Univerzita Hradec Králové
Fakulta informatiky a managementu
Katedra informačních technologií

Distribuované robotické systémy

Distributed Robotic Systems

Diplomová práce

Autor: David Voda
Studijní obor: ai2-k

Vedoucí práce: Ing. Karel Mls Ph.D.

Hradec Králové Srpen 2022

Prohlášení:

Prohlašuji, že jsem diplomovou práci zpracoval samostatně a s použitím uvedené literatury.

vlastnoruční podpis

V Hradci Králové dne 15. 8. 2022

Poděkování:

Děkuji vedoucímu diplomové práce Ing. Karlu Mlsovi Ph.D. za metodické vedení práce, odborné rady a zapůjčení potřebných zařízení.

Anotace

Diplomová práce seznamuje čtenáře s principy distribuovaných systémů se zaměřením na využití v oblasti robotiky. V textu jsou představeny principy, které obecně využívají distribuované systémy a které jsou specifické například pro využití v robotickém světě. Velký důraz je v dnešní době kladen na moderní přístupy využívající například cloudové řešení nebo nasazení systému ve virtuálním prostředí. Dále je v práci zmíněno zatím experimentální využití technologie blockchain v robotice a její srovnání s frameworkem ROS2. V praktické části je nastíněna ukázka a ověření fungování distribuovaného robotického ekosystému pomocí frameworku ROS2 (*Robot Operating System*) na rozdílných fyzických zařízení. ROS2, jakožto robotický systém, je prozkoumán i v teoretické části, kde jsou popsány jeho jednotlivé komponenty a nástroje, které umožňují s ním pracovat.

Annotation

Title: Distributed Robotic Systems

The diploma thesis introduces the reader to the principles of distributed systems with a focus on use in the robotic field. The text starts with distributed principals in general and then moves to specific principals of the robotic world. In today's world the main focus is on modern principals like cloud technology or deployment in virtual environments. The thesis also includes part about experimental use of blockchain technology in a field of robotics and comparation with ROS2 framework. There is a practical example of the robotic framework functionality specifically ROS2 (*Robot Operating System*) working on different physical hardware. The theoretical part contains fundamentals of the *Robot Operating System*, especially its tools and components.

Klíčová slova

ROS2, ZED2, Nvidia Jetson, Raspberry Pi, Robotika, Robot, Docker. Blockchain

Keywords

ROS2, ZED2, Nvidia Jetson, Raspberry Pi, Robotics, Robot, Docker, Blockchain

Obsah

1 Úvod

Dnešní svět nabízí obrovské množství různých možností a nápadů ve všech směrech lidského fungování. Robotika nás doslova obklopuje a stává se zásadní ve velkém výčtu vědních oborů. V posledních několika letech se ale její povědomí dostává více do rukou laiků, kteří díky snadné dostupnosti softwaru, hardwaru a open-source projektů, nemusí být nutně odborníky v oboru pro tvoření robotických zařízení. Díky rozšíření internetu do téměř všech koutů světa vznikají komunity sdílející své nápady a objevy. Cílem práce je představit některé z mnoha principů distribuované robotiky, která sama o sobě vychází z principu distribuovaného systému a nahrazuje tak systém centralizovaný, který skrývá značné nevýhody ve své implementaci. V úvodní kapitole je věnována část historii robotiky, aby si čtenář mohl představit, jak bylo k tématu přistupováno v historii. Dále je obecně popsáno fungování distribuovaného systému, z něhož vycházení distribuované robotické systémy. Ke konci kapitoly jsou zmíněny nejnovější přístupy, které se v robotice mohou využívat. Následuje porovnání technologie blockchain jako způsobu komunikace s frameworkem ROS2. Praktická část se zaměřuje na fungování konkrétního robotického systému ROS2 (*Robot Operating System*) v jeho druhé verzi. Přestože ROS má v názvu operační systém, jeho fungování by se více dalo přiblížit k *frameworku*, neboť k jeho životu potřebuje jiný hostitelský operační systém. Praktická část ukazuje na modelové situaci, jak ROS funguje a popisuje základní kroky k jeho zprovoznění. Závěr práce je věnován shrnutí práce s *frameworkem*, ověření jeho fungování a možnost využití moderních principů, které se v robotice objevují.

2 Robotika

Robotika se dá popsat jako vědní obor, disciplína nebo oblast výzkumu, zabývající se tvorbou, návrhem a aplikací inteligentních strojů pro zjednodušení lidského fungování v běžném životě nebo v činnostech, které jsou pro člověka fyzicky nemožné. Příkladem robota tedy může být jednoduchý kávovar, ale i komplexní stroj určený pro pohyb po planetě Mars.

V této úvodní kapitole je popsána historie robotiky a konkrétní příklady z různých odvětví, kde se robotika využívá. Dále je vysvětlen pojem distribuované

robotiky, proč je dnes tak populární a její základní principy. Konec kapitoly je věnován moderním technologiím, které jsou využívány i v oblasti robotiky jmenovitě například cloudové řešení.

2.1 Historie robotiky

Robotika jako vědní obor vznikl až ve 20. století. Vznik názvosloví je pro nás Čechy velmi blízké, neboť označení *robot* je ve světě nejznámější české slovo. Použil ho poprvé v roce 1921 Karel Čapek, rodák z Královehradeckého kraje, ve své divadelní hře R.U.R (celým názvem *Rossumovi univerzální roboti*). Označení mu poradil jeho bratr Josef Čapek, když se bavili, jaké označení pro umělé inteligentní bytosti použít. Myšlenka věcí samostatně pracujících nebo provádějící činnosti bez potřeby lidského zásahu sahá již do období antiky, zmínky můžeme najít například v díle Politika od Aristotela.

„Neboť kdyby každý nástroj na rozkaz nebo již předem dovedl vykonati své dílo, jak se vypravuje o Daidalových sochách nebo o Héfastových trojnožkách, o kterých básník pěje, že samy od sebe tkaly paličky, hrály na kitharu, nepotřebovali by stavitelé pomocníků ani páni otroků“. [1]

Dalším příkladem o několik staletí později jsou mnohé vynálezy od italského myslitele Leonarda da Vinciho nazývaný *Leonardův Robot* nebo *Leonardův mechanický rytíř*. Robot měl vzniknout na základě Leonardových studií lidského těla a funkční model byl údajně představen na oslavách Milánského vévody v roce 1495. Tento mechanický rytíř měl umět na základě mechanismu páček, lanek, kladek a ozubených kol provést jednoduché lidské pohyby. Jmenovitě si uměl sednout, vstát, volně hýbat pažemi a zvednout si hledí přilbice. [2]

O několik desítek let později na Pražském dvoře císaře Rudolfa II. údajně vznikla další ukázka pojetí robota, a to hliněný Golem. Samotná legenda o oživlé hliněné postavě sahá však až do 12. století, tedy ještě před da Vinciho mechanickým rytířem. [2]

Modernější pojetí robotů tak, jak je známe dnes, přichází až ve 20. století. Definici pojmů „robot“ a „robotika“ uvedl například McKerrow v roce 1986. [3]

„Robot je stroj, který může být naprogramován k vykonávání různých činností.“

Robotika je disciplína zahrnující:

1. Návrh, výrobu, řízení a programování robotů.
2. Použití robotů pro řešení úloh.
3. Zkoumání řídicích procesů, senzorů, akčních členů a algoritmů u lidí, zvířat a strojů.
4. Použití výše uvedeného pro návrh a použití robotů.

V roce 1950 definoval americký spisovatel Isaac Asimov ve svých sci-fi povídkách (Nadace, Já robot) tři zákony robotiky (*Asimovy zákony*). [4]:

1. Robot nesmí napadnout lidskou bytost.
2. Robot musí poslouchat příkazy lidí, pokud to neodporuje pravidlu číslo 1.
3. Robot musí chránit sám sebe, svoji existenci, pokud to neodporuje pravidlům číslo 1 a 2.

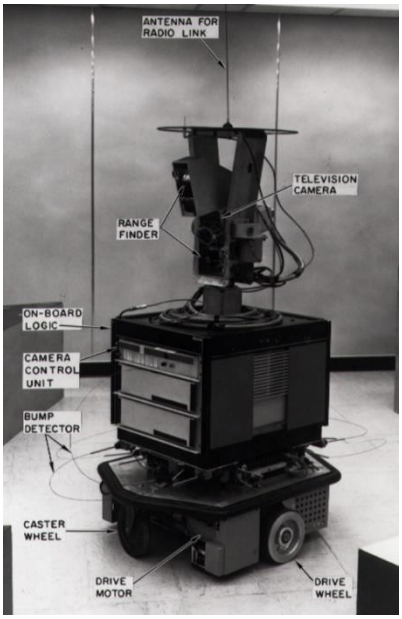
Další příklady robotických vynálezu 20. století chronologicky jdoucí za sebou:

1951 – Raymond C. Goertz navrhnul a zkonstruoval robotická ramena, která mohl operátor vzdáleně ovládat a pracovat s nebezpečným radioaktivními materiály. Goertz se také zasloužil o zapsání termínů *pitch*, *yaw* a *roll* do robotického slovníku. Tyto pojmy definují rotaci v 3D prostoru podle os XYZ. [5]

1961 – General Motors zkonstruovalo prvního industriálního robota s názvem *Unimate*. Robot měl za úkol přesouvání odlitků a sváření těchto částí k automobilové konstrukci. Jednalo se o poměrně nebezpečnou operaci pro člověka, neboť při sváření docházelo ke vzniku toxických výparů. Robot vypadal jako velké robotické rameno usazené na obdélníkovém boxu viz obrázek č. 1. Z názvu robota následně vznikla první světová robotická společnost *Unimation*, která později vyráběla další industriální roboty. [6]

Obrázek 1: Unimate robot
Zdroj: [6]

1968 – *Shakey* je přezdívka prvního mobilního robota schopného reagovat na podněty okolí bez nutného lidského řízení. Robot byl vybaven kamerou, čidlem vzdálenosti a vlastním pohonem. Programovací jazyk byl primárně LISP a projekt využíval softwarové metody jako A* algoritmus pro hledání optimálních cest nebo Houghovu transformaci pro analýzu obrazu. [7]

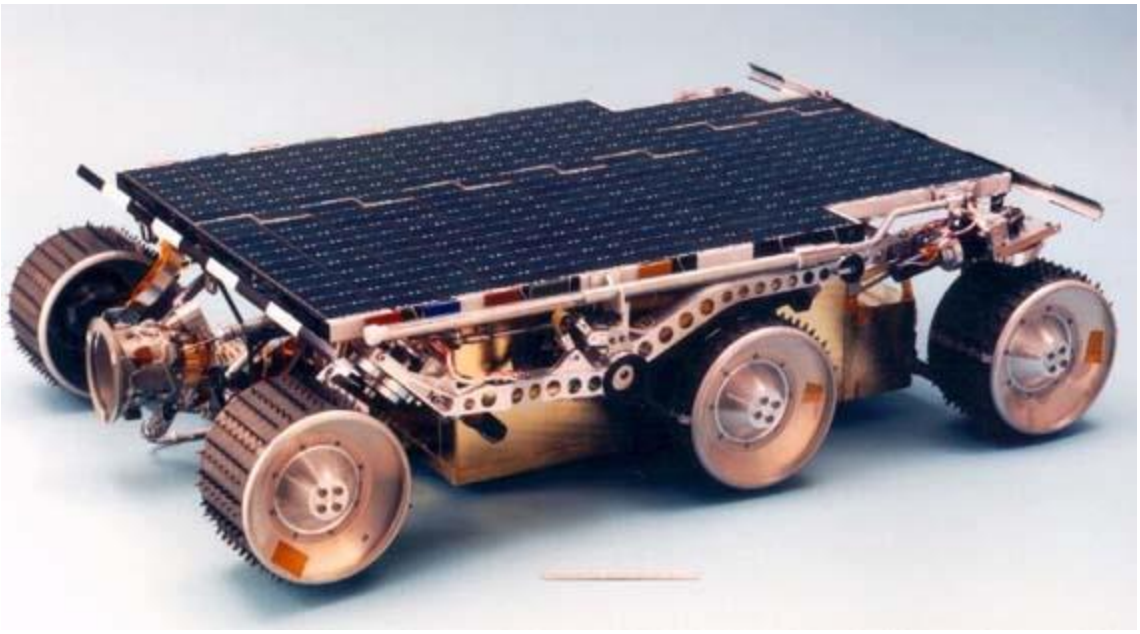


Obrázek 2: Shakey robot
Zdroj: [7]

1994 – *CyberKnife* v překladu kybernetický nůž je název pro robotické zařízení sloužící v lékařském prostředí pro léčbu nádorových ložisek. Stroj vysílá do lidského těla s extrémní přesností vysoké dávky radioaktivního záření, kterým efektivně ničí zhoubné buňky. Umožňuje tak léčit nádory po celém těle a tím nabízí neinvazivní alternativu chirurgických zákroků. *CyberKnife* vznikl jako projekt na Stanfordské univerzitě před více než 20 lety a dnes je dostupný i v České republice, konkrétně v Praze a Ostravě. [8]

Obrázek 3: CyberKnife
Zdroj: [9]

1997 – *Sojourner* bylo speciální vozítko neboli *Rover*, který byl součástí mise *Mars Pathfinder*. Tento rover bylo první vozidlo, které se pohybovalo na jiné planetě mimo Zemi (neplést si s Měsícem, který není kategorizován jako planeta). Cílem *Sojouneru* bylo primárně otestovat prostředí na Marsu a naše lidské schopnosti navrhnout zařízení schopné se v takovém neprozkoumaném prostředí pohybovat a interagovat. Robot byl navržen na fungování 7 dnů na Marsu, překonal však veškerá očekává a funkční byl 83 marťanských dní (85 dnů na Zemi). [10]



Obrázek 4: Sojourner, robot na Marsu
Zdroj: [10]

Jak můžeme vidět z výše uvedených příkladů, robotika se již od druhé poloviny 20. století prokázala jako velmi cenný vědní obor, který usnadňuje práci lidem nejen v inženýrských oborech a umožňuje dosažení splnění dříve nepředstavitelných cílů, jakým je například jízda po povrchu Marsu. Robotika pomáhá lidem i v oblasti zdraví – dá se využít nejen při stanovování diagnózy, ale i při samotné léčbě. Tohle vše jsou ale příklady z minulého století. Robotické zařízení se od přelomu století posunuly o několik stupňů výše.

2.2 Robotika 21. století

V současném době je příkladů robotických vynálezů velké množství. Pro příklad jsou zde uvedeny zajímavá či v určitém směru revoluční řešení.

Mars 2020 – Z názvu již napovídá, že jde opět o robotické zařízení, konkrétně rover *Perservance* a malého drona či helikoptéru *Ingenuity*, vyslané v roce 2020 na planetu Mars. Od svého předchůdce *Sojournera* se ale technologie výrazně posunula a fungují na planetě již od roku 2021 do současnosti. [11]

GigaFactory – Elektromobilový gigant Tesla staví obrovské továrny na jednotlivé součástky do svých vozidel. Přestože tyto obří komplexy zaměstnávají spoustu lidí, jejich chod je zajištěn do velké míry automatizovanou prací pomocí nejrůznějších robotů. Různorodost takových zařízení je zde obrovská, od robota převážející materiály z bodu A do bodu B, až po komplexní robotické rameno, které konkrétní části buduje z menších součástí. Posledním přírůstkem do rodiny těchto továren je komplex nedaleko Berlína. Jedná se o první Tesla továrnu takových rozměrů v Evropě. [12]

Exoskeletons – V překladu externí kostra je pojem, který již není pouhou fikcí z filmových pláten, ale skutečností, která je využívaná hned v několika odvětví. Jmenovitě například ve zdravotnictví, při manipulaci s těžkými předměty anebo v armádě. Tyto pomocné kostry napomáhají člověku při chůzi například po zranění a následně během rehabilitace. Při zvedání těžkých předmětů zase dokážou mnohonásobně zvýšit lidskou sílu či výdrž. Exoskeletony mají hned několik podob, od jednoduchých (skoro až neviditelných) návleků na oblečení, až po mohutné konstrukce kopírující tvar celého těla. [13]

Obrázek 5: HAL Exosketelon
Zdroj: [14]

2.3 Robotické aplikace

S robotickými zařízeními se v dnešní době setkáváme na denní bázi, často už ani nevnímáme, že za fungováním určitého stroje stojí právě robot. Většinou si pod pojmem robot představíme nějakého humanoida složeného z plastu, kovů a spousty elektroniky. Robotická zařízení ale nemusí nutně být založena na fungování s umělou inteligencí a poskytovat komplexní a složité služby. [15]

Příklady využití moderní robotiky jsou:

- Zdravotnický průmysl: Operace, rehabilitace, asistence při komplikacích.
- Industrializace: Logistika, skladování a rozvoz zboží, výrobní linky.
- Služby: Robotičtí operátoři.
- Vojenská oblast: Práce s výbušninami, převoz zbraní, vzdálený monitoring.
- Záchranné a pátrací mise: Průzkum území, záchrana z nedostupných oblastí.
- Objevování pro člověka obtížně dostupných míst: Vesmír, hlubiny oceánů.
- Školství: Výuka předmětů, pomůcky pro výuku.

2.4 Distribuované systémy

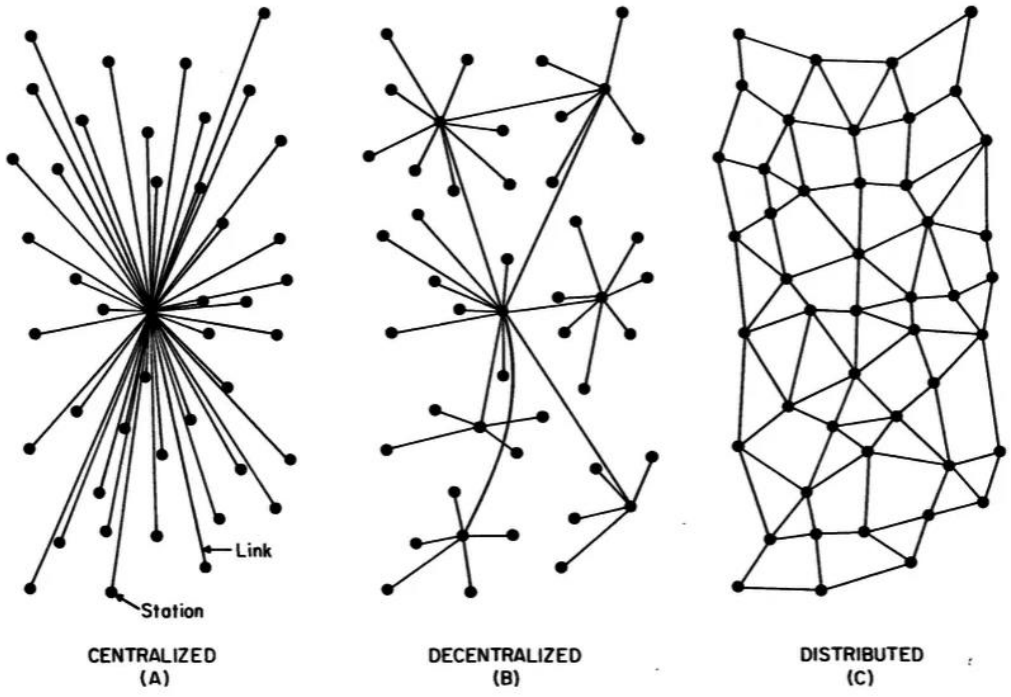
V této kapitole jsou představeny základní i moderní principy robotiky, na kterých lze stavět návrh nových systémů. Distribuovaný systém je alternativou k systémům centralizovaným a decentralizovaným. Principy fungování těchto tří systémů se využívají nejen v počítačovém světě, ale i v každodenním lidském životě. V moderním technickém světě se čím dál častěji využívají právě distribuované systémy z důvodů jejich výhod, které si popíšeme.

Princip fungování centralizovaného systému je jednoduchý. Máme nějaký centrální bod, který řídí ostatní připojené části systému. Základní bod tedy musí být robustní, aby dokázal pracovat s velkým množstvím připojených bodů. Veškerá data a informace zpracovává jeden centrální systém. Pokud tedy dojde k jeho výpadku, celá síť přestane fungovat a zkolabuje.

Hlavní velkou nevýhodu jednotného centrální bodu v centralizovaném systému eliminuje decentralizovaný systém. V takovém návrhu existuje více řídících bodů, které

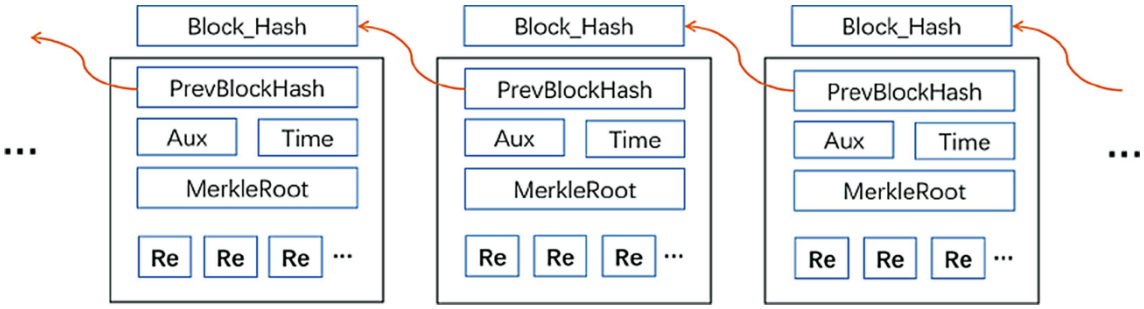
procesují informace od menších systémů, přímo připojených k hlavním bodům. Tyto řídící body jsou mezi sebou propojeny, často více než jednou cestou, a komunikují mezi sebou informace získané od menších celků. Pokud tedy dojde k výpadku jednoho řídícího bodu, ostatní můžou dále fungovat a nenastane celkový kolaps systému.

Na principu decentralizace funguje například síť *Internet*, kde jednotlivé řídící body existují na několika úrovních. Příkladem je LAN síť s jedním výstupním bodem do internetu, ten se napojuje k jednomu z bodů poskytovatele internetu a napojuje se na další body i jiných poskytovatelů. Na obrázku číslo 6 můžeme uprostřed vidět, jak vypadá schéma takového systém nebo sítě. Posledním typem je systém distribuovaný, který je pro tuto práci stěžejní. Je velmi podobný decentralizované síti, principiálně zde neexistují žádné řídící body, ale všechny koncové body jsou plně samostatné a připojí se libovolně na ostatní prvky v systému podle potřeby či možností. Jeden bod tedy teoreticky může být připojen ke všem ostatním prvkům v daném ekosystému. Typický příklad takového systému jsou peer-to-peer sítě, kde každý klient komunikuje s ostatními bez pomoci řídícího prvku nebo služby. Na principu distribuovaného systému fungují také v posledních letech velmi populární kryptoměny jako *Bitcoin* a jeho další alternativy. [16]



Obrázek 6: Schéma centralizovaného, decentralizovaného a distribuovaného systému
Zdroj: [16]

Jedním z příkladů distribuovaného a decentralizovaného systému je *Blockchain*. Tento systém je založen na datových blocích spojených v nekonečném kruhu (řetězu) za sebou. Nové bloky dat se přidávají do řetězu, a ten tak neustále roste. Řetězec těchto dat je chráněn kryptografickými algoritmy a samotným fungováním na *peer-to-peer* síti. *Blockchain* se používá jako úložiště platebních transakcí pro *Bitcoin* a další kryptoměny. [17]



Obrázek 7 Blockchain
Zdroj: [17]

2.4.1 Využití v robotice

Využití distribuovaných sítí a systémů je v robotice velmi efektivní. Představme si například situaci průzkumu neznámého terénu. K dispozici máme hned několik samostatných zařízení, které ale spolu navzájem komunikují a poskytují si cenné informace pro zjednodušení fungování. Průzkumný robotický tým je složen z leteckého dronu, jezdícího roveru a obojživelného stroje schopného fungovat ve vodě. Každý z těchto robotů prozkoumává okolí samostatně a podává informace o prostředí systému zpracovávající data, vytvářející například mapu a trojrozměrnou simulaci z kamer a senzorů. Každé jedno z těchto tří zařízení funguje samostatně a pokud dojde k přerušení datového signálu tak se může napojit na nejbližšího spolupracovníka a přeposílat informace přes něho, nebo se snažit dostat do pozice odkud spojení bude schopen obnovit. Roboti si dále předávají informace o prostředí a možných překážkách a dohromady jejich informace dávají možnost vytvořit komplexní představu o daném místě, kde jsou nasazeni.

Reálným příkladem z praxe je stále běžící mise na Mars provedena americkou společností NASA. Zde se robotický tým skládá hned z několika pojízdných strojů, létacích dronů a družic fungující na orbitu. Přestože byla tato robotická zařízení vyslána

na Mars v různých letech, jsou schopna spolu vzájemně efektivně spolupracovat a poskytovat si informace, které jsou následně odeslány na domovskou planetu. Společnost NASA využívá již několik let pro vývoj robotických zařízení *framework* ROS, který je založený na distribuované funkcionalitě. [18]

2.5 Distribuovaná robotika a cloud

Cloudové služby jsou dnes určitý standard a využívají se v mnoha odvětvích od cloudové zálohy, synchronizace uživatelských profilů, sdílení fotek mezi zařízeními (mobilní, desktopové), až po kompletní služby běžící v cloudu apod. Cloud nám nabízí abstraktní vrstvu, která sama zpracovává dynamické přidělování prostředků.

Tato technologie nám značně může pomoct právě v robotice. Mějme skupinu robotických zařízení sbírající data, která se následně vyhodnocují a sdílí mezi sebou pomocí bezdrátové komunikace. Veškeré zpracování i posílání těchto informací stojí robota určité prostředky jak výpočetní, tak napájecí. V ekosystému pohybových robotů, kteří jsou napájení z baterií, je potřeba brát v potaz kapacitu baterie a její maximální využití. V práci [19] je teoreticky popsán framework, který na základě jednoduchého výpočtu rozhodne, zda je pro robota výhodnější a efektivnější, aby získaná data zpracoval sám, sdílel s ostatními roboty ve skupině, nebo je odeslal do cloudu, který výsledky zpracuje a robotům následně poskytne jejich výsledek.

Rozhodování funguje na základě několika faktorů:

- Objem přenášených dat (přenos samotných dat je také výpočetně náročný).
- Výpočetní náročnost úkonu (vytváření 3D mapy je velmi náročné).
- Zdroj všech dostupných dat (pokud je nemáme, je lepší poslat moji část někomu jinému).

Typickým příkladem takového fungování může být SLAM (*Simultaneous localization and mapping*) nebo obyčejné mapování prostoru. V obou případech je výpočet a zpracování údajů ze senzorů a kamer poměrně náročné na výpočetní a datové kapacity robotických zařízení, proto je zde vhodné využít cloudové služby a té zaslat nashromážděná data. Následně proběhne samotný výpočet v cloudu, který má přístup k dynamicky alokovaným prostředkům. Díky tomu může zpracování dat proběhnout

téměř v reálném čase. Robot následně pracuje pouze s jednoduchou informací, jakou je například „jaký je jeho další úkol“ nebo „kam se má přesunout“. [20]

Obrázek 8: Schéma cloudové robotiky
Zdroj: [19]

Cloudové řešení nám přináší mnoho výhod:

- Samotná robotická zařízení nepotřebují být osazena výkonným hardwarem.
- Zařízením pracující na baterii se prodlužuje doba fungování mezi nabíjením.
- Pokud robot nezvládá zpracovávat výpočty, není potřeba ho ihned vyměnit za výkonnějšího, stačí přesunout zátěž do cloudu.

2.6 Multi-robotické systémy (MRS)

V předchozí kapitole jsme si představili možnost využití výpočetního výkonu v cloudové prostředí, které přináší spoustu výhod. V mnoha situacích ale tým robotických zařízení nemá možnost přistupovat k serverům, které jsou umístěny kdesi v oblacích. Důvodem může být ale i pomalé připojení k internetu, nestabilita připojení nebo nízká rychlost. Pokud máme například skupinu robotů provádějící svoji činnost ve venkovním prostředí, pokrytí signálu nemusí být stoprocentní nebo by bylo finančně nákladné.

V takovém případě je možné využít rovnoměrně všechny prostředky jednotlivých robotů ve skupině a náročné operace a výpočty rozložit mezi ně. Princip

takového rozložení zátěže spočívá v tom, že z množiny operací, které jsou potřeba vykonat jsou jednotlivé operace přiřazeny nejméně vytíženému robotovi ze skupiny a tím dochází k celkovému urychlení získání výsledku. Například technologie RAMP (*Reliable Autonomous Mobile Program*) vycházející z AMP (*Autonomous Mobile Program*) popisuje fungování takového systému, kde se skupina robotů dělí o práci na základě jejich vytížení. Tato technologie je podobná například *Load Balancingu* ze světa počítačových sítí a počítá i s možnostmi selhání ať už samotných robotů provádějících úkony či možnými problémy v lokální komunikaci na síti.

V praktickém využití technologie RAMP došli autoři k závěru, že ve vytíženém ekosystému robotů dojde k optimalizaci práce s danými úlohami oproti pevnému rozdělování na základě takzvaného systému *Round Robin* (každý robot dostane jeden úkol ze seznamu, pokud je seznam delší, nežli počet robotů tak se začíná rozdělovat znovu). V ekosystému, který vytížený není (situace, která je v realitě méně pravděpodobná) RAMP oproti *Round Robin* zaostává v malých jednotkách procent z důvodu, že samotné rozhodnutí, kde se má úkon spustit, stojí nějakou časovou jednotku, kdežto *Round Robin* úkony rovnou přiřazuje. [21, 22]

2.7 Robotické systémy a IoT

Robotika a IoT (*Internet of Things*) jsou dvě provázané oblasti, které spolu dokážou velmi dobře spolupracovat a doplňovat se. V posledních letech se dokonce začal používat i pojem IoRT (*Internet of Robotic Things*). Robotické zařízení dokážou jednoduše využívat data senzorů ze *smart* prostředí, ať už se jedná o chytrý dům, dílnu, továrnu nebo celé město. Dobrým příkladem může být autonomní robotické vozidlo zajišťující rozvoz jídel po městě. Takové vozidlo je schopné komunikovat s interním dopravním systémem města, efektivně plánovat trasu a vyhnout se případným dopravním zácpám.

V domácím prostředí lze pomocí robotiky zajistit bezpečnost zejména u nemocných nebo starších lidí. Senzor sledující životní funkce člověka je v tomto případě na lidské tělo umístěn v podobě náramku. Tento náramek je schopen detekovat pád člověka nebo zhoršení tepových funkcí a v případě potřeby automaticky přivolat pomoc. Domácí prostředí může dále obsahovat senzor kouře pro případ požáru, dveřní senzor, a hlavně pohybujícího se robota vybaveného několika senzory a kamerou pro

ověření aktuálního stavu osoby. Dalším příkladem může být robotický opatrovník, který sleduje starší či nemocné lidi, kteří tráví čas v domácnosti o samotě. Všechna tato zařízení dokážou spolu navzájem komunikovat a předávat si svá data pro lepší fungování celku. [23]

2.8 Výhody a nevýhody distribuované robotiky v praxi

Vezmeme-li si příklad například robotické domácnosti, můžeme velmi zřetelně vidět jednotlivé klady a zápory distribuovaného robotického systému. Příkladem distribuované robotiky je několik menších zařízení, kde má každé za úkol malý počet specifických úkolů. Skládá se z robotického vysavače, inteligentního topení, kamerového systému. Všechny tyto zařízení fungují samostatně, ale o svá data a informace se mohou podělit s ostatními zařízeními. Je možné přikupovat nová zařízení, která stále budou fungovat i s těmi starými…. Oproti tomu příkladem centralizovaného systému je robotický android, který funguje jako člověk. Umí luxovat, mýt nádobí, zalévat květiny a další činnosti, ale vždy jen jednu naráz. Jeho upgrade však bude velmi nákladný a pravděpodobně bude potřeba zakoupit celý nový model místo zakoupení samostatných rozšiřujících modulů. [19]

Výhody:

- Škálovatelnost: Jednotlivé robotické zařízení na specifické úkoly si můžeme pořizovat nezávisle na sobě. Pokud jeden robot na činnost nedostačuje, můžeme pořídit dalšího.
- Nízké vstupní náklady: Jeden malý robot bude levnější než komplexní robotický humanoid.
- Robustnost: Pokud se jeden z robotů rozbije, ostatní můžou fungovat dále bez jeho kooperace.
- Kooperace různých robotů od odlišných výrobců
- Náklady na opravu.
- Rychlost práce.
- Nahraditelnost.

Nevýhody:

- Složitost implementace: Distribuovaný systém je jednoznačně složitější na vzájemnou komunikaci mezi roboty a potřeba uživatelského zásahu může být s každým robotem zvlášť.
- Celková cena více robotů obstarávající jednu činnost: Cena komplexního robota je nižší, součástky pro jednotlivé úkony jsou v celkovém systému pouze jedny a náklady se tak snižují.
- Vysoké nároky na uživatelské porozumění: uživatel musí rozumět všem robotům, jak fungují, jak je nastavit a jak je udržovat. Při vyšším počtu zařízení je celkové pochopení systému pro běžného uživatele složité.
- Nekompatibilita: Nově přikoupený robot nemusí umět komunikovat s ostatními zařízeními.

2.9 Shrnutí

V této kapitole jsme se dozvěděli základní principy distribuované robotiky. Distribuované systémy nejsou převratnou novinkou v digitálním světě a za poslední desetiletí už ani v běžném každodenním životě. Příkladem distribuovaných sítí jsou člověkem už od začátku strukturalizace civilizace využívané cesty mezi městy, v informačním světe na tomto principu fungují technologie jako internet.

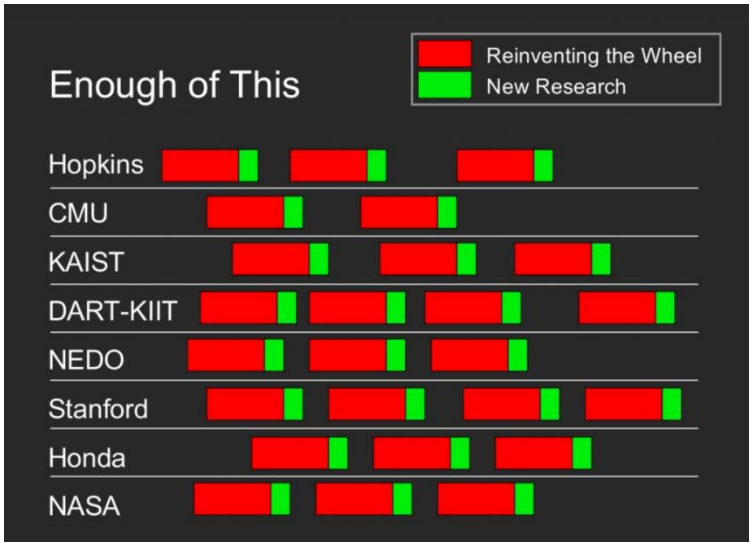
Pojem distribuovaná robotika lze vnímat jako určitý ekosystém několika robotů viz. příklad průzkumných sond vyslaných na Mars. Není to ale nutně pravidlem, neboť distribuce může fungovat i v rámci jednoho robotického celku. Očividným příkladem může být robotický člověk, kde každá jedna jeho část (ruka, noha, oko) je samostatně fungující systém a při výpadku jednoho prvku nedojde ke kolapsu celku. Jednotlivé části také můžou komunikovat s prostředím přímo a není nutné mít centrální mozek jako má člověk, který musí zpracovávat veškeré informace sám.

3 Distribuované robotické architektury

Tato kapitola z velké části popisuje fungování vybraného open-source robotického *frameworku,* jehož princip je založený distribuované robotice. Vybraný framework nese název ROS2, tento framework však není na aktuálním trhu jediný. Existuje velká řada alternativ a je tedy potřeba ho brát pouze jako ukázkový příklad možného systému aplikující principy popsané v přechozí části práce. ROS byl vybrán právě pro jeho jednoduchost, dostupnost výukových materiálů, rozsáhlou dokumentaci a silnou podporou velké komunity uživatelů.

3.1 ROS2

Robot Operating System, zkráceně ROS je, jak již z anglického názvu napovídá, operační systém zaměřený na práci s robotickými stroji. Konkrétně fungování částí robota jako celku. Původní systém před nástupcem s označením číslem 2 vznikl jako projekt na Stanfordské univerzitě. Cílem projektu bylo zrušit neustálý cyklus vytváření základního balíčku pro fungování robotických zařízení, který obstarával běžnou komunikaci mezi jednotlivými součásti robota a dokázal koordinovat samotné fungování robotického zařízení. Nové projekty tedy musely vždy začínat na zelené louce a vytvářet tento základní programový balíček a implementovat stejné principy jako ostatní projekty před nimi. Myšlenkou bylo vytvořit předpřipravený framework, který by vývojářům toto obstaral a ti by se potom mohli zaměřit více na samotné fungování jejich konkrétního robota. Jak je vidět na obrázku číslo 9, tak více než tři čtvrtiny času stráveného na konkrétních projektech bylo vloženo do vývoje něčeho, co už někdo vymyslel a vytvořil, před začátkem projektu. Systém ROS zajistil vytvoření souhrnu balíčků pro konkrétně činnosti, hardware a senzory, které můžou ostatní uživatelé v rychlosti aplikovat na svůj vlastní projekt a ušetřit tím spoustu času.



Obrázek 9: Čas strávený na robotických projektech
Zdroj: [24]

3.1.1 Verze a Distribuce

Původní verze ROS vycházela od svého začátku v roce 2010 v distribucích téměř každý rok. K dnešnímu dni vyšlo 13 distribucí s poslední pojmenovanou ROS *Noetic Ninjemys*, která bude podporována do května roku 2025. Další distribuce již nejsou v plánu, neboť se předpokládá, že valná většina uživatelů přejde na novější ROS2, který oproti původnímu systému nabízí novinky jako:

- Širší podpora hostitelských systémů. ROS je oficiálně testován pouze na *Ubuntu* Linuxové distribuci, ale ROS2 distribuce se před vydáním testují na *Ubuntu*, *OS X* a *Windows*.
- Implementována podpora *C++* 11 funkcí (ROS podporuje *C++*03).
- Implementována podpora *Python* 3.5 (ROS podporuje *Python* 2).
- ROS2 implementuje pro komunikaci rozhraní na základě DDS standardu, díky kterému je možné využít služeb QoS (Quality of Service) a vylepšit komunikaci na zahlcených sítích
- Podpora více sestavovacích nástrojů krom *CMake*.
- Vylepšená práce s hardwarovými prostředky.

ROS2 opět vychází v distribucích jako jeho předchůdce. Tyto distribuce fungují podobně jako distribuce linuxových systémů a umožňují vývojářům pracovat vždy nad

relativně stabilním kódem a v dané distribuci již nepřichází žádné novinky, pouze případné opravy chyb.

V době psaní práce existuje 7 vydaných verzí ROS2 a poslední dvě aktuálně podporované jsou *Foxy Fitzroy* a *Galactic Geochelone*. V diplomové práci budeme pracovat s verzí *Foxy Fitzroy* kvůli největší podpoře a nabídce předpřipravených materiálů. Další plánovaný balíček by měl přijít v květnu roku 2022 pod jménem *Humble Hawksbill*. ROS distribuce je také vždy spojena s konkrétní distribucí *Ubuntu*. Proto při použití jiné než doporučované verze, může dojít k nepředpovídaným problémům, je proto dobré se držet vždy doporučených verzí. Pro distribuci *Foxy Fitzroy* a *Galactic Geochelone* je doporučena verze *Ubuntu* 20.04 *Focal Fossa*.

Pro zajímavost: distribuce jsou vždy pojmenované podle druhů želv. Želvy jsou pro ROS specifické z důvodu prvotně vytvořených tutoriálů, které popisují, jak začít fungovat s tímto frameworkem. V tutoriálech se pracuje právě s pohybem jednoduché želvy v prostředí.

Obrázek 10: ROS2 Foxy Fitzroy
Zdroj: [25]

3.1.2 Základní komponenty

ROS funguje na základě principu odesílání a přijímání zpráv. Princip této komunikace využívají jednotlivé komponenty. Ty jsou dále označovány v textu práce vždy anglickým názvem, neboť to jsou klíčová slova při používání příkazů v praktické části a jejich překlad by mohl být pro čtenáře matoucí.

Node – Základní stavební kámen frameworku ROS je takzvaný *node* neboli v českém změní uzel. Jednoduchý program, schopný většinou fungovat samostatně a obstarává konkrétní funkci robota. Příkladem *nodu* může být laserový senzor, ovládání kamery, pohon, ovládání servomotorů robotické ruky atd. *Node* je tedy zodpovědný za fungování definovaného celku robota. *Nodes* následně mohou mezi sebou komunikovat a předávat si informace o svém stavu nebo zpracovávat přijatá data a ty vyhodnocovat pro svoji funkci.

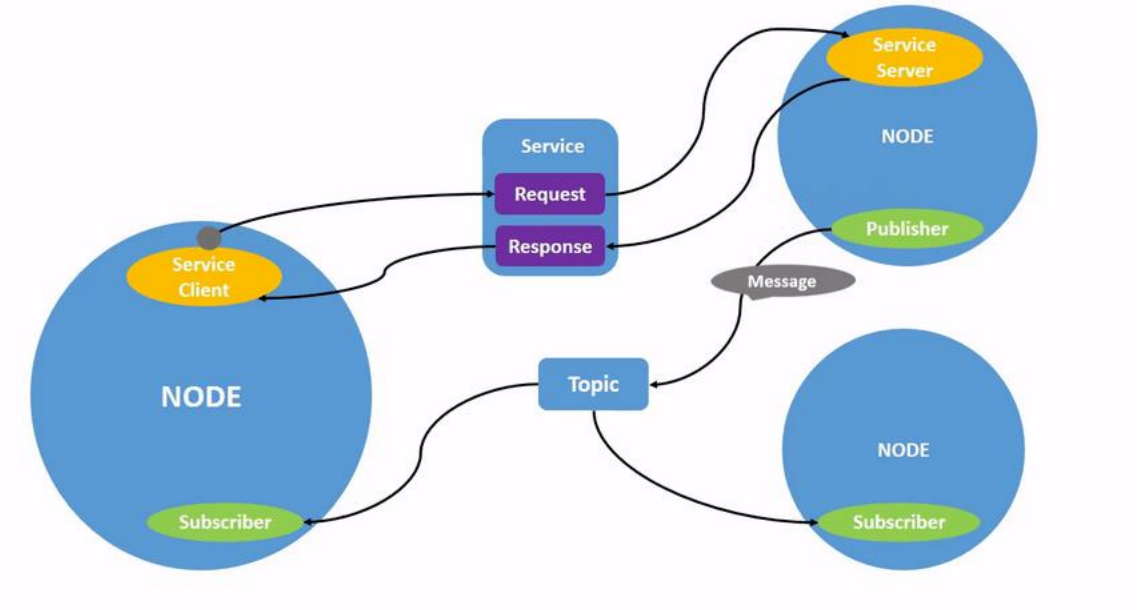
Topic – V českém překladu téma. *Topic* je způsob, jak mezi sebou *nodes* komunikují. Fungují na základě principu *publish/subscribe* (odeslat/přijmout) asynchronním protokolu pro zasílání zpráv. *Node* tedy může naslouchat *topic* nebo naopak mu posílat data. *Topic* může zprávy jak posílat, tak přijímat a může být vázaný na n *nodes*. Relace mezi *nodes* a *topics* je tedy N:M. Příkladem *topics* mohou být data z kamery (co aktuálně kamera vidí), čas, status robotické ruky (je nahoře nebo dole?), popis robota atd.

Messages – Už víme, že *node* může komunikovat s *topic*. Tedy přijímat či posílat data. Konkrétní způsob posílání dat obstarávají v ROSu *messages* (zprávy). ROS2 nabízí hned několik šablon pro *messages*, například *point* (bod v prostoru), který obsahuje tři souřadnice x, y, z datového typu *float64*. Uživatel si může vytvořit svoje šablony pro vlastní potřeby. Přestože ROS může fungovat v prostředí rozdílných systému a samotné *nody* můžou být naprogramované v různých programovacích jazycích (*Python* či *C++*), je nutné dodržet stejný formát neboli definici zpráv. Jinak komunikace nebude fungovat správně.

Parameters: *Nodes* obsahují v sobě zabudované parametry, které umožňují jednoduše měnit konfiguraci bez nutnosti zasahovat do samotného kódu. Parametr musí být vestavěný typ v ROSu a nejsou podporovány složité struktury vytvořené uživateli. Příkladem parametrů jsou: *refresh rate* kamery nebo jednotka rychlosti posunu.

Services: Služba v ROSu je nějaká operace, kterou většinou vyvolává uživatel. *services* si můžeme představit jako panel s tlačítky, kde každé jedno tlačítko zavolá příslušnou s*ervice*, která provede nějaké volání. Toto volání je synchronního charakteru. Typické příklady jsou *Start*, *Stop*, *Reset*. Volání těchto služeb často vyžaduje vložení nutných parametrů.

Actions: Podobně jako *services*, akce nám dávají možnost ovládat robota skrze volání. *Actions* jsou ale oproti *services* asynchronního charakteru. Pokud je zavoláme, nemusí se provést hned, ale zařadí se do jakési fronty příkazů. Akce také často ze svého charakteru vyžadují určitý čas na jejich vykonání. Příkladem akce může být *MoveForward*, kde vstupními argumenty budou dvě číselné hodnoty: jak daleko má robot jet a jak rychle se má pohybovat. [26]

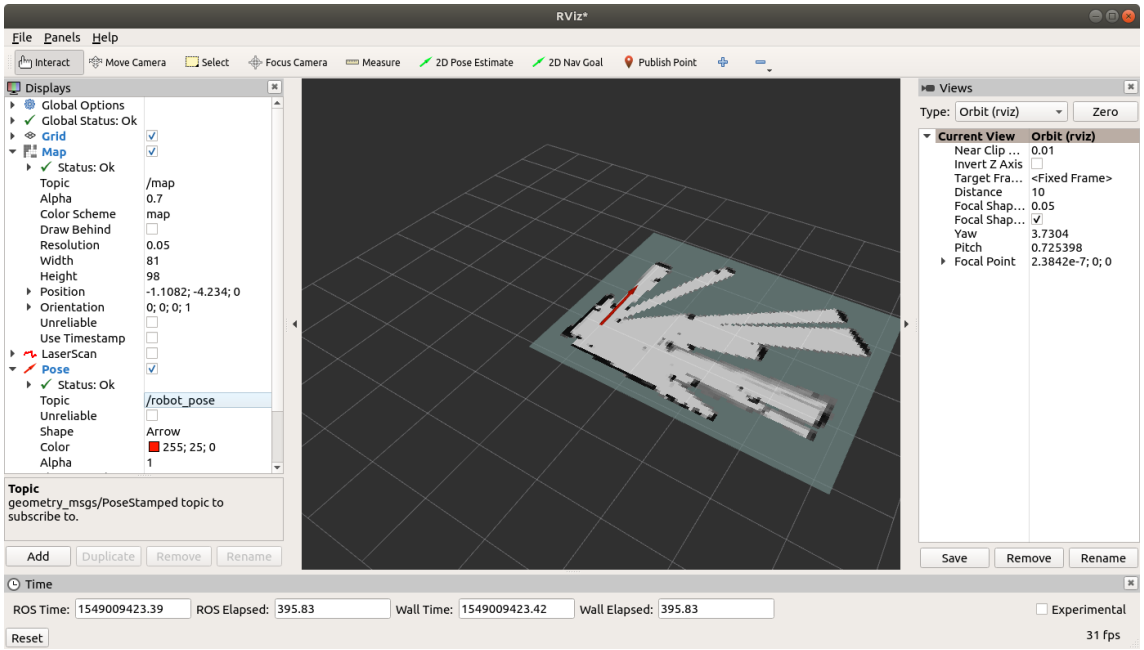


Obrázek 11: Schéma ROS node, topic a service
Zdroj: [25]

3.1.3 Nástroje

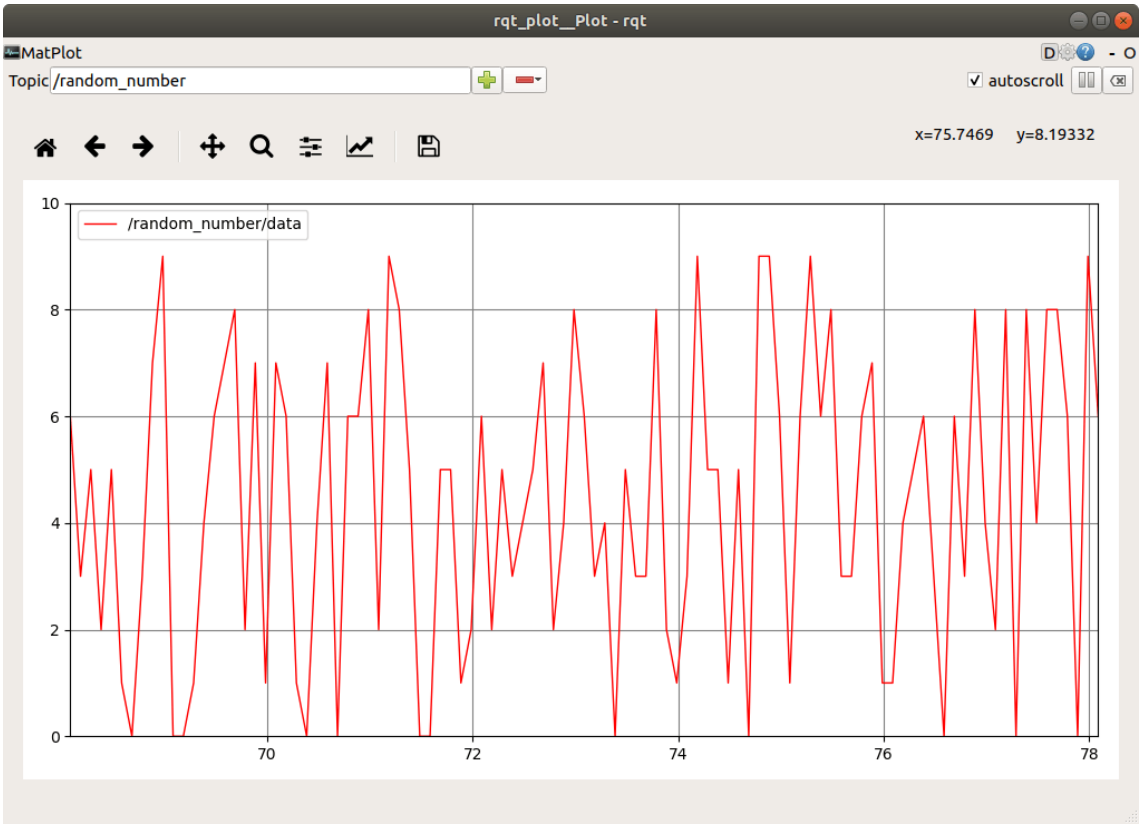
ROS pro svoji funkci a vývoj robotických zařízení využívá několik nástrojů, které ve velké míře usnadňují a urychlují práci. Tyto nástroje umožňují ladit, vizualizovat a vykreslovat stav našeho programu bez nutnosti testovat vývoj na skutečném stroji. Díky tomu ušetříme spoustu času nahráváním programů na konkrétního robota a vyvarujeme se také jeho případnému poškození při výskytu chyb a problémů v kódu. Základní trojice nástrojů jsou:

- **Rviz** – Zkratka je odvozena od ROS *Vizualization*. *Rviz* je nejzákladnějším a nejpoužívanějším nástrojem při práci s ROS frameworkem. Tento nástroj poskytuje 3D vizualizaci robota, kterého můžeme popsat pomocí URDF (*Unified Robotics Description Format*, XML formát popisující tvar robota pomocí částí a spojů). Nástroj umí vizualizovat základní typy zpráv jako laserové skeny, obrazy z kamer a výstupy ze senzorů. *Rviz* tak poskytuje celkový pohled na robota v prostředí a umožňuje základní práci s konkrétními součástmi robotického stroje. Díky tomu můžeme rychle identifikovat problémy nebo chyby bez nutnosti testování na skutečném robotovi.



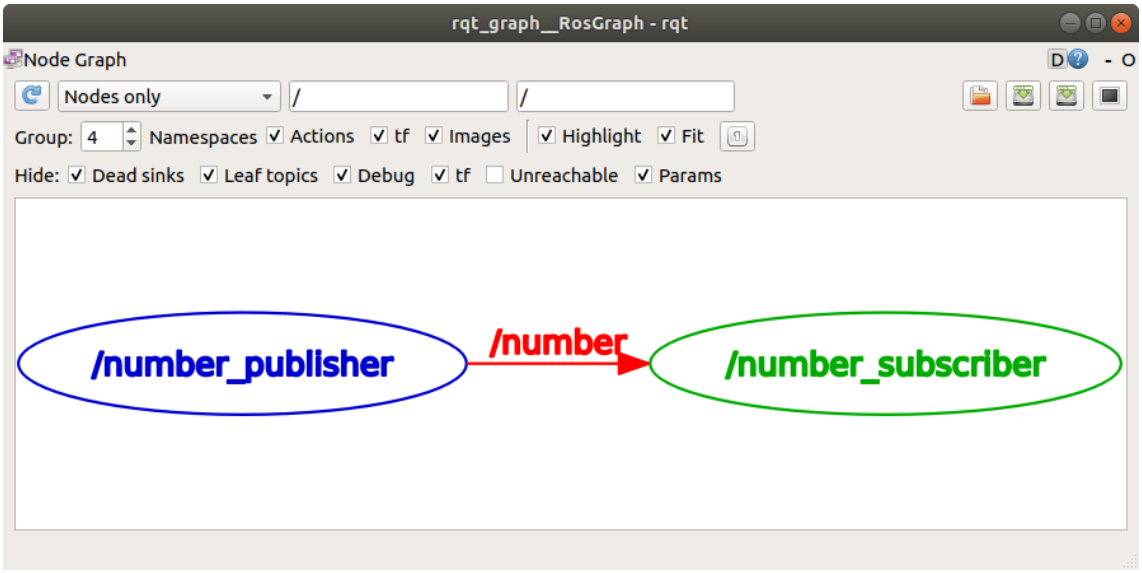
Obrázek 12: Rviz vizualizační nástroj pro ROS framework
Zdroj: [27]

- **RQt** – Založen na obecném nástroji *Qt* nabízí tento nástroj hned několik modulů pro vizualizaci, úpravu či vytváření jednotlivých rozhraní pomocí předpřipravených šablon a změn jejich konfigurace. ROS nabízí hned několik předpřipravených uživatelských rozhraní, některé vybrané z nich si zde popíšeme.
- **RQt_plot** – Jak již z anglického názvu vypovídá, jedná se o zobrazení nějakého grafu. Příkladem hodnot může být pozice, rychlost nebo akcelerace robota. Díky tomu můžeme sledovat výchylky a plynulost pohybu. Nástroj umožňuje vykreslování několika hodnot v jednom grafu. [28]



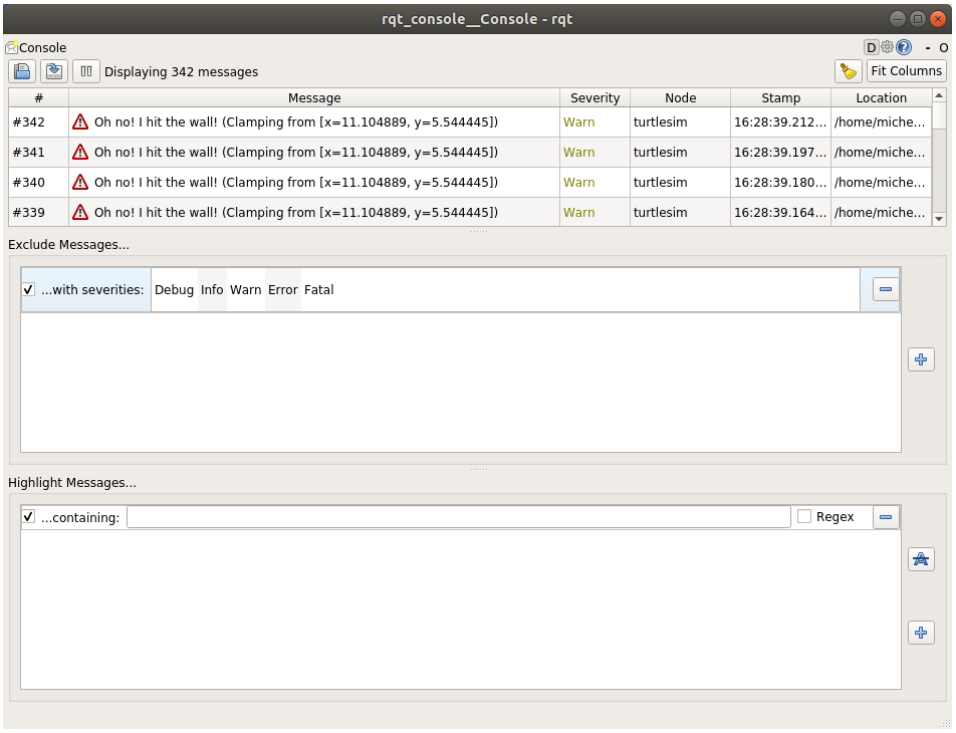
Obrázek 13: RQt plot
Zdroj: [28]

- **RQt_qraph** – Nástroj sloužící k zobrazení *nodů*, *topiců* a *messages*, které si posílají mezi sebou. Využití najde například při debugování základní komunikace, kdy dokáže přehledně zobrazit, jak jednotlivé části robota spolu komunikují. Na obrázku 14 je vidět jednoduchý příklad dvou *nodů*, kde jeden posílá zprávu druhému. [29]



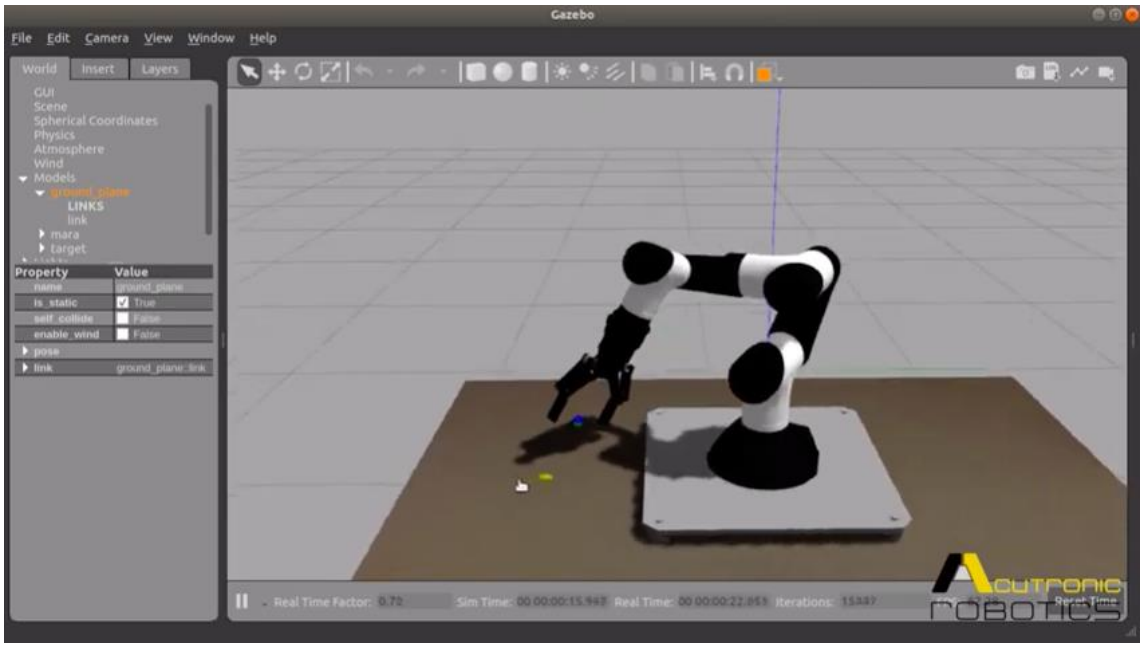
Obrázek 14: RQt graph
Zdroj: [29]

- **RQt_console** – Tato konzole umožňuje odchytávat a filtrovat zprávy, které vytváří jednotlivé části robota a posílá dále. Díky pokročilému filtrování můžeme sledovat například pouze důležité zprávy, které nám dávají informaci, že robot narazil do zdi. [30]



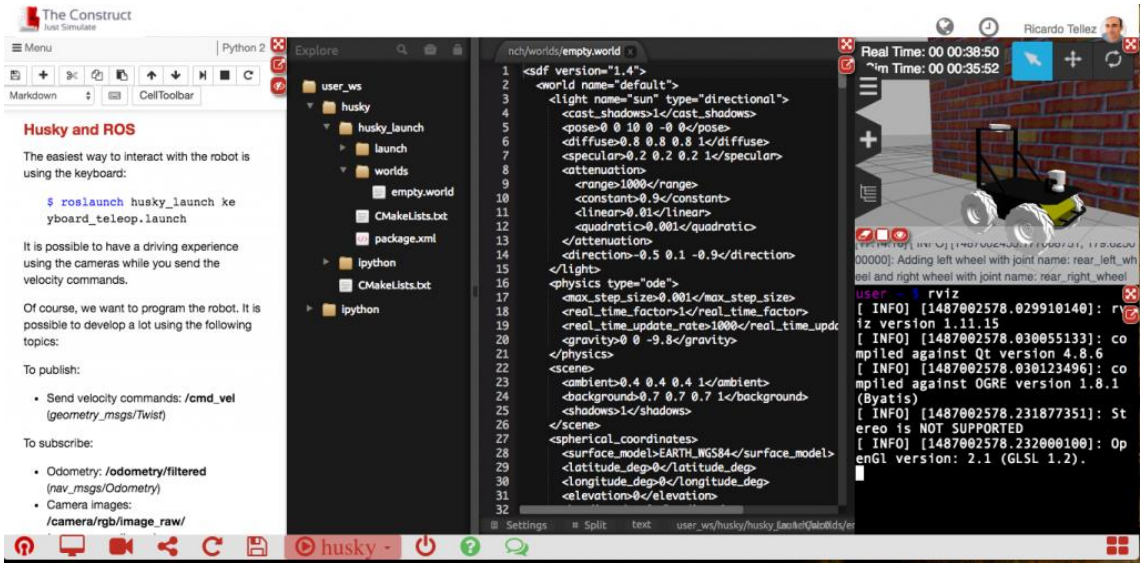
Obrázek 15: RQt konzole
Zdroj: [30]

- **Gazebo** – Simulátor třetí strany, který poskytuje silnou integraci do *frameworku* ROS. *Gazebo* je simulátor prostředí s realistickou fyzikou založenou na *enginu Ignition*. Využívá SDF (*Simulation Description Format*) algoritmus, který ale vyžaduje poměrně pokročilou znalost fyzikálních veličin pro správné nastavení konfigurace jednotlivých SDF souborů. *Gazebo* nabízí rozmanitou nabídku předpřipravených prostředí a modelů pro jednoduchou alteraci či vytvoření kompletně nového prostředí pro specifické testovací účely. [31]



Obrázek 16: Gazebo simulátor
Zdroj: [31]

- **ROS Development Studio** – Přestože ROS jako takový nabízí již předpřipravené nástroje pro vývoj robotických strojů, tak i přesto uživatel musí projít úvodní kolečko přípravy a instalace jednotlivých součástí frameworku. Proto společnost *The Construct* přišla s projektem vývojového studia, které již uživateli předpřipraví balíček pro vývoj a je možné skrze webový prohlížeč bez nutnosti instalovat balíčky na své fyzické zařízení vyvíjet robotický program. Základní kurzy a vyzkoušení Development Studia je zdarma, pokročilé funkce jsou ale zpoplatněny, neboť provoz serverů, na kterých software běží, není zdarma. Je to ale zajímavý přístup k vývoji a sdílení nápadů v oblasti robotiky. Společnost také nabízí programy pro školy či firmy. [32]

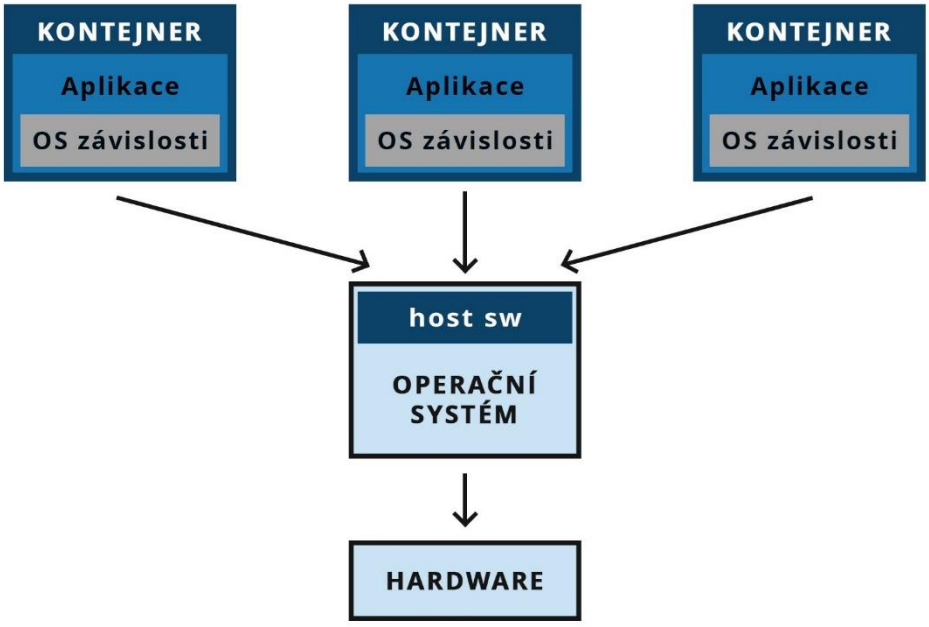


Obrázek 17: ROS development studio
Zdroj: [32]

3.1.4 ROS a Docker

Kontejnerová virtualizace jakožto jedna z možností, jak využít hardwarové prostředky pro vícero operačních systému, je ve světě robotiky už velmi dlouho. Její masové popularitě se ale dostalo až v pozdějších letech díky řešení zvané *Docker* se symbolickou ikonou velryby.

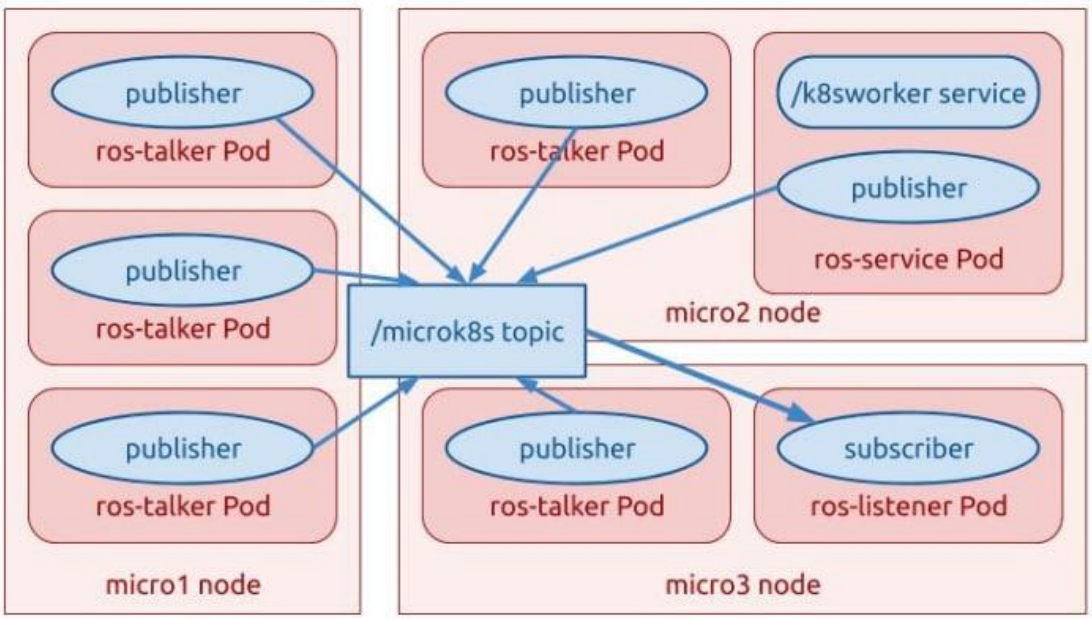
Docker je sbírka komplexních nástrojů nabízející velmi zjednodušenou práci s virtuálními kontejnery, ve kterých můžeme pouštět téměř vše od webových serverů až po databáze. *Docker* umožňuje velmi rychle nainstalovat a spustit předpřipravené nástroje v konkrétní verzi, kterou zrovna potřebujeme. Jakékoliv testování, obnovování a přenastavování systémů a služeb je otázka pár minut a pouze několika příkazů. ROS2 je tedy možné plně spouštět v kontejnerovém prostředí *Dockeru* a umožňuje to lidem, kteří se robotikou zabývají, pustit se plně do vývoje robotů bez potřeby relativně složité instalace a konfigurace prostředí a systému. [33]



Obrázek 18 Schéma kontejnerové virtualizace
Zdroj: [34]

3.1.5 ROS2 a Kubernetes

ROS ale nemusí nutně běžet pouze v kontejnerech jako je *Docker*. *Kubernetes* je systém pro takzvanou orchestraci kontejnerové virtualizace na úrovni samotného operačního systému. Pod pojmem orchestrace si představme službu, která zajišťuje automatickou konfiguraci, nasazení, správu a údržbu systémů (kontejnerů), které fungují v rámci *kubernetes*. Jednotlivé kontejnery se slučují do takzvaných *podů* a umožňuje tedy jednoduché logické uskupení. Pokud máme jednu aplikaci, která se skládá z vícero součástí (například samotná aplikace + databáze), tak všechny tyto části fungují v jednom *podu*. Jeden *kubernetes* pod může obsahovat jeden až nekonečno ROS *podů*, které spolu následně komunikují jako by byly v jedné fyzické síti i přesto, že realita může být taková, že jednotlivé *pody* jsou od sebe vzdáleny fyzicky desítky nebo stovky kilometrů. Na obrázku 19 můžeme vidět znázorněnou strukturu komunikace ROSu v *kubernetes* prostředí. Nastavení ROS *nodů* v *kubernetes* vyžaduje poměrně pokročilou znalost samotného fungování této technologie a je to kapitolou samo o sobě. V práci tedy nebude obsažena praktická ukázka fungování ROS *frameworku* v *kubernetes* prostředí. [35]



Obrázek 19 Kubernetes a ROS struktura
Zdroj: [35]

3.1.6 Výkon a využití hardwarových prostředků

Důležitým faktorem u robotických systémů je schopnost co nejefektivněji využít prostředky, které má software k dispozici jak z hlediska výkonu, tak z hlediska úspory, neboť spousta robotických zařízení je navržena na pohyb v prostředí a jejich zdroj fungování jsou tedy dost často baterie. Pokud se budeme bavit například o robotovi fungujícím v autonomním skladu, který dostává neustále nové a nové úkoly a nemá čas na dobití energie, musí se už s tímto faktorem počítat při návrhu.

3.2 Rojová robotika

Se stále snižující se cenou hardwarových prostředků a zvyšující se poptávkou po automatizovaných robotických jednotkách se také zvýšil zájem o princip rojové robotiky (z anglického *Swarm robotics*). Princip takového systému vychází z pozorování přírodních rojových uskupení včel, kde jedna včela je zanedbatelný článek celku. Tento princip tvoří v podání robotiky malé, jednoduché, levné a nahraditelné robotické zařízení, které můžeme použít pro efektivní a často automatizovanou práci. Ekosystém je využitelný například při prozkoumávání prostoru, převozu materiálu nebo k usnadnění sběru rostlin na farmách.

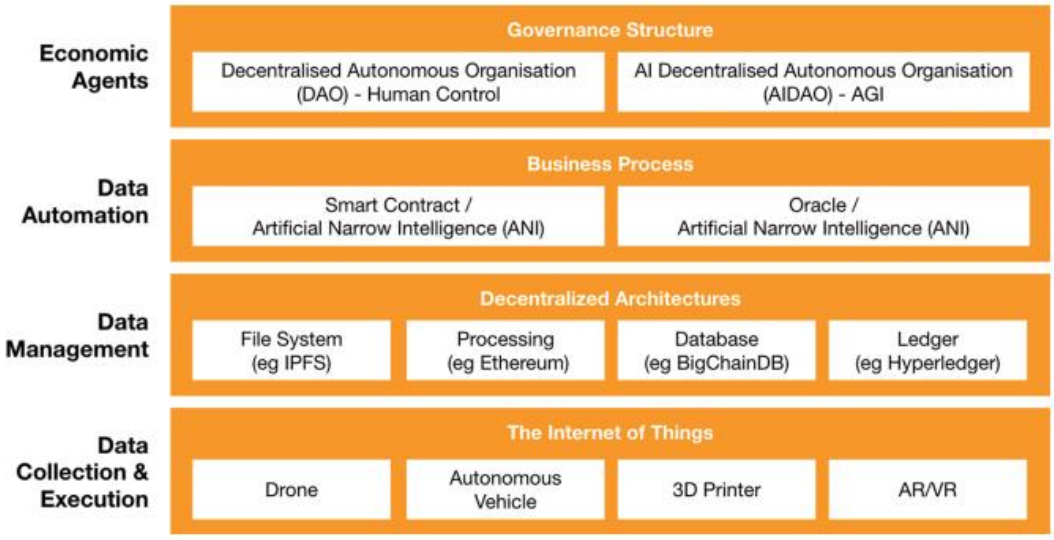
Specifikum roje tradičně bývá lokální udržování informací a jeden prvek nezná celkový pohled na svět, komunikuje vždy s ostatními nebližšími roboty. Tato vlastnost má své opodstatnění, přináší ale i spoustu nevýhod. Každý robot se nemusí vždy rozhodovat správně podle potřeb celku a proto je ve světě robotiky potřeba řešit i otázka bezpečnosti a možnosti různých útoků při nasazení v komerčním prostředí. [36]

3.2.1 ROS2swarm

ROS2 nabízí modul pro řízení skupiny robotů v roji. Tento modul si dává za úkol zjednodušení chování vícero robotů provozující stejnou činnost, jako je například pohyb v prostředí. Pomocí tohoto modulu je skupina robotů schopna přesunout se z místa A na místo B v určité formaci a předávat si informace ze svých senzorů a čidel pro efektivnější pohyb v prostředí, přesun přes překážky a předcházení případným kolizím. [37]

3.2.2 Blockchain

Alternativou k distribuovanému *frameworku* ROS2, který nám představuje klasické principy digitální komunikace, může být *blockchain*. Systém, který původně vznikl jako princip fungování kryptoměny *Bitcoin,* nachází využití i v jiných oblastech jako je například i distribuovaná robotika. *Blockchain* lze využít jako prostředek pro vytvoření jednotné databáze informací pro velké uskupení robotů v roji popsaném na začátku této podkapitoly. *Blockchain* nabízí oproti jiným způsobům jednotnou datovou strukturu, kde každý robot má stejný pohled na svět jako všichni ostatní. Základní princip komunikace je, jak již z názvu vypovídá, publikování informací v takzvaných blocích, které se na sebe navzájem řetězí. Kontrola a přijetí těchto bloků je řízena ostatními zařízeními v systému. Není zde tedy žádná centrální autorita, která hlídá ostatní zařízení v systému. Výhodou je plná distribuce, odolnost proti útokům, škálovatelnost systému nebo téměř autonomní chování celku. Komunikace ale může být z hlediska množství dat a požadavku vysokého výkonu poměrné náročná, což je problém nejen rojové robotiky, ale i robotiky obecně. Při požadavku malého, levného a nenáročného zařízení totiž musíme poskytnout dostatečný hardwarový výkon, který zajistí komunikaci v reálném čase. [36] [38]

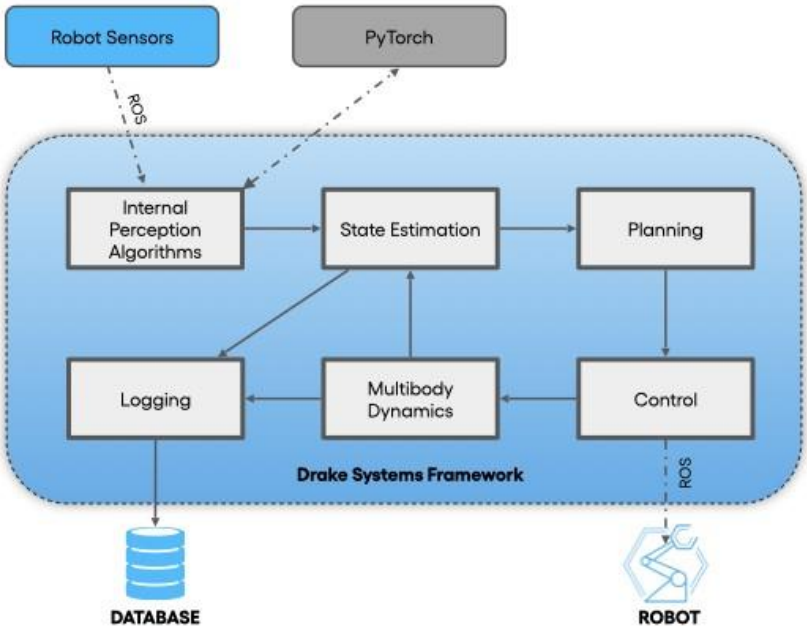


Obrázek 20 Oblasti využití Blockchainu
Zdroj: [38]

3.3 Alternativy k ROSu

Jak již bylo zmíněno v úvodu této kapitoly, ROS je pouze jeden z mnoha prostředníků poskytující základní kámen neboli vrstvu, která nám umožňuje rychlejší a jednodušší vývoj robotických zařízení. Vybrané alternativy jsou popsané níže.

Drake je *toolbox*, který vznikl jako projekt týmu *Robot Locomotion Group* z centra MIT a momentálně jeho hlavní vývoj je veden týmem z *Toyota Research Institute*. Je to tedy sada nástrojů pro návrh, analýzu a řízení robotických zařízení. Nástroje jsou zaměřeny na vývoj komplexních robotických strojů. Je založen na jazyku *Python* a *C++*. Jakožto open-source projekt tak nabízí již předpřipravené rozhraní a algoritmy pro využití ve vlastní implementaci. *Drake* je zaměřen primárně na složité robotické fungování, optimalizaci pohybu a je pro to od základu vybaven. V následujícím schématu na obrázku 21 vidíme dokonce využití *frameworku* ROS. Dobře navržený robotických systém je schopný jednoduše komunikovat s ostatními prvky v ekosystému a nemusí to nutně být další robotický *framework*. [39]



Obrázek 21: Schéma interního fungování frameworku Drake
Zdroj: [39]

Moos je podobně jako ROS komunikační framework pro robotická uskupení, který nabízí uživatelům jednoduchou platformu, na které můžou stavět vlastní robotická zařízení. Skládá se z takzvaného *CoreMOS* jakožto robustní komunikační platformy, kterou využívají další *Moos* nástroje a aplikace. Tyto aplikace fungují na podobný princip jako moduly ROS frameworku. Aplikace si můžeme naprogramovat sami v jazyce C++ či využít již vytvořené. Dokumentace a nabídka aplikací jsou oproti ROSu slabší. [40]

Microsoft Robotics Developer Studio je nástroj založený na .NET pro vývoj robotických zařízení. Tento nástroj se více zaměřuje na chování robustního specifického robota než na komunikaci v distribuovaném systému jako ostatní zmíněné alternativy. Studio poskytuje uživatelům pokročilé nástroje pro jednoduché vytváření chování robota pomocí schémat a ladění pomocí GUI s velmi dobrou simulací. Vývoj nástroje jako takového již ale nepokračuje a není doporučeno s ním začínat nové projekty z důvodu možných problému s kompatibilitou nového hardwaru. [41]

Další alternativy robotických systémů jsou zmíněny pouze výčtem: Carmen, GOBOT, DART (Dynamic Animation and Robotics Toolkit).

4 Porovnání metod komunikace

V následující kapitole porovnáme způsob komunikace „klasického“ robotického frameworku a poměrně experimentálního způsobu komunikace pomocí technologie *blockchain*. *Blockchain* byl vybrán pro porovnání právě pro svoji odlišnou formu komunikace. ROS2 a ostatní podobné frameworky fungují ve své podstatě na stejných principech a odlišují se pouze v malých odchylkách a detailech.

4.1 Porovnání zpracování dat

ROS2 – Data jsou vždy uložena lokálně na jednotlivých zařízeních a robot propaguje do sítě pouze to, co je potřeba. Ostatní robotické zařízení pak sbírají data od ostatních podle vlastní potřeby.

Blockchain – Všechna data, která robot propaguje či potřebuje, jsou uložená v *blockchainu*. Z principu fungování *blockchainu* jsou tedy informace vždy jednotné. Zařízení tedy nemají vlastní zdroj dat, ale musí si držet vždy nějakou aktuální část *blockchainu*. Není totiž reálné, aby všechna zařízení držela celý blockchain, neboť ten neustále roste s každou novou informací v něm.

4.2 Porovnání zabezpečení

ROS2 – Komunikace v základním režimu není nijak zabezpečena a nedochází k žádnému šifrování. Předpokladem je, že síť ve které se robotická zařízení nachází, je dostatečně zabezpečena. Pokud tento předpoklad není dostatečný, ROS nabízí šifrování komunikace pomocí balíčku SROS2. ROS2 funguje na základě modulů neboli balíčků a jeho fungování je tedy jednoduše rozšiřitelné. Balíček SROS2 nabízí šifrování komunikace mezi robotickými zařízeními na základě kryptografických klíčů. Tato funkcionalita není nijak složitá, ale pro uživatele přidává navíc další vrstvu konfigurace a potenciálních komplikací. Zpracování šifrované komunikace je také náročnější na výpočetní výkon, který by mohl být využit jinde a ve výsledku tím snižujeme celkovou výdrž baterie, což může být velký problém u menších robotů. [42]

Blockchain – Princip fungování blockchainu je založen právě na bezpečné komunikaci. Každá informace, která je umístěna do blockchainu musí být schválena ostatními zařízeními. Pokud by tedy útočník chtěl umístit falešnou informaci do

systému, musel by současně napadnou všechna zařízení v síti. Čím více zařízení v ekosystému existuje, tím je útok složitější a téměř neproveditelný.

4.3 Shrnutí porovnání

Tabulka č. 1 ukazuje přehledné srovnání principů fungování frameworku ROS2 a *Blockchainu*. Z tabulky a informací v předchozích kapitolách můžeme říct, že největší výhodou Blockchainu je jeho škálovatelnost, možnost silného autonomního chování a hlavně implicitní bezpečnost. Komunikace v blockchainu může být ale náročná a vyžaduje rychlou a robustní síť. Naopak ROS2 nabízí tradičnější způsob komunikace posíláním zpráv, které druhá strana musí zpracovat v čas přijetí. ROS2 bych tedy doporučil pro komunikaci menšího počtu zařízení v definovaném prostředí. Pro velké množství autonomních robotu je určitě dobré zvážit využití komunikace na principu blockchainu.

	ROS2	**Blockchain**
Způsob komunikace	Zprávy skrze služby nebo témata.	Zasíláním bloků, které se řetězí.
Výpočetní náročnost	Na malé robotické zařízení náročná, ale nenarůstá.	Zvyšuje se s počtem robotů.
Bezpečnost	Předpokládá se použití na bezpečné síti. Možnost využít modul SRO2.	Bezpečné ze základního principu fungování.
Rychlost	Podle rychlosti sítě	Zpracování bloků v síti může být zdlouhavé v řádu až několika minut.
Složitost návrhu	Čím více různorodých robotů, tím složitější návrh.	
Škálovatelnost	Nutná implementace na úrovni každého robota.	Jednoduchá
Autonomní chování	Potřeba zásahu třetí strany, složitější implementace komunikace.	Princip komunikace může být plně autonomní.

Využití v roji	Pomocí modulu ROS2swarm	Ideální pro využití v rojové robotice.

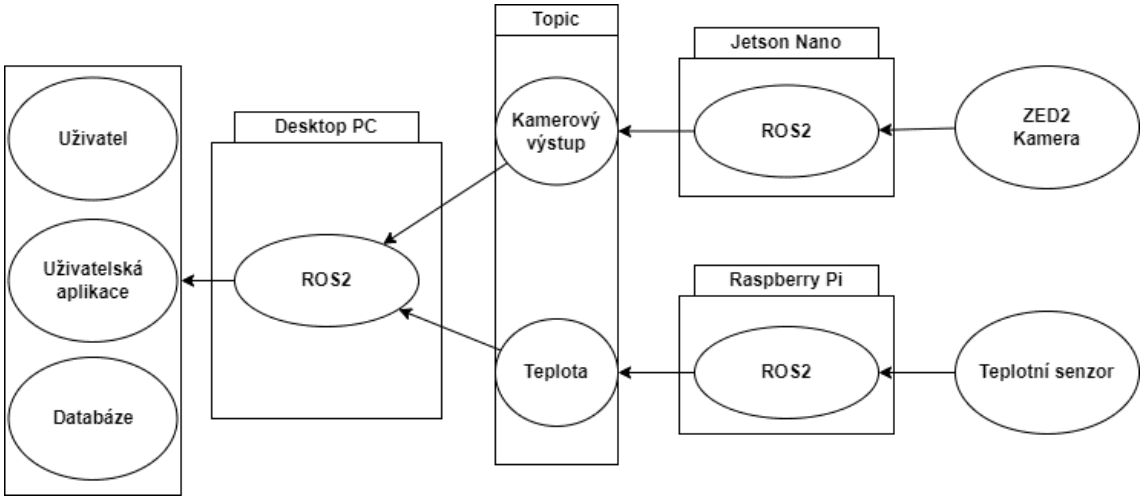
Tabulka 1: Porovnání komunikace

5 Praktická ukázka

V praktické části si ukážeme příklad fungování robotického frameworku ROS2 na jednoduchém příkladu, který obsahuje tři různá zařízení, ke kterým jsou připojeny periferie jako kamera a teplotní senzor.

5.1 Návrh

Ze schématu na obrázku 22 vidíme, že se náš robotický systém skládá ze tří větších celků. Robotický systém bude zaznamenávat vizuální data a teplotu prostředí. Tato data mohou být následně uložena do datového úložiště, ze kterého se můžou dále vyhodnocovat statistické údaje, nebo zobrazovat přímo uživateli. Prvním zařízením je klasický desktop PC, který zastává hlavní roli výpočetního systému. Sbírá, ukládá a vyhodnocuje data, která následně poskytuje uživatelům či aplikacím které s nimi pracují. Druhé zařízení je *Nvidia Jetson,* ke kterému je připojena kamera *ZED2* zpracovávající obraz a detekci osob v prostoru. Třetím zařízením je *Raspberry Pi*, ke kterému je připojen jednoduchý senzory teploty.

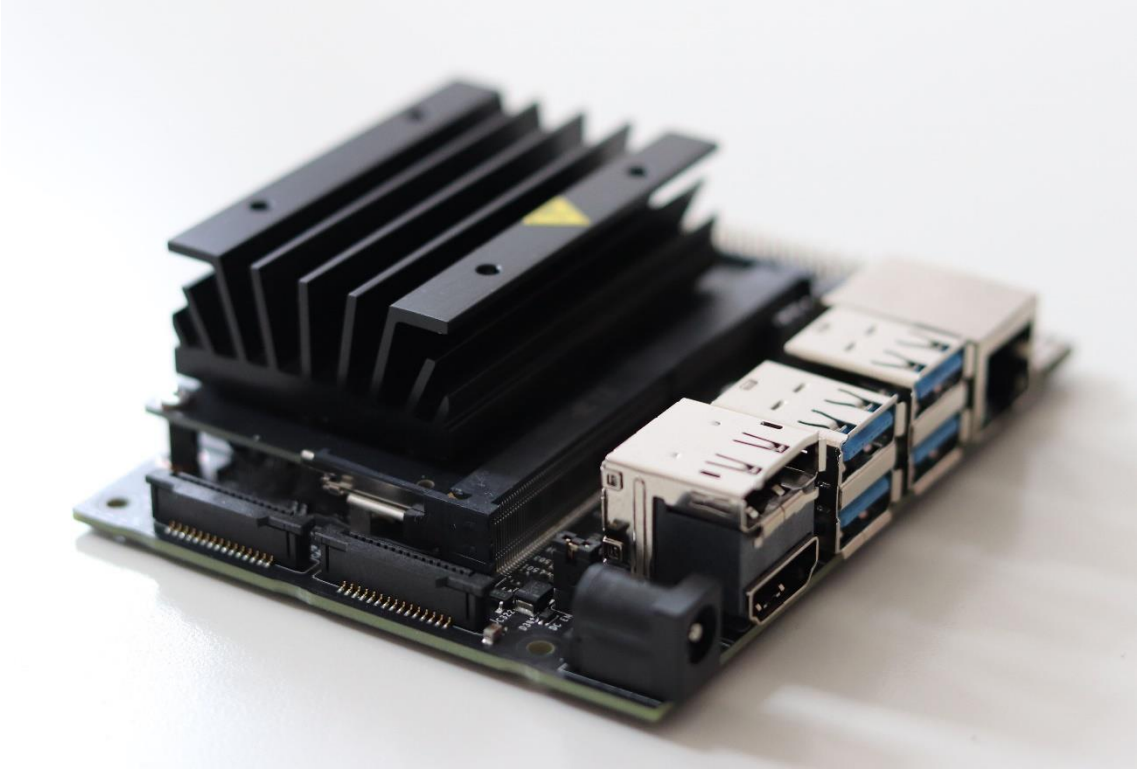


Obrázek 22: Schéma vlastního návrhu

5.2 Nvidia Jetson Nano

Jetson je zařízení od výrobce grafických čipů *Nvidia*, které na svoji velikost nabízí výkonný hardware v malé podobě. Je vhodný pro vývoj umělé inteligence a robotických zařízení pro aplikace jako rozpoznávání obrazu, detekce objektů, segmentace nebo zpracování hlasu. Po technické stránce je *Jetson* malá destička osazena čipem architektury ARM, 128jádrovým grafickým Maxwell čipem, 4GB operační pamětí, slotem na SD kartu, několika USB porty, Ethernetový portem, HDMI a DisplayPort pro výstup videa, dva MPI kamerové vstupy a pasivní chlazení. Grafický čip podporuje *encodování* a *dekódování* až 4K videa. Pokud člověk plánuje nějakým způsobem pracovat s obrazem na svém robotickém zařízení, *Jetson* je určitě favorit číslo 1. Díky open-source projektům a silné komunitě uživatelů je práce s *Jetsonem* velmi uživatelsky přívětivá. [43]

Nvidia nabízí k *Jetsonu* svůj vlastní operační systém s názvem *Jetpack*, který obsahuje různé ukázky a tutoriály pro začátečníky. V naší robotické soustavě ale použijeme systém *Ubuntu* kvůli podpoře ROS2.



Obrázek 23: Nvidia Jetson Nano

5.2.1 ZED2 kamera

ZED2 je stereo kamera (má tedy dvě „oči“, stejně jako člověk), která je schopna zpracovávat 3D prostor s využitím umělých neuronových sítí a umělé inteligence. Svým fungováním a zpracováním je tedy velmi blízko imitaci lidských očí, neboť rozteč mezi kamerami odpovídá zhruba roztečí očí lidského těla. *ZED* ale nabízí ještě více, v kameře jsou již zabudovány pokročilé funkce jako rozpoznávání objektů a ploch, měření rychlosti pohybujících se věcí, sledování vlastní pozice, vnímání hloubky obrazu, rozpoznávání lidského těla a určení pozic lidského těla. Dále je kamera vybavena senzory teploty, tlaku, náklonu a dokáže mapovat prostředí při pohybu (SLAM). [44]

Obrázek 24: ZED2 stereo kamera

5.3 Raspberry Pi

Raspberry Pi je malé zařízení o velikosti kreditní karty, které je schopné vykonávat stejné úkony jako klasický stolní počítač či laptop. Zvládá vše od surfování na internetu, úpravu dokumentů, přehrávání videí či výuku programování například v jazyce *Python*. *Raspberry Pi* je cenově velmi dostupné, jednotlivé modely se (v závislosti na použitém čipu a velikosti paměti) pohybují s cenovkou v řádu 1000–4000 Kč. K tomuto zařízení můžeme připojit standardní příslušenství jako ke klasickému stolnímu počítači, od klávesnice až po monitor.

Raspberry Pi získalo velkou popularitu jako řídící modul pro různé projekty jako jsou meteostanice, hudební sestavy, chytré kamerové pasti, řídící síťové prvky nebo

také robotická zařízení, kvůli kterým je *Raspberry Pi* využito i v této práci. Jednodušší a levnější alternativa pro *Raspberry Pi* může být *Arduiono*, které ale svým výkonem nedosahuje požadovaných hardwarových zdrojů, které jsou potřeba například pro fungování ROS2 *frameworku*. [45]

Obrázek 25: Raspberry Pi 4 Model B

5.4 ROS2

Přestože oficiální dokumentace ROSu uvádí mezi podporované systémy Linux, MacOS nebo Windows. Nejlepší hostitelský systém je zcela určitě Ubuntu. Instalace ROSu na Ubuntu je nejjednodušší a má také největší základnu uživatelů. Přestože ROS funguje i na systému Windows, většina komunity se shodne, že to není ideální řešení a v pokročilejších věcech či úpravách můžeme narazit na určité limity či nedořešené problémy s kompatibilitou. Další nástroj jako například *Gazebo* také nemusí uspokojivě fungovat na jiném systému než Ubuntu (potažmo jiné linuxové distribuci).

Provozování ROSu je možné i v kontejnerech, konkrétně se můžeme bavit třeba o *Dockeru*. Na linuxové distribuci se jedná o jednoduchý proces, skládající se z několika málo příkazů. Pro simulaci v programu *Gazebo* ale není doporučeno spouštět hostitelský systém ve virtuálním prostředí, neboť dochází k problémům a následně zaseknutí celého systému. Z vlastní zkušenosti můžu tuto skutečnost potvrdit, původní

instalaci jsem provedl ve virtuálním prostředí *VMware Workstation Player* a práce se simulátorem *Gazeboo* je velmi nestabilní.

5.5 Základní příkazy

Před začátkem práce s *frameworkem* ROS2 je dobré si nastudovat sadu základních příkazů pro fungování. Většinu těchto příkazů lze spustit s dalšími volitelnými parametry.

ros2 – Všechny ROS2 příkazy začínají tímto. Říkáme tím našemu terminálu, který program chceme použít.

ros2 pkg – Zobrazení informací o balíčcích ve *workspacu*, ve kterém se právě nacházíme.

ros2 pkg list – Zobrazení informací o balíčcích v podobě kratšího seznamu.

ros2 pkg executables <název_balíčku> – Zobrazení spustitelných souborů v balíčku.

ros2 node – Zobrazení informací o spuštěných nodech.

ros2 node list – Zobrazení seznamu spuštěných nodů.

ros2 node info /<název_nodu> – Zobrazení *topiců* a služeb, vybraného *nodu.*

ros2 topic list – Zobrazení seznamu všech aktuálně běžících *topiců*.

ros2 topic info /<název_topicu> – Zobrazení informací o vybraném *topicu*.

ros2 topic echo /<název_topicu> – Vypsání výstupu *topicu* do konzole.

ros2 service list – Zobrazení seznamu všech aktuálně běžících služeb.

ros2 service show <název_služby> – Zobrazení definice konkrétní služby.

ros2 service call /<název_služby> '{parametry}' – Manuální zavolání služby s potřebnými parametry (zjistíme příkazem show).

ros2 launch <název_balíčku> <název_souboru> – Spuštění ROS2 souboru, například *nodu*.

colcon build – Spuštění buildu pomocí nástroj *colcon*.

rviz2 – Spuštění vizualizačního nástroje.

5.6 Obecný postup instalace ROS2 na Ubuntu distribuci

Jelikož je ROS2 distribuce vždy určitým způsobem vázána na distribuci operačního systému *Ubuntu*, je nutné dodržet a používat správné verze. V práci, jak již bylo zmíněno, je použita distribuce *Ubuntu Focal Fossa* (20.04) a ROS2 *Foxy Fitzroy* pro všechny tři zařízení.

Nejjednodušší postup instalace je pomocí linuxových balíčků, které obsahují rovnou i nástroje a ukázky pro ověření fungování. Postup instalace je převzat z oficiální dokumentace ROS2. Zde můžeme najít i další alternativy, jak *framework* nainstalovat a následně zprovoznit. [46]

Postup instalace:

V linuxovém terminálu spustíme postupně tyto příkazy.

Přidání ROS2 repositářů:

```
~$ sudo apt update && sudo apt install curl gnupg2 lsb-release
~$ sudo curl -sSL https://raw.githubusercontent.com/ros/rosdistro/master/ros.key  -o /usr/share/keyrings/ros-archive-keyring.gpg
```

A následně přidání repositářů do seznamu zdrojů:

```
~$ echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/ros-archive-keyring.gpg] http://packages.ros.org/ros2/ubuntu $(source /etc/os-release && echo $UBUNTU_CODENAME) main" | sudo tee /etc/apt/sources.list.d/ros2.list > /dev/null
```

Instalace ROS2 balíčku:

```
~$ sudo apt update
~$ sudo apt install ros-foxy-desktop
~$ sudo apt install ros-foxy-ros-base
```

Před spuštění jakéhokoliv ROS2 programu je nutné v terminálu definovat zdroj:

```
~$ source /opt/ros/foxy/setup.bash
```

Pro ověření funkcionality můžeme spustit jednoduchý demo program:

```
~$ ros2 run demo_nodes_cpp talker
```

A v druhém terminálu nebo na druhé zařízení:

```
~$ ros2 run demo_nodes_py listener
```

Tento jednoduchý demo program publikuje do *topicu* text:

```
Publishing ‘Hello world: X’ kde X je iterováno od 1 do nekonečna
```

A *listener* tento *topic* odebírá a vypisuje hodnoty do terminálu. Výstup tedy vypadá takto:

„I heard: [Hello World: X]“ Kde X je opět variabilní hodnota, kterou inkrementuje Publisher.

5.7 Struktura vlastního projektu

ROS2 funguje podobně jako jiný programátorský projekt na základě souborů, které fungují v určité struktuře. Základem takové struktury je takzvaný *workspace*, v překladu pracovní místo pro naše výtvory. Předpokladem pro vytváření vlastního *workspace* je již nainstalovaný balíček ROS2, tento postup jsme si blíže popsali v přechozí kapitole. Tento předpoklad je nutný pro spuštění ROS2 příkazů a je nutné opět v konzoli provést příkaz:

```
~$ source /opt/ros/foxy/setup.bash
```

Začneme tedy vytvořením složky, která bude sloužit jako náš *workspace* a v ní budou uloženy všechny potřebné složky a soubory. Použijeme příkaz:

```
~$ mkdir -p ~/dev_ws/src
```

Přepneme se do vytvořené složky:

```
~$ cd ~/dev_ws/src
```

V dalším kroku si vytvoříme takzvaný *package* neboli balíček, obsahující zdrojové soubory našeho projektu. Takových balíčků můžeme mít v jednom workspace kolik chceme. Při změně zdrojových kódu a následném *buildování* (vytváření spustitelných souborů ze zdrojových kódů) je pak ale nutné specifikovat, které balíčky chceme znovu sestavit.

Package vytvoříme příkazem:

```
~$ ros2 pkg create --build-type ament_cmake --node-name  <název_nodu>
 <název_balíčku>
```

Nebo pokud zdrojový kód bude v P*ythonu*:

```
~$ ros2 pkg create --build-type ament_python --node-name  <název_nodu>
<název_balíčku>
```

ROS2 nám následně vytvoří potřebné soubory, které následně můžeme upravovat. Vytvořené soubory jsou jmenovitě:

setup.py – *Python* soubor obsahující instrukce pro instalaci balíčku.

setup.cfg – Potřebný konfigurační soubor, pokud náš balíček používá jiné spustitelné soubory.

package.xml – Soubor obsahující metadata.

my_node.py – Zdrojový soubor obsahující logiku našeho *nodu*.

Po vytvoření souborů se přepneme zpět do složky našeho workspace a provedeme *build* balíčku. K tomu využijeme nástroj *colcon*. Syntaxe příkazu je velmi jednoduchá. Pokud máme více balíčku, můžeme parametrem specifikovat pouze ty, které chceme *vybuildovat*:

```
~$ colcon build --packages-select <název_balíčku>
```

Nyní můžeme spustit nově vytvořený *node*, který je prozatím prázdný a nic nedělá. Předtím je ale nutné opět terminálu specifikovat zdroj. Podobně jako jsme specifikovali zdroj ROS2, tak nyní musíme specifikovat zdroj našeho vlastního balíčku:

```
~$ . install/setup.bash
```

Spuštění nodu:

```
~$ ros2 run <název_balíčku> <název_nodu>
```

Výstup do konzole bude vypadat následovně, neboť ROS při vytváření prázdného nodu přidal jednoduchý kód, který vypíše název balíčku do konzole.

```
Hi from „název_balíčku“.
```

Nyní máme vytvořený ROS2 workspace s jedním, již spustitelným, balíčkem. Pro shrnutí si uvedeme, jak vypadá struktura takového workspace:

/dev_ws – workspace
- /src – složka se zdroji
  - /package1 – balíček
    - /package.xml
    - /setup.py
    - /setup.cfg
    - /my_node.py
  - /package2
- /build – složka se sestaveným kódem
- /install – složka obsahující primárně *setup.bash* pro spuštění našeho nodu

5.8 Instalace a konfigurace ROS2 – Nvidia Jetson

Pro jednoduší práci s *Jetsonem* jsou dostupné předpřipravené systémy, ať už od přímo od výrobce (*Nvidia Jetpack*) nebo různě upravené linuxové distribuce. Z důvodu potřeby dodatečné instalace ROS2 a *ZED* balíčků je pro nás jednodušší volba *Ubuntu* distribuce připravená pro *Jetson* spolu s podporou *ZED* kamery. V praktické ukázce je použita linuxová distribuce *Ubuntu* 20.04 (*Focal Fossa*) předpřipravena pro *Jetson Nano*. Poměrně dlouhá konfigurace systému pro správné fungování je popsána

v odkazu: [47]. K nahlédnutí je zde dostupný i předinstalovaný image, který byl pro jednodušší práci využit. Po nainstalování systému na SD kartu je možné *Jetson* zapojit a spustit. Systém se po chvíli načte a je možné na něj instalovat další potřebné balíčky, konkrétně *ZED* SDK potřebné k fungování *ZED2* kamery a ROS2 balíček. Kompletní návod na zprovoznění *ZED2* kamery spolu s frameworkem ROS2 poskytuje přímo výrobce kamer *StereoLabs* na tomto odkazu: [48]. ZED SDK spolu s ROSem funguje na takzvaném *wrapperu* neboli obalu, který poskytuje jakousi abstraktní vrstvu pro různé typy kamer a poskytuje informace a data do několika různých *topiců*.

Příklady topiců:

~/stereo/image_rect_color – obraz z levé a pravé kamery

~/right/camera_info – kalibrační data z pravé kamery

~/right_raw/image_rect_gray – černobílý obraz z pravé kamery

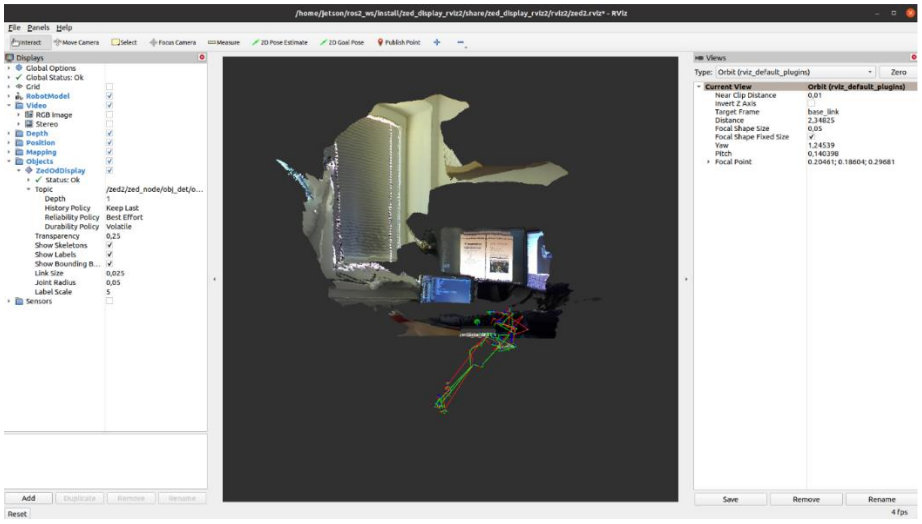
~/temperature/imu – teplota kamery (IMU senzor)

~/imu/data – data z akcelerometru, gyroskopu a orientace

K *topicům* pravé kamery existuje protiklad levé. Místo klíčového slova *right* vložíme *left*. Po nainstalovaní všech potřebných balíčků je možné spustit *wrapper* spolu s vizuálním zobrazením výstupu kamery pomocí RViz.

```
~$ ros2 launch zed_display_rviz2 display_zed2.launch.py
```

Výstupem na obrazovce je:



Obrázek 26: ZED2 Rviz obrazovka

Shrnutí instalace v bodech:

- Nahrání předpřipraveného operačního systému *Ubuntu* na SD kartu.
- Instalace ROS2 na systém.
- Instalace *ZED* SDK.
- Spuštění ROS2 *nodu*, který publikuje data z *ZED* kamery do výše zmíněných *topiců*, které bude náš desktop PC odebírat.

Hardwarová specifikace Nvidia Jetson Nano: [43]

- Procesor: Čtyř-jádrový ARM A57, 1.43 GHz
- RAM: 4 GB LPDDR4
- 4x USB 3.0
- HDMI a DisplayPort
- 2x konektor pro CSI/DSI kameru
- Micro USB pro napájení (výkon je limitovaný – 5 W)
- GPIO napájecí konektor (umožňuje využít maximální výkon – 10 W)
- Slot na micro SD kartu
- M2 slot
- 260 pinové pole

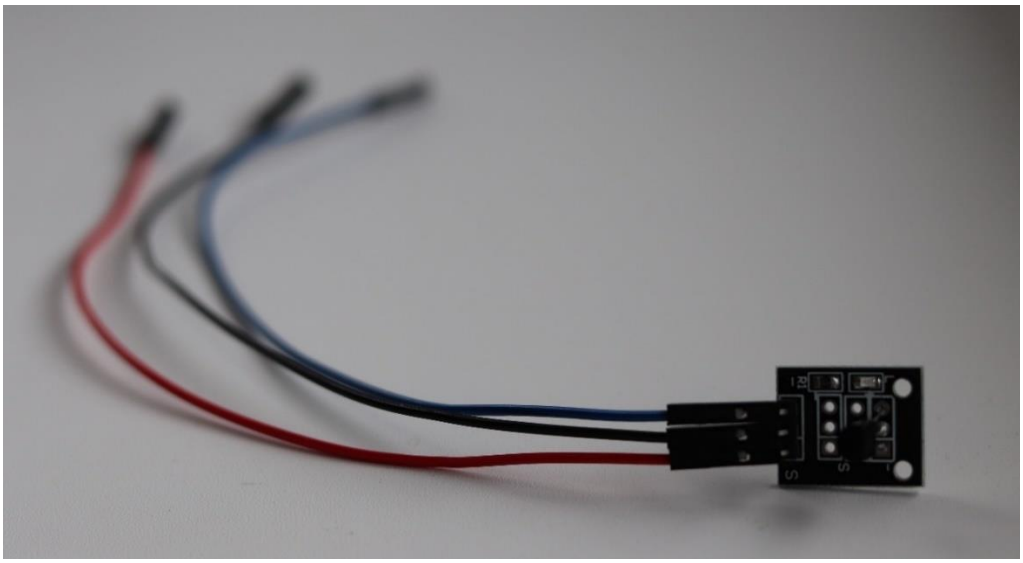
5.9 Instalace a konfigurace ROS2 – Raspberry Pi

Zprovoznění ROSu na *Raspberry Pi* je velmi podobné jako v přechozí kapitole 4.6 s *Jetsonem* od *Nvidie*. Rozdíl v těchto dvou zařízeních je však jejich výkon. *Raspberry Pi* nabízí tabulkově podobně výkonný hardware, avšak *Jetson* má velkou výhodu v grafickém čipu, který mu umožňuje mnohonásobně rychleji zpracovávat grafické informace. Pro šetření prostředků je tedy použit operační systém *Ubuntu* Server ve verzi 20.04 (*Focal Fossa*). Serverová verze *Ubuntu* nenabízí grafické rozhraní a pracuje se pouze s příkazovou řádkou, která ale pro naši práci naprosto dostačuje a již tak omezené prostředky nejsou využity na zobrazení grafického prostředí.

Pro usnadnění práce byl opět použit *image Ubuntu* serveru s předpřipraveným natavení připojení přes SSH terminál. Zapojení *Raspberry Pi* je tedy velmi jednoduché. Stačí, aby bylo připojeno do sítě pomocí ethernetového kabelu, do zdroje napájení a

připojenou SD kartou s vypáleným imagem operačního systému. Alternativní zapojení pro mobilní fungování je využití Wi-Fi modulu a napájení například z power banky nebo baterie.

Po prvotním zapojení je opět potřeba nainstalovat framework ROS2 a otestovat jeho fungování na úvodních příkladech. Po úspěšném testu je možné tvořit vlastní ROS2 *nody*, které budou publikovat potřebné informace v úvodu kapitoly 4 na obrázku 22 je znázorněné, že *Raspberry Pi* bude publikovat informace o teplotě z prostředí. K tomu je potřeba připojit konkrétní senzor. V práci byl využit senzor *ky-001*, jednoduchý teplotní modul využívající digitální sériový bus. Jeho fungování a ukázka jednoduchého kódu je uvedena zde: [49]. Tento kód, můžeme s malými úpravami přímo využít a integrovat do našeho ROS2 *publishera*, neboť je již psaný v *Pythonu*, který ROS2 také podporuje.



Obrázek 27: Teplotní senzor ky-001

Ukázka kódu publikujícího teplotu:

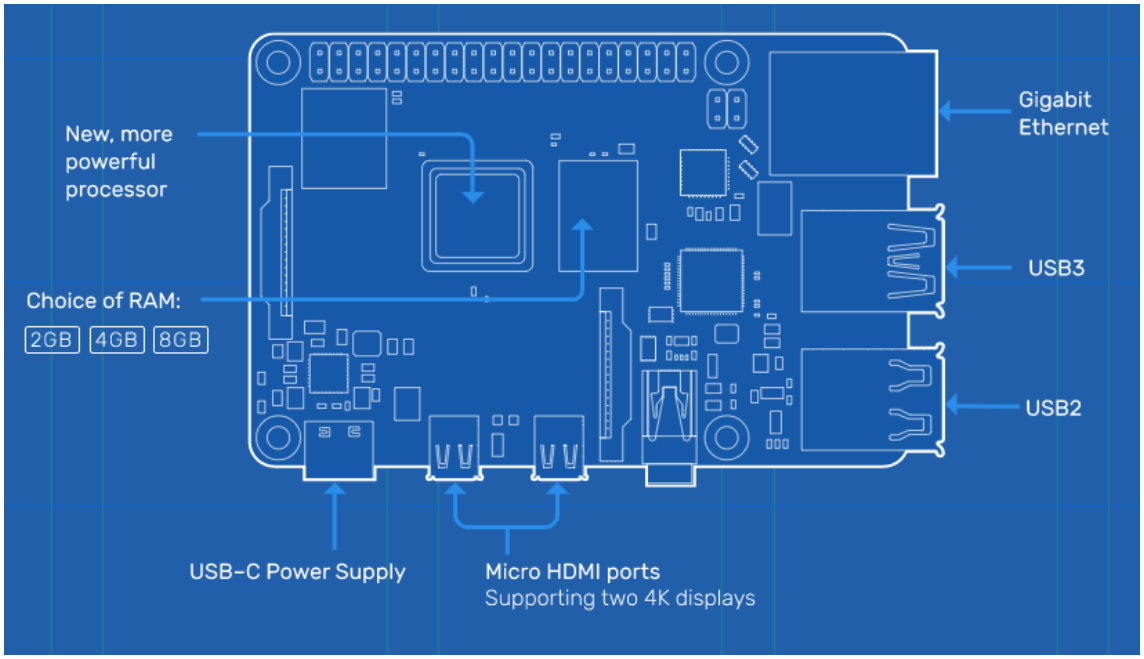
```
#!/usr/bin/env python3
import rclpy
import os
import re
import sys
import time
from rclpy.node import Node

from example_interfaces.msg import Int64

class TemperatureSensorNode(Node):
```

```
    def __init__(self):
        super().__init__("temperature_sensor")
        self.temperature_publisher_ = self.create_publisher(
            float, "temperature", 10)
        self.temperature_timer_ = self.create_timer(
            2.0, self.publish_temperature)

    def publish_temperature(self):
        ky001_filename = None
        dev_path = '/sys/bus/w1/devices/'

        if len(os.listdir(dev_path)) < 1:
            sys.exit('Have you updated config.txt and run modprobe?')

        pattern = '^[1-9][1-9]-[A-Za-z0-9]*$'  # File pattern like 28-20320c40bf7c

        for filename in os.listdir(dev_path):
            if re.match(pattern, filename):  # If filename match the pattern
                print("Using file '{}'".format(filename))
                ky001_filename = filename

        if not ky001_filename:
            sys.exit("Couldn't find any file matching requirements.")

        try:
            ky001_file = open("{}{}/w1_slave".format(dev_path, ky001_filename))
            data = ky001_file.read()
            ky001_file.close()
            temp_c = float(data.split(" t=")[1].strip()) / 1000
            temp_f = round(temp_c * 1.8 + 32, 2)
            print('Temp:  {}ÂşC  {}ÂşF  (CTRL+C to exit)'.format(temp_c, temp_f))
            time.sleep(1.5)
        except KeyboardInterrupt:
            pass

        temperature = temp_c
        msg = Int64()
        msg.data = temperature
        self.temperature_publisher_.publish(msg)

def main(args=None):
    rclpy.init(args=args)
    node = TemperatureSensorNode()
    rclpy.spin(node)
    rclpy.shutdown()

if __name__ == "__main__":
    main()
```

V ukázce kódu vidíme postupně od začátku import potřebných závislostí jako *sys* (systém) nebo *time* (čas). Následně je zde definovaná třída jakožto *node*, která v inicializační metodě (*init*) definuje *publishera* a časovač, který je navázaný na druhou metodu této třídy (*publish_temperature*). V této druhé metodě se provede inicializace propojení senzoru, nalezení souboru, kam senzor ukládá data, jeho načtení a získání konkrétní hodnoty. Program je připraven pro použití jak v teplotních stupních Celsia, tak Fahrenheita, pro změnu stačí přepsat řádek: *„temperature = temp_c“* na *„temperature = temp_f“*. Jakmile máme teplotu získanou, uložíme informaci do data zprávy, kterou ROS *publisher* následně posílá do sítě.

Tento kód jednoduchým příkazem spustíme a můžeme informace odebírat na našem desktopovém klientovi.

```
~$ ros2 run ros2_ws temperature_sensor
```



Obrázek 28: Schéma Raspberry Pi 4
Zdroj: [50]

Hardwarová specifikace Raspberry Pi 4: [50]

- Čtyřjádrový procesor *Broadcom* BCM2711, 1.5 GHz
- RAM: 4 GB
- 4x USB (2x USB 3.0, 2x USB 2.0)
- Gigabit Ethernet
- 40 pinové GPIO pole (např. pro připojení senzorů)
- 2x micro-HDMI (s podporou 4k)
- 2x konektor pro CSI/DSI kameru
- USB C pro napájení
- Slot na micro SD kartu
- Integrovaný Wi-Fi a Bluetooth přijímač

5.10 Instalace a konfigurace ROS2 – Ubuntu desktop PC

Instalace ROSu na desktopové stanici je shodná jako na předchozích dvou zařízení. Jako hostitelský systém byl opět použit *Ubuntu* ve verzi 20.04 a ROS2 v distribuci *Foxy Fitzroy*. *Ubuntu* bylo instalováno klasickým způsobem z instalačního balíčku umístěného na flash disku staženého z oficiálního zdroje.

Desktop v našem schématu na obrázku 22 hraje roli „správce“ informací, který sbírá data publikovaná ostatními zařízeními v našem ekosystému. Od zařízení *Jetson* sbírá vizuál z kamery a od *Raspberry Pi* získává informace o okolní teplotě. Tyto informace následně ukládá do databáze, do které umožňuje uživatelům přistoupit a informace statisticky vyhodnocovat. Jak jsme se dozvěděli v přechozích kapitolách, ROS2 funguje na principu *subscriber* a *publisher*. Pokud jsme tedy na *Jetsonu* a *Raspberry Pi* spustili *publishery*, na desktopu spustíme *subscriber nody*.

Ukázku kódu si představíme níže:

```
import rclpy
from rclpy.node import Node

from std_msgs.msg import String


class TemperatureSubscriber(Node):

    def __init__(self):
        super().__init__('temperature_subscriber')
        self.subscription = self.create_subscription(
            float,'temperature', self.listener_callback, 10)
        self.subscription  # prevent unused variable warning

    def listener_callback(self, msg):
        self.get_logger().info('Temperature: "%s"' % msg.data "C")


def main(args=None):
    rclpy.init(args=args)

    temperature_subscriber = TemperatureSubscriber()

    rclpy.spin(minimal_subscriber)

    temperature_subscriber.destroy_node()
    rclpy.shutdown()


if __name__ == '__main__':
    main()
```

Tato část kódu nám získává data z *topicu temperature*, kam posílá informace *Raspberry Pi*. Na začátku kódu jsou opět definované závislosti nutné k funkcionalitě. Následně máme definovanou třídu *TemperatureSubscriber*, kde jsou nastaveny parametry *topicu* (*temperature*) a datový typ zprávy (*float*). Dále vidíme výpis do konzole pro manuální kontrolu.

5.11 ROS2 v Docker kontejneru

V kapitole 3.1.4 jsme uvedli, že ROS2 může běžet i v kontejnerové virtualizaci, například v prostředí *Docker*. Proces spuštění ROSu je velmi jednoduché a skládá se

z několika málo příkazů. Oficiální dokumentace ROS2 poskytuje podrobný návod pro spuštění vždy konkrétní distribuce. [51]

Výhoda spuštění frameworku ROS v kontejneru je, že nemusíme procházet instalací celého *frameworku*, ale využijeme takzvaný image dostupný na *dockerhubu* [52] (místo pro sdílení kontejnerových předpřipravených systémů).

Postup spuštění je následující:

Získáme image z dockerhubu:

```
~$ docker pull osrf/ros:foxy-desktop
```

Spustíme image v Dockeru (předpokládá se, že Docker je na hostitelském systému nainstalován, pokud ne, spustíme:

```
~$ apt-get install docker
```

Pokud již Docker nainstalovaný máme, provedeme příkaz:

```
~$ docker run -it osrf/ros:foxy-desktop
```

A to je vše. Nyní můžeme v rámci kontejneru spouštět všechny ROS2 příkazy jako po instalaci na hostitelský systém pomocí stahovatelného balíčku. Pro ověření funkcionality je možné zpustit ukázkový příklad.

```
~$ ros2 run demo_nodes_cpp talker
```

A v druhém terminálu spustíme druhý Docker kontejner opět pomocí příkazu:

```
~$ docker run -it osrf/ros:foxy-desktop
```

A následně spustíme ROS2 node:

```
~$ ros2 run demo_nodes_cpp listener
```

Pro zautomatizování celého procesu můžeme vytvořit soubor docker-compose. Tento soubor má v sobě uloženou konfiguraci, co vše se má spustit. Pro příklad uvedeme spuštění dvou instancí Dockeru, kde na jedné se spustí ukázkový *publisher* a na druhém ukázkový *subscriber*.

```
version: '2'

services:
  talker:
    image: osrf/ros:foxy-desktop
    command: ros2 run demo_nodes_cpp talker
  listener:
    image: osrf/ros:foxy-desktop
    command: ros2 run demo_nodes_cpp listener
    depends_on:
      - talker
```

Spuštění kontejnerů definovaných v souboru docker-compose.yaml provedeme pomocí následujícího příkazu. Je nutné nacházet se ve složce se souborem:

```
~$ docker-compose up
```

6 Shrnutí výsledků

V teoretické části došlo k porovnání dvou různých metod komunikace a přístupu robotického fungování. Dozvěděli jsme se o výhodách použití blockchainu, který se zdá být velmi efektivním při použití v autonomním prostředí velkého množství robotických zařízení, neboť nabízí velmi dobrou škálovatelnost a hodí se na použití v rojové robotice. Naopak robotický framework ROS2 poskytuje uživatelům možnost pustit se rovnou do práce s robotickými zařízeními, bez nutnosti složité konfigurace na různorodých platformách. Možnost využití již naprogramovaných modulů od široké základky uživatelů ROSu nám velmi urychlí práci.

V praktické části jsme si ověřili, jakým způsobem funguje framework ROS2 a jaká je jeho myšlenka v distribuovaném světě robotiky. Na jednoduchém ukázkovém modelu jsme si představili komunikace na různých hardwarových architekturách jako je *amd64* (klasické PC) nebo ARM (*Raspberry Pi, Nvidia Jetson*). V ukázce je také popsán postup při práci s kontejnerovou virtualizací v *Dockeru* a výhody jejího využití. ROS2 je jeden z moderních systémů, který nabízí abstraktní vrstvu pro vývoj komplexních robotických zařízení. Jeho funkcionalita se neustále rozvíjí a s každou novou distribucí lze očekávat malé či velké novinky. V budoucnu pravděpodobně uvidíme kompletní předělání frameworku a vydání něčeho jako ROS3.

Obrázek 29: Zařízení využita v práci

7 Závěry a doporučení

Přestože práce se zabývá převážně *frameworkem*/systémem ROS2, existují i další prostředníci pro tvorbu robotických ekosystémů. ROS je momentálně nejvíce rozšířen mezi obecné povědomí lidí, kteří se o robotiku alespoň trochu zajímají. V diplomové práci jsme si objasnili principy distribuovaných systémů a jejich využití v robotice. Tyto principy jsme si vyzkoušeli na jednoduché ukázce, kde mezi sebou komunikovala tři různá zařízení. Klasický desktop, mikro počítač *Raspberry Pi* a *Nvidia Jetson*. K *Raspberry Pi* byl připojen jednoduchý teplotní senzor, který publikoval informace o teplotě a k *Jetsonu* byla připojena moderní stereo kamera *ZED2*, z které jsme sbírali obrazová data. Data a informace z těchto dvou mikro počítačů odebíral náš stolní počítač, který data shromažďoval a v případě potřeby poskytoval dalším uživatelům, kteří by o informace měli zájem. Případně mohl data ukládat do databáze a z nich vyhodnocovat statistiku. Stolní počítač slouží jako prostředník pro správu ostatních robotů v ekosystému, protože jeho umístění je statické a může být neustále připojen k periferiím pro interakci s uživatelem.

V závěru lze říct, že framework ROS2 lze na základě praktické ukázky doporučit pro účely robotizace. Jeho rozsáhlá a uživatelsky přívětivá dokumentace spolu s ukázkovými příklady nabízí uživatelům rychlý vstup do světa robotiky. K využívání systému stačí základní povědomí o programování a práci s linuxovým prostředím. Velký bonus je samozřejmě znalost jazyka *C++* nebo *Python* a případně zkušenost s různým typem periferních zařízení jako je kamera, dotykový senzor, teplotní čidlo, gyroskop, GPS modul apod. Distribuované systémy rozhodně nabízí nespočet výhod oproti jiným přístupům, jejich fungování ale může být občas složité a je nutné zavádět opatření a praktiky, jak se vyhnout různým kolizím, duplicitě dat nebo nadbytečnému komplikování systému. Distribuovaný ekosystém se nám ale při dobrém návrhu odměňuje stabilním a robustním řešením odolným na poruchy a výpadky.

Při návrhu velkého distribuovaného systému s vícero stejnými či silně podobnými robotickými, které mohou využít chování roje, je určitě dobré zvážit ekosystém blockchain. Reálné použití blockchainu v robotice ale zatím zůstává spíše v experimentální oblasti. Se stále větší popularitou můžeme ale očekávat rostoucí využívání blockchainu v aplikacích komerčního prostředí.

8 Seznam zdrojů

[1] ARISTOTELES a Antonín KRIZ. *Politika*. Praha: Rezek, 1998. ISBN 978-80-86027-10-4.

[2] *Leonardo's Robot: Leonardo da Vinci's Mechanical Knight and Other Robots* - [online]. 30. květen 2019 [vid. 2022-01-30]. Dostupné z: https://www.ststworld.com/leonardos-robot/

[3] MCKERROW, Phillip John. Robotics, an academic discipline? *Robotics* [online]. 1986, **2**(3), 267–274. ISSN 0167-8493. Dostupné z: doi:10.1016/0167-8493(86)90035-5

[4] ASIMOV, Isaac, Oldrich CERNÝ, Alexandr KRAMER a Zuzana MEYEROVÁ. *Já, robot*. Praha: Triton : Argo, 2012. ISBN 978-80-7387-491-9.

[5] HEMAL, Ashok Kumar a Mani MENON, ed. *Robotics in Genitourinary Surgery* [online]. London: Springer, 2011 [vid. 2022-04-20]. ISBN 978-1-84882-113-2. Dostupné z: doi:10.1007/978-1-84882-114-9

[6] *Unimate - ROBOTS: Your Guide to the World of Robotics* [online]. [vid. 2022-01-30]. Dostupné z: https://robots.ieee.org/robots/unimate/

[7] *Shakey - CHM Revolution* [online]. [vid. 2022-04-20]. Dostupné z: https://www.computerhistory.org/revolution/artificial-intelligence-robotics/13/289

[8] *Často kladené otázky - Kybernetický nůž Cyberknife* [online]. [vid. 2022-01-30]. Dostupné z: https://cyberknife.fno.cz/cs/clanky/casto-kladene-otazky

[9] CyberKnife. *Stargen* [online]. [vid. 2022-04-20]. Dostupné z: https://www.stargen-eu.cz/radioterapie/cyberknife/

[10] *Description of the Rover Sojourner* [online]. [vid. 2022-04-20]. Dostupné z: https://mars.nasa.gov/MPF/rover/descrip.html

[11] MARS.NASA.GOV. *Mars 2020 Perseverance Rover* [online]. [vid. 2022-02-20]. Dostupné z: https://mars.nasa.gov/mars2020/

[12] *Tesla Gigafactory | Tesla* [online]. [vid. 2022-04-24]. Dostupné z: https://www.tesla.com/gigafactory

[13] FERRIS, Daniel P. The exoskeletons are here. *Journal of NeuroEngineering and Rehabilitation* [online]. 2009, **6**(1), 17. ISSN 1743-0003. Dostupné z: doi:10.1186/1743-0003-6-17

[14] Meet HAL - Hybrid Assistive Limb - RoboFit - Rehabilitation Centre. *RoboFit* [online]. [vid. 2022-04-24]. Dostupné z: https://robofit.com.au/meet-hal/

[15] SAHNI, Yuvraj, Jiannong CAO a Shan JIANG. Middleware for Multi-robot Systems. In: Habib M. AMMARI, ed. *Mission-Oriented Sensor Networks and Systems: Art and Science: Volume 2: Advances* [online]. Cham: Springer International Publishing, 2019 [vid. 2022-04-03], s. 633–673. ISBN 978-3-319-92384-0. Dostupné z: doi:10.1007/978-3-319-92384-0_18

[16] Centralized vs Decentralized vs Distributed Systems · Berty Technologies. *Berty Technologies* [online]. [vid. 2022-02-19]. Dostupné z: https://berty.tech/blog/decentralized-distributed-centralized

[17] ZHANG, Yuan. Blockchain. In: Xuemin (Sherman) SHEN, Xiaodong LIN a Kuan ZHANG, ed. *Encyclopedia of Wireless Networks* [online]. Cham: Springer International Publishing, 2020 [vid. 2022-04-28], s. 115–118. ISBN 978-3-319-78262-1. Dostupné z: doi:10.1007/978-3-319-78262-1_171

[18] ROS…in…space! *Open Robotics* [online]. [vid. 2022-02-19]. Dostupné z: https://www.openrobotics.org/blog/2022/2/2/rosinspace

[19] HU, Guoqiang, Wee Peng TAY a Yonggang WEN. Cloud Robotics: Architecture, Challenges and Applications. *IEEE Network - NETWORK* [online]. 2012, **26**, 21–28. Dostupné z: doi:10.1109/MNET.2012.6201212

[20] SHAKYA, Subarna. Survey on Cloud Based Robotics Architecture, Challenges and Applications. *Journal of Ubiquitous Computing and Communication Technologies* [online]. 2020, **2**, 10–18. Dostupné z: doi:10.36548/jucct.2020.1.002

[21] VALKOV, Ivan, Phil TRINDER a Natalia CHECHINA. Reliable distribution of computational load in robot teams. *Autonomous Robots* [online]. 2021, **45**(3), 351–369. ISSN 1573-7527. Dostupné z: doi:10.1007/s10514-021-09967-8

[22] MUNERA, Eduardo, Jose-Luis POZA-LUJÁN, Juan-Luis POSADAS-YAGÜE, J. SIMÓ a J. F. B. NOGUERA. Distributed Real-time Control Architecture for ROS-based Modular Robots [online]. 2017. Dostupné z: doi:10.1016/J.IFACOL.2017.08.1600

[23] LYAZID, Sabri. Internet of Robot Things in a Dynamic Environment: Narrative-Based Knowledge Representation and Reasoning. In: Takahiro HARA a Hirozumi YAMAGUCHI, ed. *Mobile and Ubiquitous Systems: Computing, Networking and Services* [online]. Cham: Springer International Publishing, 2022, s. 520–526. Lecture Notes of the Institute for Computer Sciences, Social Informatics and Telecommunications Engineering. ISBN 978-3-030-94822-1. Dostupné z: doi:10.1007/978-3-030-94822-1_33

[24] A History of ROS (Robot Operating System). *The Construct* [online]. 9. červenec 2019 [vid. 2022-02-01]. Dostupné z: https://www.theconstructsim.com/history-ros/

[25] *Understanding ROS 2 nodes — ROS 2 Documentation: Foxy documentation* [online]. [vid. 2022-03-25]. Dostupné z: https://docs.ros.org/en/foxy/Tutorials/Understanding-ROS2-Nodes.html

[26] Getting Started with ROS 2. *Ubuntu* [online]. [vid. 2022-02-11]. Dostupné z: https://ubuntu.com/tutorials/getting-started-with-ros-2

[27] *Sending Commands from rviz — Dev documentation* [online]. [vid. 2022-04-28]. Dostupné z: https://ardupilot.org/dev/docs/ros-rviz.html

[28] rqt plot - Easily Debug ROS topics. *The Robotics Back-End* [online]. 8. červenec 2019 [vid. 2022-04-20]. Dostupné z: https://roboticsbackend.com/rqt-plot-easily-debug-ros-topics/

[29] rqt graph - Visualize and Debug Your ROS Graph. *The Robotics Back-End* [online]. 15. červenec 2019 [vid. 2022-02-08]. Dostupné z: https://roboticsbackend.com/rqt-graph-visualize-and-debug-your-ros-graph/

[30] *Using rqt_console — ROS 2 Documentation: Foxy documentation* [online]. [vid. 2022-04-20]. Dostupné z: https://docs.ros.org/en/foxy/Tutorials/Rqt-Console/Using-Rqt-Console.html

[31] SCHIERENEBCK, Tom. Creating a Gazebo with ROS2 for you own robot. *Creating a Gazebo Simulation with ROS2 for your own robot* [online]. 14. říjen 2020 [vid. 2022-02-08]. Dostupné z: https://medium.com/creating-a-gazebo-simulation-with-ros2-for-your/introduction-8daf6efa12f4

[32] The ROS Development Studio by The Construct. *The Construct* [online]. 18. únor 2017 [vid. 2022-02-06]. Dostupné z: https://www.theconstructsim.com/the-ros-development-studio-by-the-construct/

[33] DILLENBURG, Stefan. A brief history of container virtualization. *An Idea (by Ingenious Piece)* [online]. 15. říjen 2020 [vid. 2022-02-25]. Dostupné z: https://medium.com/an-idea/a-brief-history-of-container-virtualization-57fc96c02924

[34] DEVELOPMENT, Bonsai. Kontejnery a virtualizace | Bonsai Development | Web na míru. *Bonsai Development* [online]. [vid. 2022-02-26]. Dostupné z: https://bonsai-development.cz/clanek/kontejnery-a-virtualizace

[35] ROS 2 and Kubernetes Basics. *Ubuntu* [online]. [vid. 2022-04-01]. Dostupné z: https://ubuntu.com/blog/exploring-ros-2-with-kubernetes

[36] FERRER, Eduardo Castelló. The blockchain: a new framework for robotic swarm systems. In: [online]. 2019 [vid. 2022-07-30], s. 1037–1058. Dostupné z: doi:10.1007/978-3-030-02683-7_77

[37] KAISER, Tanja Katharina, Marian Johannes BEGEMANN, Tavia PLATTENTEICH, Lars SCHILLING, Georg SCHILDBACH a Heiko

HAMANN. ROS2SWARM - A ROS 2 Package for Swarm Robot Behaviors. In: *2022 International Conference on Robotics and Automation (ICRA)* [online]. Philadelphia, PA, USA: IEEE, 2022, s. 6875–6881 [vid. 2022-08-07]. ISBN 978-1-72819-681-7. Dostupné z: doi:10.1109/ICRA46639.2022.9812417

[38] VERMESAN, Ovidiu, Arne BRÖRING, Elias TRAGOS, Martin SERRANO, Davide BACCIU, Stefano CHESSA, Claudio GALLICCHIO, Alessio MICHELI, Mauro DRAGONE, Alessandro SAFFIOTTI, Pieter SIMOENS, Filippo CAVALLO a Roy BAHR. Internet of Robotic Things – Converging Sensing/Actuating, Hyperconnectivity, Artificial Intelligence and IoT Platforms. In: [online]. 2017, s. 97–155. ISBN 978-87-93609-10-5. Dostupné z: doi:10.13052/rp-9788793609105

[39] INSTITUTE, Toyota Research. Drake: Model-based design in the age of robotics and machine learning. *Toyota Research Institute* [online]. 3. květen 2021 [vid. 2022-04-20]. Dostupné z: https://medium.com/toyotaresearch/drake-model-based-design-in-the-age-of-robotics-and-machine-learning-59938c985515

[40] *MOOS : Main - Introduction browse* [online]. [vid. 2022-08-15]. Dostupné z: https://www.robots.ox.ac.uk/~mobile/MOOS/wiki/pmwiki.php/Main/Introduction

[41] KEXUGIT. *Robotics: Simulating the World with Microsoft Robotics Studio* [online]. [vid. 2022-08-15]. Dostupné z: https://docs.microsoft.com/en-us/archive/msdn-magazine/2008/june/robotics-simulating-the-world-with-microsoft-robotics-studio

[42] Robotics security: What is SROS 2? *Ubuntu* [online]. [vid. 2022-08-07]. Dostupné z: https://ubuntu.com/blog/what-is-sros-2

[43] Jetson Nano Developer Kit. *NVIDIA Developer* [online]. 6. březen 2019 [vid. 2022-03-24]. Dostupné z: https://developer.nvidia.com/embedded/jetson-nano-developer-kit

[44] *ZED 2 - AI Stereo Camera* [online]. [vid. 2022-03-24]. Dostupné z: https://www.stereolabs.com/zed-2/

[45] What is a Raspberry Pi? *Raspberry Pi* [online]. [vid. 2022-03-31]. Dostupné z: https://www.raspberrypi.org/help/what-is-a-raspberry-pi/

[46] *Installing ROS 2 via Debian Packages — ROS 2 Documentation: Foxy documentation* [online]. [vid. 2022-04-05]. Dostupné z: https://docs.ros.org/en/foxy/Installation/Ubuntu-Install-Debians.html

[47] Q-ENGINEERING. *Install Ubuntu 20.04 on Jetson Nano - Q-engineering* [online]. [vid. 2022-03-25]. Dostupné z: https://qengineering.eu/install-ubuntu-20.04-on-jetson-nano.html

[48] *Getting Started with ROS 2 and ZED | Stereolabs* [online]. [vid. 2022-03-25]. Dostupné z: https://www.stereolabs.com/docs/ros2/

[49] *KY-001: Temperature sensor :: TkkrLab Arduino cursus* [online]. [vid. 2022-04-27]. Dostupné z: https://arduino.tkkrlab.space/sensoren/ky-001/

[50] LTD, Raspberry Pi. Raspberry Pi 4 Model B specifications. *Raspberry Pi* [online]. [vid. 2022-04-01]. Dostupné z: https://www.raspberrypi.com/products/raspberry-pi-4-model-b/

[51] *Running ROS 2 nodes in Docker [community-contributed] — ROS 2 Documentation: Foxy documentation* [online]. [vid. 2022-04-19]. Dostupné z: https://docs.ros.org/en/foxy/How-To-Guides/Run-2-nodes-in-single-or-separate-docker-containers.html

[52] *osrf/ros2 - Docker Image | Docker Hub* [online]. [vid. 2022-04-19]. Dostupné z: https://hub.docker.com/r/osrf/ros2/

9 Seznam obrázků

10 Seznam tabulek

Univerzita Hradec Králové
Fakulta informatiky a managementu

Zadání diplomové práce

Autor: **Bc. David Voda**

Studium: I2000795

Studijní program: N1802 Aplikovaná informatika

Studijní obor: Aplikovaná informatika

Název diplomové práce: **Distribuované robotické systémy**

Název diplomové práce AJ: Distributed robotic systems

Cíl, metody, literatura, předpoklady:

Cíl: Představit a porovnat vybrané metody komunikace a zpracování dat u distribuovaných výpočetních systémů

Osnova:

Centralizované a distribuované architektury

Porovnání metod komunikace

Ověření

Shrnutí výsledků, závěr

CHECHINA, Natalia. Reliable distribution of computational load in robot teams. *Autonomous Robots*, 2021.

SHAKYA, Subarna. Survey on Cloud Based Robotics Architecture, Challenges and Applications. *Journal of Ubiquitous Computing and Communication Technologies (UCCT)*, 2020, 2.01: 10-18.

MUNERA, Eduardo, et al. Distributed real-time control architecture for ROS-based modular robots. *IFAC-PapersOnLine*, 2017, 50.1: 11233-11238.

Garantující pracoviště: Katedra informačních technologií,
Fakulta informatiky a managementu

Vedoucí práce: Ing. Karel Mls, Ph.D.

Datum zadání závěrečné práce: 15.10.2021