost formátu požadavku

Firma IMTS s.r.l vyvinula aplikaci s názvem OpenShowVar, která slouží k navázání komunikace mezi robotem a externím PC. Zatímco rozhraní KukavarProxy může být spuštěno pouze na operačním systému Windows s jádrem reálného času obsaženém v jednotce řídící robota. OpenShowVar je aplikace navržena pro běžné operační systémy jako je Linux, MacOS a Windows. Aplikaci OpenShowVar je možné spustit na jakémkoliv zařízení a zahájit komunikaci s robotem přímo z něj, zatímco rozhraní KukaVarProxy může být spuštěno pouze na robotovi a uživatel stačí napsat TCP/IP klienta na běžném počítači, pomocí které se k KukavarProxy připojí.

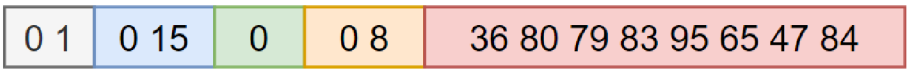
3.2 Formát požadavků

Formát požadavku pro čtení proměnné

KukavarProxy disponuje předem definovaným formátem zprávy, kterou od uživatele přijme. V tomto formátu je přesně definována délka, datový typ a pořadí dílčích parametrů:

- ID požadavku - 2 byty typu uint16

- délka požadavku - 2 byty typu uint16
- režim čtení/zápisu - 1 byt typu uint16 (0 pro čtení)
- délka názvu proměnné - 2 byty typu uint16
- název proměnné - N bytů kódovaných pomocí ASCII

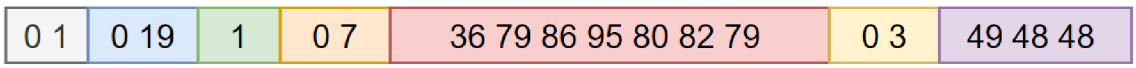


Obrázek 3.1: Bitové vyjádření požadavku pro zjištění pozice

Formát požadavku pro zápis proměnné

Formát požadavku pro zápis proměnné se od požadavku pro čtení liší pouze v hodnotě, která je do dané proměnné zapsána. Délka a hodnota pro zapsání je přidáme na konec požadavku:

- ID požadavku - 2 byty typu uint16
- délka požadavku - 2 byty typu uint16
- režim čtení/zápisu - 1 byt typu uint16 (0 pro zápis)
- délka názvu proměnné - 2 byty typu uint16
- název proměnné - N bytů kódovaných pomocí ASCII
- délka hodnoty pro zapsání - 2 byty typu uint16
- hodnota proměnné - M bytů kódovaných pomocí ASCII



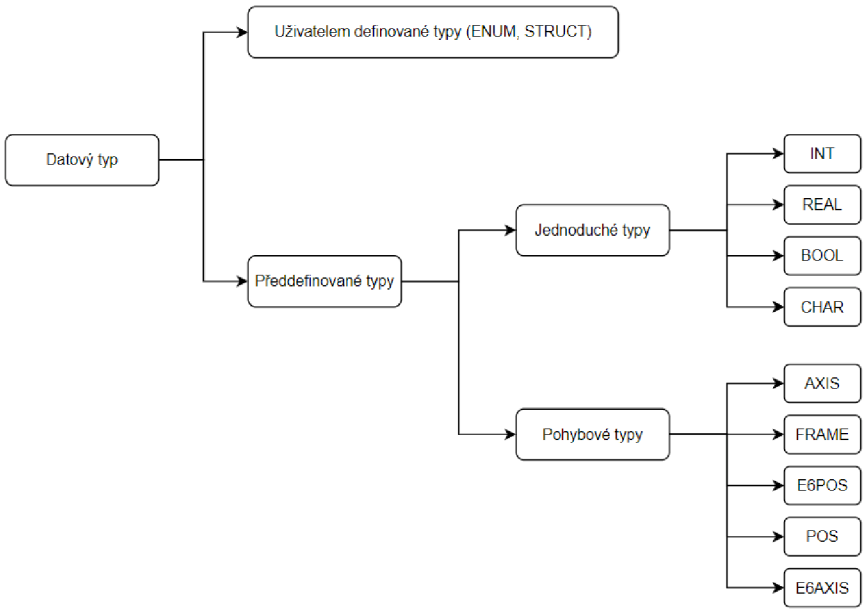
Obrázek 3.2: Bitové vyjádření požadavku pro zapsání rychlosti

Formát odpovědi na požadavek uživatele

Formát odpovědi na požadavek uživatele specifikuje, na který požadavek robot odpovídá, v jakém datovém typu bude jeho odpověď vyjádřena a zda byl požadavek správně přečten. Vše splňuje předem definovanou strukturu:

- ID požadavku - 2 byty typu uint16
- délka požadavku - 2 byty typu uint16
- režim čtení/zápisu - 1 byt typu uint16 (0 = zápis, 1 = čtení, 2 = čtení pole, 3 = zápis do pole)
- délka hodnoty proměnné - 2 byty typu uint16
- hodnota proměnné - N bytů kódovaných pomocí ASCII
- zpráva o stavu čtení/zápisu - 3 byty typu uint16 (000 pro chybu, 011 pro úspěch)

Robot spolu s hodnotou proměnné, na kterou se uživatel dotazoval, pošle informaci, o jaký typ proměnné se jedná. Roboti od společnosti KUKA pracují s proměnnými, které jsou předem definovány.



Obrázek 3.3: Dělení datových typů KRL

KukavarProxy pracuje především s předdefinovanými typy. Autorova aplikace se zabývá proměnnými, které spadají do části pohybových typů. Z pohledu zadání byla nejdůležitější proměnná **$POS_ACT**, která popisuje aktuální pozici nástroje robota v kartézských souřadnicích. Proměnná je datového typu E6POS, jež odkazuje na to, že proměnná bude obsahovat složitější struktury a bude serverem poslána jako pole hodnot.[8]

Odpověď na dotaz o aktuální poloze:

E6POS: X -928.5408, Y -64.18051, Z 849.3752, A -134.9867, B -29.41913, C 165.6175, S 1, T 39, E1 0.0, E2 0.0, E3 0.0, E4 0.0, E5 0.0, E6 0.0

V této struktuře je zobrazena poloha nástroje robota vzhledem k souřadnicovému systému základny, ve které X, Y a Z jsou odsazení počátku podél os v milimetrech a A, B, C jsou rotační odsazení úhlů os ve stupních.[9]

4 Ethernet KRL

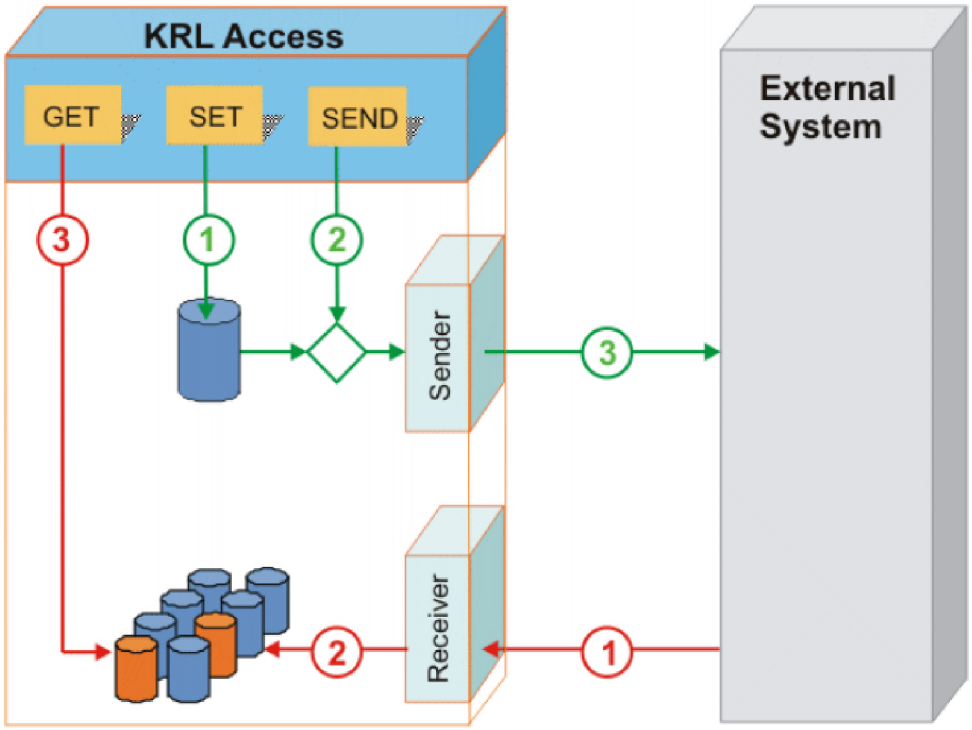
KUKA Ethernet KRL je rozšíření jazyka KRL, díky kterému je robot schopen komunikovat přes ethernetové rozhraní. S robotem je schopen komunikovat přes protokol TCP/IP a UDP/IP. Komunikace přes protokol UDP nezaručuje správné a bezchybové doručení zprávy příjemci, je tedy z pohledu využití popisované aplikace zcela nevhodný.[10]

4.1 Funkce Ethernet KRL

Ethernet KRL je doplňkový softwarový balíček, který zajišťuje komunikaci externího systému s řídicí jednotkou robota. Výměnu dat mezi externím a řídicím systémem lze zprostředkovat pomocí XML dat, která jsou přijata nebo odeslána do externího systému. Datová výměna může probíhat i za pomoci binárních dat.

4.2 Způsob komunikace Ethernetu KRL

Ethernetové připojení je definováno pomocí konfiguračního souboru, který musí být uložen ve správném adresáři na kontroléru robota. Konfigurace spojení je načtena při vytvoření spojení, ke kterému dojde spuštěním mnou vytvořené aplikace. Čas odezvy řídicího systému je závislý na objemu přijímaných/odesílaných dat a dobře napsaném programu pro ovládání pohybu robota. V ideálním případě lze dosáhnout odezvy až 2 ms. V případě ztráty spojení zajišťují funkce EKI správnost zpracování přijatých dat.



Obrázek 4.1: Příklad komunikace pomocí Ethernet KRL [10]

Přijímání dat z externího systému (označeno červeně) přenášených pomocí protokolu a jejich správné přijmutí je zajištěno EKI (1). Následně se data uloží strukturovaně do paměti (2), odkud je k nim přistupováno pomoci KRL příkazu GET (3).

Před odesláním dat je pomocí příkazu SET nastavena požadovaná proměnná do datové pamětí (1), odkud si ji pomocí instrukce SET uživatel vyžádá (2) a rozhraní EKI ji pomocí protokolu odešle do externího systému (3).

4.3 Práce s daty

Všechna přijatá data jsou automaticky ukládána do datové paměti systému. Data je systém schopen přijmout v binárním nebo XML formátu, s každým typem dat je nakládáno jiným způsobem. Přijatá data jsou do systému uložena jako paměťové zásobníky (FIFO/LIFO). Data přijatá ve formátu XML jsou ukládána dle svého typu do různých paměťových zásobníků. Pro binární data je vytvořen jeden datový zásobník na jedno připojení.

V případě čtení uložených dat z paměti záleží na tom, jak byla data uložena do datového zásobníku. Pokud jsou data uložena do fronty (FIFO), první uložený prvek je prvním přečteným, v případě uložení do zásobníku (LIFO) je poslední uložený prvek prvním přečteným. Velikost paměti pro ukládání dat je pevně daná, ve chvíli, kdy dojde k překročení limitu, je ethernetové spojení ukončeno, protokol tím zajistí vyloučení ztráty dat z důvodu nedostatku místa.

Rozdíl mezi komunikací Ethernet KRL a KukavarProxy

KukavarProxy využívá k přístupu do paměti softwarové rozšíření IntervalZero RTX64, zatímco Ethernet KRL, jakožto softwarové rozšíření od firmy KUKA, má zajištěno přístup do paměti přímo. Důsledkem toho je, že KukavarProxy potřebuje pro svou správnou funkci propojení systému Windows s řídicím systémem robota. Problém s tímto propojením řeší systém RTX, který umožňuje čtení z adresovatelné paměti robota.

Sběrnice zajišťující komunikaci mezi PLC a robotem

Řídicí jednotka robota obsahuje PLC. Z toho plyne, že řídicí jednotka robota využívá celého jeho potenciálu. Hlavním potenciálem PLC jsou jeho průmyslové sběrnice. Pro PLC jsou standardizované sběrnice ProfiBus, ProfiNet a EtherCAT, všechny tyto sběrnice slouží pro automatizaci výrobních linek.

Sběrnice ProfiNet je založena na průmyslovém Ethernetu a pracuje s jeho přenosovým protokolem TCP/IP. Jeho rychlost se pohybuje mezi 10 až 100 Mbit/s. ProfiNEt využívá mezinárodního standardu, který umožňuje připojení webových klientů přes k tomu určené protokoly (*HTTP, XML, HTML*). ProfiNet lez při využití mezinárodních standardů propojit s jinými průmyslovými sběrnicemi, zpravidla se ProfiNet propojuje se sběrnicemi ProfiBus nebo InterBus. Sběrnice ProfiNet dovoluje uživateli přenášet data ve standardizovaných tvarech XML a HTML.[11]

Další sběrnice, která byla založena na principu Ethernetu, se jmenuje EtherCAT. Funkčnost této sběrnice je založena na upřednostňování časově citlivých dat před daty méně citlivými nebo daty s velkým zatížením. EtherCAT se používá hlavně pro přenosy dat tvrdých a měkkých výpočtů v reálném čase, jejichž rychlost se blíží 100 Mbit/s. Sběrnice EtherCAT dovoluje svým plně duplexním charakterem použití

všech síťových topologii, tato funkce umožňuje uživateli vytvořit velmi složité uspořádání zařízení připojených na sběrnici. Pro svou vysokou rychlost je EtherCAT používán v implementacích s vysokým požadavkem na synchronicitu, bezpečnost a integritu dat. Typickými aplikacemi jsou tváření kovů, montážní systémy a robotika.[12]

Nejpomalejší, ale stále využívanou sběrnicí je ProfiBus, u které rychlost přenosu dat dosahuje 9 KiBit/s až 12 MiBit/s. Sběrnice se využívá hlavně z důvodu možnosti její délky, která může v krajních případech dosahovat až 1,2 Km. Přenosová technologie je založena na principu sériové sběrnice RS485, která umožňuje kombinaci metalických a optických přenosových cest. Ovládání sběrnice funguje na principu dvou metod, které jsou určeny logickým uspořádáním sběrnice. První metodou je token ring, tato metoda se používá v případě kdy je nutné předat stejnou informaci na více místech. Metoda klient-server zajištuje předaní informace mezi dvěmi body sběrnice. [13]

5 Aplikace pro komunikaci

Navržená aplikace zajišťuje komunikaci mezi PC a řídicím systémem robota, respektive KukavarProxy. Aplikace je vytvořena jako vícevláknová, aby byl uživatel schopen v libovolném okamžiku přerušit její chod.

5.1 Vývojové prostředí a volba programovacího jazyka

Vývojové prostředí Visual Studio

Pro vývoj a testování aplikace bylo zvoleno vývojové prostředí (IDE) Microsoft Visual Studio, které se používá pro vývoj a testování konzolových aplikací a aplikací s grafickým rozhraním. Visual Studio obsahuje mnoho vestavěných nástrojů, editor kódu spolupracující s technologii IntelliSense. Nástroj IntelliSense pomáhá vývojáři v lepší orientaci v právě používaných funkcích, knihovnách a nabízí velmi zajímavé návrhy pro pokračování kódu. Dalším výkonným nástrojem je vestavěný debugger, který pracuje jak na úrovni kódu, tak na úrovni stroje.

Visual Studio podporuje rovněž tvorbu grafických rozhraní aplikace, tříd a databázových schémat. Všechna rozšíření jsou eventuálně přidávána v průběhu tvoření kódu, což usnadňuje práci při vývoji aplikace. Podpora programovacích jazyků probíhá formou jazykových služeb, což umožňuje Visual Studiu pracovat v podstatě s každým jazykem, pro který byla vytvořena jazyková služba. Základními jazyky, které Visual Studio používá, jsou C/C++, C# a VB.NET. Podpora jazyků jako je F#, Python, Ruby atd. funguje přes nainstalování jazykových balíčků pro příslušný jazyk.

Visual Studio je komplexním vývojovým prostředím, v němž je uživateli umožněno vybrat si z mnoha způsobů programování a programovacích jazyků. Uživatel využívající všechny vestavěné funkce si nadmíru usnadní práci při vývoji. [14]

Volba programovacího jazyka C#

Visual Studio umí pracovat s mnoha programovacími jazyky. Při studování jazyků, které podporují tvorbu grafických aplikací, byl využit programovací jazyk C#. Programovací jazyk C# je objektově orientovaný, má obrovskou výhodu v přehlednosti programové části.

Objektově orientovaný programovací jazyk je jazyk, ve kterém se výkonný kód děje v objektech, což umožňuje snadnou přenositelnost našich objektů mezi různými programy. Objekty jsou jednotlivé prvky reality, které jsou seskupeny do entit a představují daný objekt. Každý objekt představuje definovanou operaci, kterou navenek sdílí s hlavním programem při volání objektu. Díky stromové organizaci objektů je programátor schopen dědit z jiného druhu objektu. Základním předpokladem objektově orientovaného programování jsou čtyři principy.

Prvním principem je zapouzdření dat a metod, každý objekt se vyznačuje svou částí, kterou nesdílí s okolím. V této části jsou data vnějšímu okolí neviditelná, přístup k nim mají pouze interní funkce metody. Rozšířením zapouzdření je abstrakce, která sdílí s okolím pouze nezbytně nutná data a která skrývá svou programovou podstatu. Výhodou abstrakce objektu je možné rozšíření bez zásahu do vnitřní funkce třídy.

V mnoha případech je žádoucí, aby objekty sdílely části své logiky s objekty jinými. Této vlastnosti se říká dědičnost, která rozdělí třídy na potomky a rodiče. Potomek je odvozen od rodiče, využívá metody a data na základě implementované dědičnosti z nadřazené třídy a přitom má k dispozici svou jedinečnou funkčnost sám o sobě.

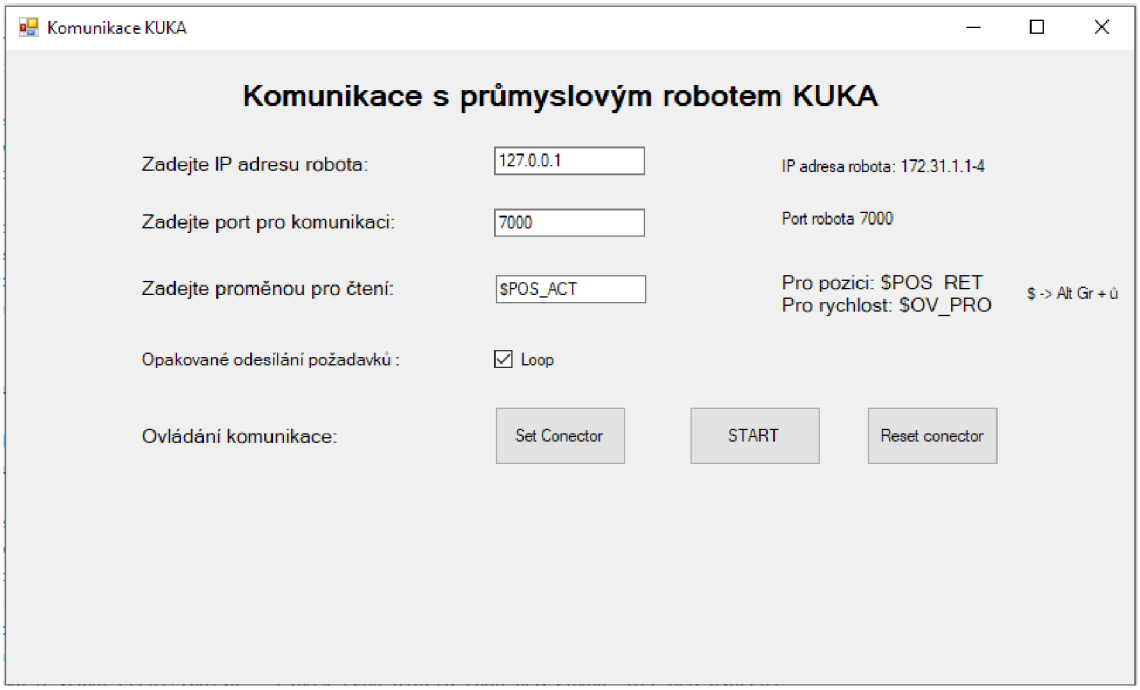
V případě, že rodič má více potomků a naším záměrem je dědit z rodičova potomka, využijeme vlastnosti objektově orientovaného programovaní nazvané polymorfismus. Vytvořením abstraktních entit nám jazyk C# umožní používat třídu exaktně jako rodič třídy. Toto lze provádět opakovaně a potomci odvození od potomků rodičů tak mohou mít svoji vlastní jedinečnou funkcionalitu.[15]

Tento princip si lze představit jako aplikaci pro databázi zaměstnanců, ve které má každý zaměstnanec své pevně dané jméno, příjmení, datum narození a náplň práce. Za tímto účelem byla vytvořena třída, jež všechny tyto prvky zohlednila.

Zaměstnanec má svoji jedinečnou náplň práce, která je definována pomocí metody, která je ve třídě definována obecně a každý zaměstnanec si ji dále upřesní dle svého uvážení.

V případě demonstrační aplikace jsou používány objekty, které nejsou spojeny s grafickým rozhraním aplikace (vytvoření zprávy, vytvoření spojení, uložení odpovědi). Jednotlivé prvky objektu mezi sebou sdílí data nutná k vypracovaní jejich programové části. Pro účely popisované aplikace jsou využity následující parametry: IP adresa, port a proměnná, kterou chce uživatel číst/zapisovat.

5.2 Grafické rozhraní aplikace



Obrázek 5.1: Grafické rozhraní aplikace

V grafickém rozhraní jsou umístěny prvky pro definování spojení a proměnné, kterou chce uživatel číst. Mezi grafické komponenty aplikace patří tlačítka, textové boxy a zaškrtávací políčko. Pro tvorbu grafického rozhraní byl vybrán první framework platformy .NET Windows Forms.

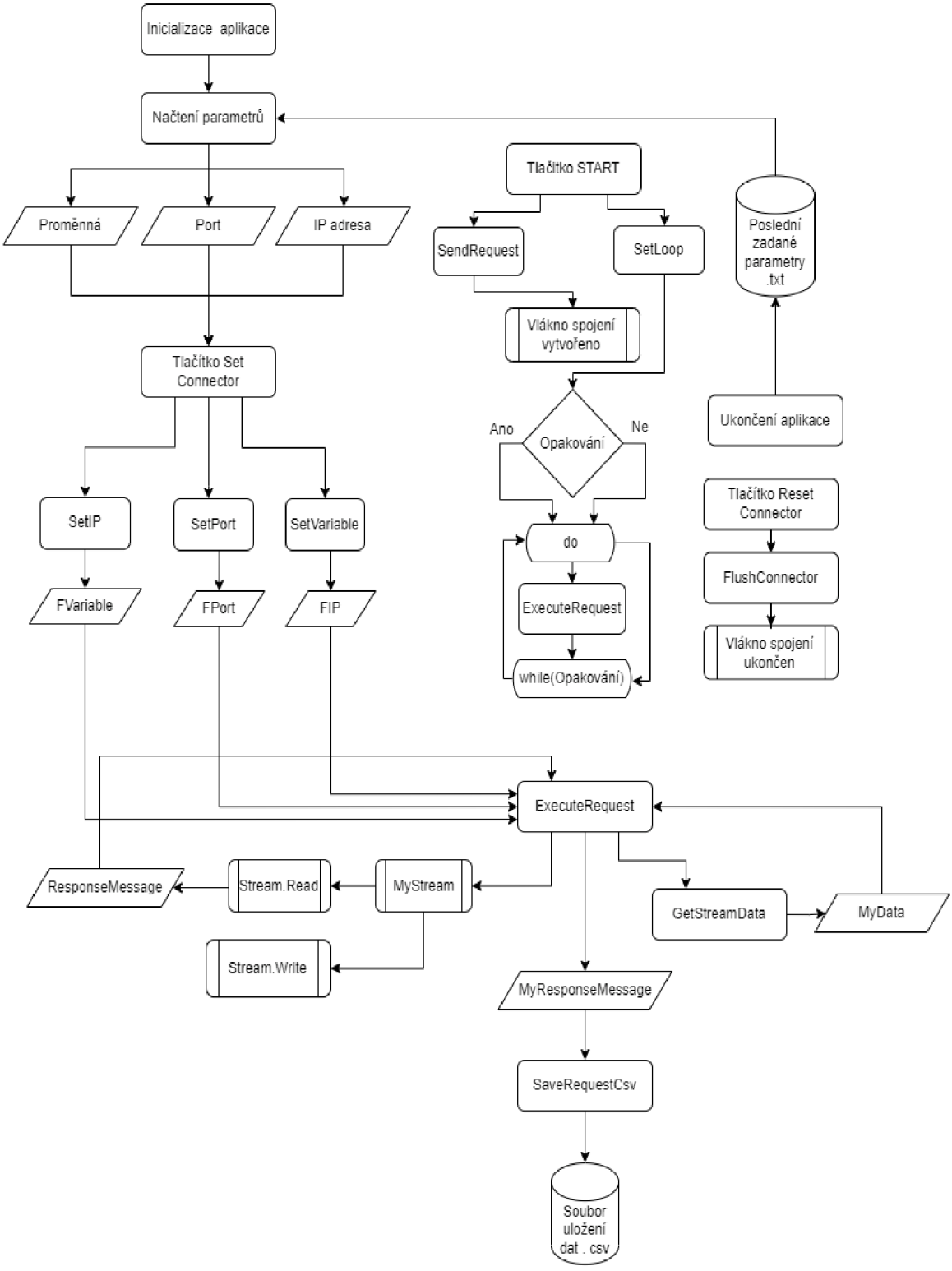
Jazyk C# patří do třetí generace programovacích jazyků. Na rozdíl od jazyků první a druhé generace jsou jazyky třetí generace velmi podobné lidskému vnímání operací a je možné si představit určitou abstrakci nad tím vystavěnou. Třetí generace jazyků se dělí do třech kategorií: kompilované jazyky, interpretované jazyky a jazyky s virtuálním strojem. Jazyk C# spadá do kategorie třetí a jeho virtuálním strojem je platforma .NET. Výhodami jazyků s virtuálním strojem jsou stabilita, jednoduchý vývoj, rychlost, přenositelnost.

Platforma .NET je opensource vývojová platforma od firmy Microsoft pro výstavbu různých typů aplikací. Vývojová platforma nabízí sadu mnoha různých jazyků a knihoven, které lze při vývoji aplikace snadno implementovat. Platforma .NET Framework má v sobě zabudovanou sadu ovládacích komponent a je schopna pracovat s několika programovacími jazyky (C#, Java, Object Pascal, atd.). Všechny jazyky jsou objektově orientované a obsahují různé výhody. Největší výhodu má však programovací jazyk C#, který byl vyvinut právě pro framework .NET. Platforma .NET obsahuje mnoho dalších platforem, které umožňují vyvíjet aplikace pro různé operační systémy.

První z mnoha platforem je .NET Core, který umožňuje vytvořit aplikaci pro jakýkoliv počítačový operační systém (Windows, Linux, macOS). Platforma .NET Framework slouží pro vyvíjení okenních aplikací a webových stránek pro operační systém Windows. Frameworky určené pro vývoj aplikací na mobilní telefony (operační systém iOS nebo Android) se nazývají Xamarin a Mono. Vývojář má k dispozici framework .NET Standard, který zaručuje běh aplikace na jakémkoliv zařízení bez ohledu na operační systém.

Největší výhodou platformy .NET jsou knihovny, které jsou Microsoftem dodávané jako kompletní sada. Microsoft vytvořil pro všechny struktury a komponenty knihoven velmi přehlednou dokumentaci. Knihovny umožňují práci s konzolí, databázemi a formulářovými prvky.[16]

5.3 Logická část programu



Obrázek 5.2: Vývojový diagram aplikace

Konvence pojmenovávání proměnných

Pro program byla zvolena konvence pojmenovávání proměnných takovým způsobem, aby byl program pro autora co nejpřehlednější. Při deklaraci lokální proměnné, která vznikne v kódu jako například pomocné proměnné, obsahuje před svým názvem klíčové slovo **My** *(např. MyFileName, MyKukaSender...)*. Parametry metody jsou odlišeny písmenem **A** *(AInputID, AInputIP...)*. Písmeno **T** slouží pro odlišení autorem definované třídy *(TKukaSender)*. Privátní proměnné v metodách a funkcích byly označeny písmenem **F** *(FIP, FPort...)*. Programovací jazyk C# označuje proměnné zcela odlišně. Proměnné v průběhy tvorby byly přejmenovány, aby bylo sjednoceno jejich formátování.

První spuštění aplikace pro komunikaci

Po prvním spuštění aplikace je nutné zadat počáteční parametry pro komunikaci, mezi než patří IP adresa, která je pro daného robota pevně definovaná a každý robot je rozlišen posledním číslem IP adresy. Parametr portu je z pohledu KukavarProxy pevně dán, ale pro komplexnost aplikace může uživatel hodnotu portu definovat. Posledním parametrem je proměnná, kterou chce uživatel přečíst, popřípadě zapsat. Název proměnné se musí shodovat s názvem proměnné, která je definována v dokumentaci pro KRL proměnné.

Po zadání parametrů pro komunikaci si uživatel může vybrat, zda chce poslat požadavek jen jednou nebo ho opakovat do zastavení aplikace. Může tak učinit zaškrtnutím políčka *Opakované odesílání požadavků*. Nastavení všech příslušných parametrů proběhne v programu stisknutím tlačítka *Set Connector*. Pro zahájení komunikace je nadefinováno tlačítko *START*, které zahájí komunikaci s robotem, naopak pro zastavení komunikace slouží tlačítko *Reset Connector*. Uživateli je umožněno znovu nastavit parametry a opakovat celou proceduru znovu. Pokud se uživatel rozhodne aplikaci ukončit, může tak učinit pomocí klávesové zkratky Alt+F4, nebo tlačítkem k tomu určenému.

Uložení parametrů po ukončení aplikace

Před ukončením aplikace se vyvolá vlastnost platformy .NET Framework *FormClosed*, v této vlastnosti je nastaveno uložení všech uživatelem zadaných parametrů. Ve vlastnosti byly načteny v první řadě všechny parametry z textových polí určených k zadání jednotlivých parametrů.[16]

Pro textová pole IP adresy a proměnné byla využita metoda *Object.ToString()* k převedení textu do textového řetězce. Metodu *Object.ToString()* lze použít k více operacím, její výchozí implementací je vrácení plně kvalifikovaného typu daného objektu, na který je metoda použita. V případě popisované aplikace metoda vrátila datový typ řetězce.

Vyčtení čísla portu bylo složitější, jelikož další používané metody vyžadují celé číslo jako datový typ parametru portu. Na textové pole portu byla znovu použita metodu *Object.ToString()* a následně byla implementována metoda *Int16.Parse()*. Tato metoda převádí řetězcové vyjádření čísla na jeho 16bitovou podobu čísla se znaménkem.

V následujícím kroku byl založen soubor s příponou *.txt*, do kterého jsou parametry ukládány. Pro vytvoření cesty k souboru byla aplikována metoda *Path.Combine()*. tato metoda je schopna spojit až čtyři textové řetězce do sebe a vytvořit z nich umístění v paměti počítače. Pro uvedenou implementaci stačily pouze dva řetězce. Prvním řetězcem bylo aktuální umístění aplikace na uživatelově počítači, druhým řetězcem bylo autorem definované jméno pro soubor, do kterého budou parametry ukládány.

Pro využití stejného textového souboru po každém spuštění aplikace musel být jeho obsah při každém zápisu smazán. Pro smazání obsahu souboru byla implementována metoda *File.WriteAllText()*, která otevře soubor na zadaném umístění a přepíše celý obsah souboru na definovaný řetězec, v případě aplikace se jednalo o řetězec prázdný.

Načtené parametry jsou v dalším kroku zapsány do souboru pomocí třídy *StreamWriter*. Třída *StreamWriter* potřebuje pro svou správnou funkci cestu k souboru, do kterého zapisuje, lze však definovat i další parametr, který je datového typu Boolean, a který třídě oznámí, zda bude zapisovat řetězce jako jeden nebo je bude

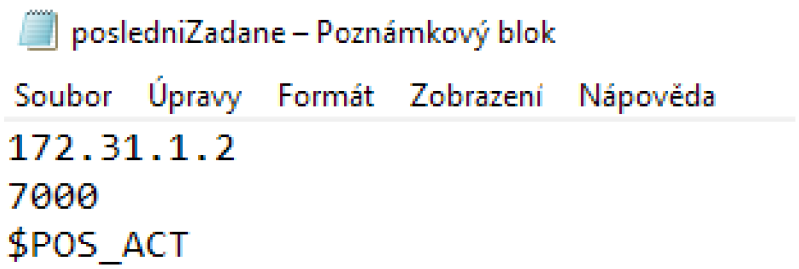
zapisovat jednotlivě na nový řádek. Třída má tento parametr standardně vypnut, řetězce se tedy zapisují za sebe. K předefinování bylo nutné za řetězec umístění připsat boolovské vyjádření pravdy.

Tento způsob se dá do budoucna vylepšit ukládáním parametrů do inicializačního souboru. Platforma .NET Framework v současné době nemá podporu pro ukládaní souborů s příponou *.ini*, musela by být vytvořena nová třída, ve které by bylo nadefinováno chování inicializačních souborů, které by mohly toto řešení využít.

Načtení parametrů po spuštění aplikace

Po uložení parametrů z předchozí instance aplikace se po každém dalším spuštění parametry načtou do příslušných textových polí. Při spuštění aplikace se vykoná funkce *public KUKAForm()*, jejíž tělo obsahuje metodu *InitializeComponent()*, která vytvoří prvky aplikace a nastaví jim jejich parametry. Tato metoda je standardně definována platformou .NET Frameworke. V další části funkce se vykoná načtení parametrů do textových polí. Kód je řešen obdobným způsobem jako ukládání parametrů. Nejdříve dochází k nastavení cesty k souboru, ve kterém jsou parametry uloženy.

Vytvoření cesty je řešeno metodou *Path.Combine()*, k následnému přečtení parametrů je aplikována metoda *File.ReadAllLines()*. Tato metoda otevře soubor v definovaném adresáři a načte každý řádek do pole řetězců. Vypsání parametrů je poté řešeno jednoduchým *for* cyklem, znalostí posloupnosti ukládaných parametrů a vlastností textového pole *TextBox.Text*. Vlastnost textového pole nastaví daný text do příslušného prvku.



Obrázek 5.3: Ukázka souboru pro načítaní parametrů

Po každém spuštění aplikace se soubor pro uložení dat přepíše pomocí metody *File.WriteAllText()* na prázdný textový řetězec. Na první řádek souboru se uloží parametry dané pro proměnnou $POS_ACT. Proměnná je typu *E6POS*, na první řádek se tedy zapíší souřadnice X, Y, Z a natočení podél os A, B, C.

Třída TKukaSender

Třída zajišťuje fungování celé aplikace mimo hlavní vlákno programu. V programovacím jazyce C# jsou třídy jednou z možností, jak zobecnit a vyčistit program. Fungování třídy je závislé na správném nastavení a předání vstupních parametrů.

Na začátku celé třídy jsou deklarovány privátní proměnné, které budou nadále ve vytvořených metodách používány. Privátní proměnné jsou v jazyce C# dostupné pouze ve třídě nebo struktuře, pro kterou byly nadeklarovány. Pro autorovu implementaci bylo vhodné je takto nastavit. Třída pokračuje vytvořením vlákna pomocí třídy Thread.

Metody Get() a Set()

V dalším bloku jsou definované metody pro načtení a jejich následné zapsání do privátních proměnných, které nastaví parametry pro komunikaci k následnému použití. Metody *Get()* a *Set()* jsou inspirovány interními vlastnostmi .NET Framework *get* a *set*. Metoda get vyčte parametr a uloží jej do své privátní proměnné, metoda set ji v dalším kroku nastaví do privátní proměnné třídy.

Metoda GetStreamData()

Metoda *GetStreamData()* je metodou datového typu pole bytů. V této metodě dojde k vytvoření požadavku, který má být poslán serveru KukavarProxy. Kódování zprávy se řídí pevně danou strukturou KukavarProxy. Na začátku kódování je definována celková délka požadavku, která byla určena spočítáním všech bitů, které jsou pevné a po celou dobu vykonávání programu se nemění. K těmto bitům byla přičtena jediná možná proměnnou délku, kterou je počet znaků uživatelem zadané proměnné. Získání tohoto počtu bylo dosaženo vlastností *String.Lenght*, která spočítá přesný počet znaků v řetězci, na němž byla vlastnost použita.

Pro vytvoření požadavku v datovém typu byte bylo použito přetypování z řetězce a celého čísla na byte. Přetypování v jazyce C# je uskutečněno přidáním závorky s žádaným typem před přetypovávaný typ. KukavarProxy definuje, že parametry požadavku, které nejsou názvem proměnné, jsou přenášeny pomocí dvou bytů. Z tohoto důvodu byly nadefinovány bity s nejmenším a největším významem (LSB a MSB). Při výpočtu bylo vycházeno ze znalosti datového typu byte, který je schopen interpretovat číslo 0-255.

Bit s největším významem je bit, který znázorňuje největší číslo ve dvojkovém zápisu, dá se tedy získat podělením aktuálního čísla největším možným číslem, které je byte schopen zobrazit.[17] Bit s nejmenším významem byl získán zjištěním zbytku po dělení číslem 256.[18] Jméno proměnné je pouze přetypováno na datový typ byte a zapsáno na konec požadavku.

Metoda SaveRequestToCsv()

Metoda vyčistí a uloží přijatou zprávu do souboru s příponou *.csv*. Soubor CSV, neboli *Comma Separated Values*, má definované oddělovače jednotlivých hodnot. Nejčastěji se jako oddělovač používá tečka, středník nebo tabulátor. V autorově aplikaci je jako oddělovač použit středník.

Tabulka 5.1: Ukázka přijatých dat

X [mm]	Y [mm]	Z [mm]	A [°]	B [°]	C [°]	ODEZVA [ms]
307.3490	-1139.912	613.0444	-132.3377	-11.41061	-156.8258	5,1322
313.1039	-1151.238	630.5124	-133.1892	-12.29605	-155.0395	4,6754
318.9886	-1162.469	648.9775	-134.0938	-13.15786	-153.1617	3,5775
324.9881	-1173.546	668.4591	-135.0514	-13.98960	-151.1910	3,6658
330.6833	-1183.700	687.5952	-135.9939	-14.73373	-149.2648	8,9462
340.9887	-1201.140	723.9247	-137.7825	-15.96196	-145.6319	4,016
346.0010	-1209.166	742.4471	-138.6914	-16.50130	-143.7912	3,6009

K zapsání do předem připraveného souboru dojde pomocí třídy *StreamWriter*. Tato třída používá klíčové slovo using, které správně vyčistí textové řetězce ve třídě používané. Příkaz using na sebe automaticky po ukončení vykonávání kódu zavolá metodu *Dispose*, tato metoda uvolní nebo resetuje nepoužívané prvky a připraví se tak na zapsání nového řetězce.

Metoda ExecuteRequest()

V této metodě dojde k vytvoření spojení mezi externím PC a robotem. Využitím třídy *TcpClient* se zahájí spojení dle uživatelem definovaných parametrů. Definování nové instance třídy proběhne klíčovým slovem new a zadáním parametrů IP adresy a portu. Za pomoci další třídy *NetworkStream* je vytvořen datový proud pro přístup k síti. Aplikováním metody *GetStream()* na instanci třídy *NetworkStream* je vyvolán datový proud, který je schopen posílat data v instanci třídy *TcpClient*. Pomocí metody *NetworkStream.Write()* je zapsán do datového proudu požadavek, který byl získán použitím *GetStreamData()*.

Každých 100 ms se odešle jeden požadavek. Časový úsek byl změřen pomocí instance třídy *Stopwatch*, která poskytuje nástroje pro přesné měření uplynulého času. Využitím vlastnosti třídy *Elapsed* je získána hodnota uplynulého času od zahájení měření času. Tato hodnota je uložena do struktury *TimeSpan*, díky které je následně vlákno pro komunikaci uspáno na 100 ms.

Vlastností *ReadTimeout()* je ošetřen problém, který by mohl vzniknout v případě zadání špatných parametrů spojení nebo neaktivním serverem KukavarProxy. Vlastnost nastaví hodnotu v milisekundách která určí, jak dlouho se datový proud bude snažit přečíst odpověď před tím, než spojení ukončí. Pokud uplyne definovaný čas, vlastnost vyvolá výjimku *InvalidOperationException*, která je zachycena a uživateli pomocí vyskakovacího okna oznámena.

Přijímaní odpovědi probíhá pomocí metody *NetworkStream.Read()*. Metoda jako parametry využívá pole bytů, index v poli, od kterého začne zapisovat a délku pole. Aplikováním metody *NetworkStream.Read()* jsou vyčtena všechna data na příchozí data na datovém toku spojení. Přijatá data jsou v datovém typu byte. Data se musí dekódovat zpět do textového řetězce. Za tímto účelem byla použita metoda *Encoding.GetString()*. Metoda přepíše v odvozené třídě byte do řetězce.

Metoda SendRequest()

Metodou SendRequest() je vytvořena nová instance třídy *Thread*. Spuštěním aplikace se zformuje její instanci nazvaná proces. Vytvořením procesu jsou spuštěna pro příslušnou aplikaci její vlákna. Jednoduché aplikace mohou běžet pouze na jednom hlavním vlákně, v případě aplikace vytvořené v rámci této bakalářské práce to ale nebylo možné.

Aplikací se spouští komunikace, která se vykonává dle strojového kódu. Vykonávání dle strojového kódu znemožnilo možnost vypnutí aplikace nebo zastavení komunikace. Vytvořením speciálního vlákna pro komunikaci bylo docíleno možnosti přerušit spojení grafickými prvky v jakémkoliv čase. Ke spuštění vlákna byla aplikována metoda *Thread.Start()*, tato metoda změní stav aktuálního vlákna na *Running*.

Metoda FlushConnectort()

V metodě *FlushConnector()* dojde k ukončení vlákna, na kterém se vykonává komunikace. Pro tuto akci je použita metoda *Thread.Abort()*, která v případě jejího vyvolání zahájí proces ukončení vlákna. Metoda *Abort* při ukončování vlákna vyvolá výjimku *ThreadAbortException*, k ošetření výjimky dochází v metodě *ExecuteRequest()*.

Možnosti ovlivnění trajektorie robota

Aplikaci je možné upravit a umožnit tak uživateli, aby robota ovládal. V tomto případě by ale bylo potřeba vytvořit souběžně program, který by byl spuštěn v řídicí jednotce robota a přijímal příkazy z rozhraní KukavarProxy. Úprava v logické části vytvořené aplikace by byla minimální. Aplikace by zpracovala pomocí přidání dalšího textového pole požadovanou hodnotu proměnné, kterou by chtěl uživatel ovlivnit. Na straně robota by bylo potřeba vytvořit zcela nový program, který by byl schopen přijímat příkazy vytvořené uživatelem a zpracované rozhraním KukavarProxy.

5.4 Obsluha grafického rozhraní

Po prvním spuštění aplikace je nutné, aby uživatel zadal všechny parametry pro komunikaci do příslušných textových polí. Uživatel v dalším kroku stisknutím tlačítka *Set Connector* nastaví příslušné parametry do programu aplikace. Po stisknutí tlačítka *START* uživatel zahájí spojení s robotem, pro ukončení spojení slouží tlačítko *Reset Connector*. Spojení je také možné ukončit normálním způsobem pomocí tlačítka k uzavření aplikace, nebo klávesovou zkratkou *ALT+F4*

Operace vyvolané stisknutím tlačítka Set Connector

Po stisknutí tlačítka se provede tělo události *SetConnector_Click()*. V události se vytvoří instance třídy *TKukaSender*, do které jsou pomocí metod Set nastaveny parametry pro komunikaci. Vyčtení textových řetězců z textových polí pro nich určených probíhá stejným způsobem jako u ukládání parametrů po ukončení aplikace.

Operace vyvolané stisknutím tlačítka START

Stisknutím tlačítka *START* se rozhodne, zda bude požadavek odeslán pouze jednou nebo opakovaně. Pro nastavení opakovaní je použito zaškrtávací políčko a jeho vlastnost *CheckBox.Checked*. Tato vlastnost vyčte, zda bylo políčko zaškrtnuto a vrátí pravdu, v opačném případě vrátí nepravdu. Po rozhodnutí opakování dojde k vyvolání metody *SendRequest()*.

Operace vyvolané stisknutím tlačítka Reset Connector

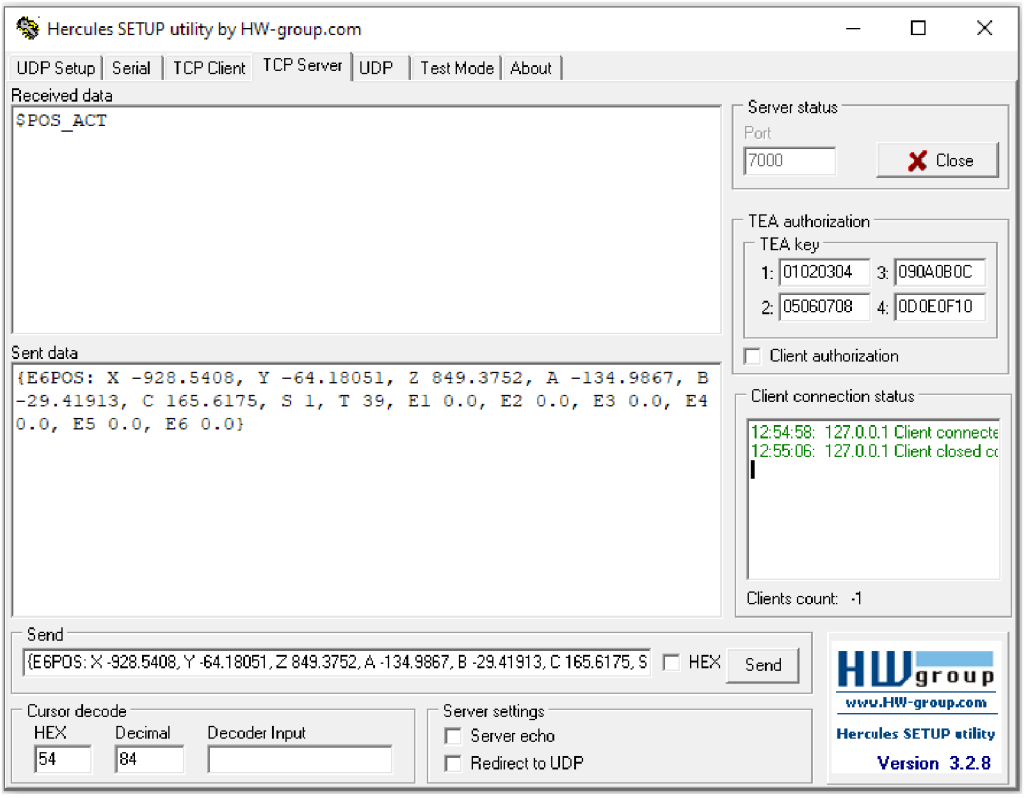
Stisknutím tlačítka *Reset Connector* se vyvolá metoda *FlushConnector*, která ukončí vykonávání vlákna pro komunikaci. Parametry vytvořené instance třídy jsou nastaveny do výchozí hodnoty proměnných typu nulového odkazu.

Akce vyvolané na textový polích

Textová pole fungují pouze pro vyčítání a zapisování parametrů pro komunikaci. Veškeré operace na nich vykonávané jsou spouštěny tlačítky a jimi spouštěnými metodami.

5.5 Testování aplikace

Aplikace Hercules



Obrázek 5.4: Terminál Hercules s připojenou aplikaci pro komunikaci

Ladění aplikace je provedeno pomocí freewarové aplikace Hercules. Aplikace poskytuje mnoho možností použití (sériový terminál, UDP...), TCP Server poskytl pro autora nejzajímavější funkce. Aplikace je přehledně rozdělena a v každém okamžiku byla možnost sledovat, co se v připojení děje. V oknech aplikace je vidět kdy se komunikační aplikace připojila a odpojila, dále zobrazuje data serverem přijatá a odeslaná.[19]

Aplikace Hercules byla použita také z bezpečnostních důvodů. Při prvních spuštěních kódu byla na server odesílána nesmyslná data, která by mohla robotu uškodit. Výhodou Hercula je jeho přenositelnost a jednoduchost použití. Autorova aplikace mohla být zkoušena na místech s přípojením k Internetu bez nutnosti přímé účasti robota.

Měření odezvy robota

Pro správnou funkci aplikace bylo důležité změřit i její odezvu, respektive odezvu serveru KukavarProxy na požadavek zaslaný uživatelem. Měření odezvy bylo realizováno třídou *Stopwatch*, která byla inicializována před odesláním požadavku a zastavena při přijmutí odpovědi.

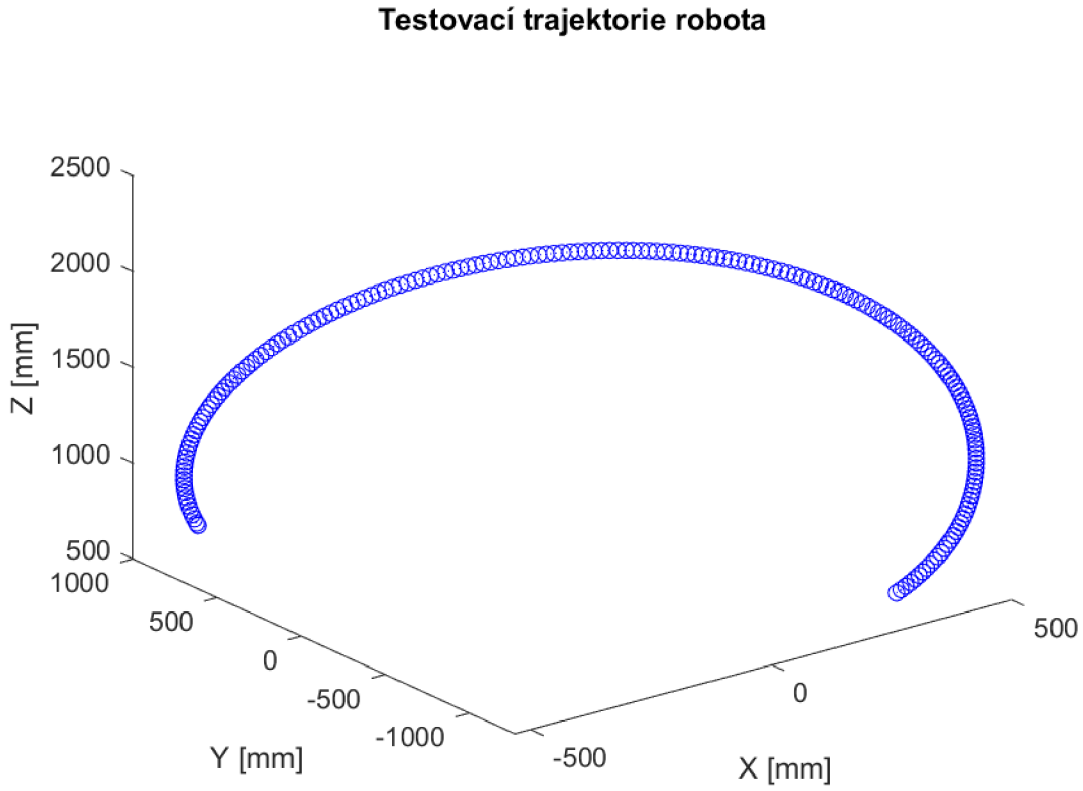
Tabulka 5.2: Tabulka dat přijatých aplikací pro komunikaci

X [mm]	Y [mm]	Z [mm]	A [°]	B [°]	C [°]	ODEZVA [ms]
307.3490	-1139.912	613.0444	-132.3377	-11.41061	-156.8258	5,1322
313.1039	-1151.238	630.5124	-133.1892	-12.29605	-155.0395	4,6754
318.9886	-1162.469	648.9775	-134.0938	-13.15786	-153.1617	3,5775
324.9881	-1173.546	668.4591	-135.0514	-13.98960	-151.1910	3,6658
330.6833	-1183.700	687.5952	-135.9939	-14.73373	-149.2648	8,9462
336.0796	-1192.986	706.3351	-136.9170	-15.39640	-147.3871	5,076
340.9887	-1201.140	723.9247	-137.7825	-15.96196	-145.6319	4,016
346.0010	-1209.166	742.4471	-138.6914	-16.50130	-143.7912	3,6009
351.4752	-1217.571	763.3661	-139.7134	-17.04438	-141.7215	4,8379
356.4571	-1224.875	783.0729	-140.6701	-17.49492	-139.7807	4,9339
361.1461	-1231.435	802.2458	-141.5938	-17.87902	-137.9009	3,3833
365.8992	-1237.755	822.3459	-142.5531	-18.22680	-135.9395	4,9855
370.6993	-1243.778	843.3795	-143.5453	-18.53319	-133.8973	4,7572
375.3641	-1249.263	864.5902	-144.5323	-18.78521	-131.8491	3,3547
379.8898	-1254.209	885.9628	-145.5112	-18.98389	-129.7970	4,7715
384.2726	-1258.619	907.4822	-146.4793	-19.13037	-127.7430	4,0403
388.3602	-1262.364	928.3582	-147.4002	-19.22336	-125.7626	4,3525
392.1649	-1265.501	948.5647	-148.2731	-19.26929	-123.8574	4,0344
395.8377	-1268.180	968.8614	-149.1303	-19.27360	-121.9555	3,6654
399.6427	-1270.555	990.8072	-150.0338	-19.23313	-119.9127	5,0366
403.2888	-1272.404	1012.831	-150.9148	-19.14748	-117.8770	4,1089
406.8946	-1273.766	1035.708	-151.8013	-19.01297	-115.7777	3,4699

Z naměřených dat byla vypočítána průměrná odezva, která dosáhla hodnoty 4,6 ms. Při měření odezvy byl souběžně vypočítán největší rozdíl mezi jednotlivými odezvami, který činil 5 ms. Získáním informací o odezvě byly zjištěny limity aplikace.

Hlavní omezení aplikace tkví v kolísající odezvě, která zapříčinila znemožnění použití aplikace pro úlohy, pro které je potřeba znát přesně definovanou odezvu. Aplikaci je možné použít na vyhodnocení obrazu z kamerového systému, nelze ji však použít k přímému ovlivňování trajektorie robota za jeho běhu nebo ovlivňovat jeho vstupy a výstupy externím počítačem. Pomocí naměřených dat byl vytvořen graf testovací trajektorie.

Zobrazení testovací trajektorie



Obrázek 5.5: Zobrazení testovací trajektorie

Pro zobrazení testovací trajektorie byl vytvořen program v aplikaci MATLAB. Účelem vynesení trajektorie bylo zobrazit trajektorii robota, jejíž body byly zaznamenávány vytvořenou aplikací pro komunikaci. Pro vynesení trajektorie stačila pouze odsazení os od počátku, tedy souřadnice X, Y a Z. Vyčítáním proměnné *$POS_ACT* se získá informace o aktuální pozici nástroje. Výše vynesená trajektorie zobrazuje pohyb nástroje v prostoru okolo robota.

6 Závěr

Cílem bakalářské práce bylo vytvoření testovací aplikace pro komunikaci pro komunikaci nadřazeného systému a průmyslového robota KUKA.

V teoretické části práce bylo pojednáváno o modelu TCP/IP, byly popsány základní charakteristiky jednotlivých vrstev modelu. Dále se práce zabývala problematikou použití volně dostupného rozšíření KukavarProxy, kde bylo rozebráno pozadí vzniku rozšíření a jeho funkce. Popsáním Ethernet KRL byl pokryt způsob komunikace, kterou nabízí firma vyrábějící roboty KUKA.

Praktická část bakalářské práce se zabývala vývojem a testováním aplikace pro komunikaci nadřazeného systému a průmyslového robota KUKA. Po seznámení se s možnostmi, kterými je robot schopen komunikovat přes síťové rozhraní Ethernet, začal návrh logického řešení problému. Na začátku byla vytvořena jednoduchá konzolová aplikace, která se spojila s robotem a ověřila tak fungování serveru KukavarProxy. Problémem v logické struktuře bylo zakódování zprávy pro server. Problém byl v definované struktuře požadavku pro KukavarProxy, která požaduje většinu parametrů zprávy ve formě dvou bitů. Logickou úvahou se došlo k závěru použit bity, které reprezentují nejvyšší a nejnižší číslo v binárním vyjádření parametru požadavku. Pro nejvyšší číslo byl použit bit MSB a pro číslo nejnižší bit LSB. Ovládání demonstrační aplikace bylo realizováno použitím více vláken aplikace. Dále byla provedena analýza dat přijatých demonstrační aplikací. Analýza a vynesení dat do grafu potvrdily správnou funkci vyčítání hodnot proměnných. Analýzou dat byla zjištěna průměrná doba příchodu odpovědí na uživatelův požadavek.

Při tvorbě grafického rozhraní bylo docíleno spojení jednoduchosti a přehlednosti rozhraní. Pro uživatele aplikace byly vytvořeny nápovědy, které pomohou k lepší orientaci v rozhraní. Cíle stanovené v úvodu bakalářské práce byly úspěšně splněny.

Pro budoucí vylepšení lze programovou část doplnit o třídu inicializačních souborů, které by zjednodušily zpětné načítání zadaných parametrů do textových polí pro ně příslušné. Grafické rozhraní by bylo možné zdokonalit jak po stránce vzhledové, tak prvkové. KukavarProxy podporuje také zápis proměnných a v dalším vylepšování aplikace dojde k rozšíření o možnost posílat data do robota pomocí demonstrační aplikace, například nastavit robotu požadovanou trajektorii.

Literatura

[1] *Rodina protokolů TCP/IP* [online]. Jiří Peterka, 2015 [cit. 2022-05-19]. Dostupné z: https://www.earchiv.cz/anovinky/ai1592.php3.

[2] *Transmission Control Protocol (TCP)* [online]. Pamela Fox, 2022 [cit. 2022-05-19]. Dostupné z: https://cs.khanacademy.org/computing/informatika-pocitace-a-internet/x8887af37e7f1189a:internet/x8887af37e7f1189a:tcp-protokol/a/transmission-control-protocol--tcp.

[3] *Jaký je rozdíl mezi TCP a UDP?* [Online]. TheFastCode, 2017 [cit. 2022-05-19]. Dostupné z: https://www.thefastcode.com/cs-czk/article/what-s-the-difference-between-tcp-and-udp.

[4] *Vrstvy TCP/IP* [online]. Richard Liska, 2018 [cit. 2022-05-19]. Dostupné z: http://kfe.fjfi.cvut.cz/~liska/unix/node18.html.

[5] *Comparison of KVP and RSI for Controlling KUKA Robots Over ROS* [online]. Mathias Hauan Arbo, Ivar Eriksen, Filippo Sanfilippo, Jan Tommy Gravdahl, 2020 [cit. 2022-05-19]. Dostupné z: https://www.researchgate.net/publication/344777582_Comparison_of_KVP_and_RSI_for_Controlling_KUKA_Robots_Over_ROS.

[6] *RTX* [online]. IntervalZero, 2022 [cit. 2022-05-19]. Dostupné z: https://www.intervalzero.com/en-products/en-rtx64/.

[7] *KukavarProxy* [online]. Lionel du Peloux a Massimiliano Fago, 2019 [cit. 2022-05-19]. Dostupné z: https://github.com/ImtsSrl/KUKAVARPROXY.

[8] *Controlling Kuka Industrial Robots: Flexible Communication Interface JOpenShowVar* [online]. F. Sanfilippo, L. I. Hatledal, H. Zhang, M. Fago, K. Y. Pettersen, 2015 [cit. 2022-05-19]. Dostupné z: http://filipposanfilippo.inspitivity.com/publications/controlling-kuka-industrial-robots-flexible-communication-interface-jopenshowvar.pdf.

[9] *System Variables* [online]. KUKA Roboter GmbH, 2012 [cit. 2022-05-19]. Dostupné z: http://www.wtech.com.tw/public/download/manual/kuka/krc4/KUKA%5C%20System%5C%20Variables%5C%208.1%5C%208.2%5C%208.3.pdf.

[10] *KST-Ethernet-KRL-21-En datasheet* [online]. KUKA Roboter GmbH, 2012 [cit. 2022-05-19]. Dostupné z: http://www.wtech.com.tw/public/download/manual/kuka/krc4/KST-Ethernet-KRL-21-En.pdf.

[11] *PROFINET* [online]. Neznámý, 2021 [cit. 2022-05-19]. Dostupné z: https://cs.wikipedia.org/wiki/PROFINET.

[12] *EtherCAT* [online]. Neznámý, 2022 [cit. 2022-05-19]. Dostupné z: https://en.wikipedia.org/wiki/EtherCAT.

[13] *ProfiBus* [online]. Neznámý, 2022 [cit. 2022-05-19]. Dostupné z: https://cs.wikipedia.org/wiki/Profibus.

[14] *Microsoft Visual Studio* [online]. Neznámý, 2021 [cit. 2022-05-19]. Dostupné z: https://cs.wikipedia.org/wiki/Microsoft_Visual_Studio.

[15] *Co je OOP?* [Online]. Education-WIKI.com, 2022 [cit. 2022-05-19]. Dostupné z: https://cs.education-wiki.com/5870796-what-is-oop.

[16] *Programovací jazyk C#* [online]. Marek Běhálek, 2022 [cit. 2022-05-19]. Dostupné z: http://www.cs.vsb.cz/behalek/vyuka/pcsharp/text.pdf.

[17] *Nejvýznamnější bit* [online]. Neznámý, 2021 [cit. 2022-05-19]. Dostupné z: https://cs.wikipedia.org/wiki/Nejv%5C%C3%5C%BDznamn%5C%C4%5C%9Bj%5C%C5%5C%A1%5C%C3%5C%AD_bit.

[18] *Nejméně významný bit* [online]. Neznámý, 2021 [cit. 2022-05-19]. Dostupné z: https://cs.wikipedia.org/wiki/Nejm%5C%C3%5C%A9n%5C%C4%5C%9B_v%5C%C3%5C%BDznamn%5C%C3%5C%BD_bit.

[19] *Aplikace Hercules SETUP* [online]. HW group, 2022 [cit. 2022-05-19]. Dostupné z: https://www.hw-group.com/cs/software/aplikace-hercules-setup.

A Přílohy

- Celý projekt Komunikace (Visual Studio 2022) (Komunikace.zip)