VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INTELIGENTNÍCH SYSTÉMŮ
DEPARTMENT OF INTELLIGENT SYSTEMS

MOBILNÍ KAMERA REALIZOVANÁ PROSTŘEDKY ROS2

THESIS TITLE

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **DANIEL ONDERKA**
AUTHOR

VEDOUCÍ PRÁCE **doc. Ing. VLADIMÍR JANOUŠEK, Ph.D.**
SUPERVISOR

BRNO 2023

Abstrakt

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v českém (slovenském) jazyce.

Abstract

Do tohoto odstavce bude zapsán výtah (abstrakt) práce v anglickém jazyce.

Klíčová slova

Sem budou zapsána jednotlivá klíčová slova v českém (slovenském) jazyce, oddělená čárkami.

Keywords

Sem budou zapsána jednotlivá klíčová slova v anglickém jazyce, oddělená čárkami.

Citace

ONDERKA, Daniel. *Mobilní kamera realizovaná prostředky ROS2*. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Vladimír Janoušek, Ph.D.

Mobilní kamera realizovaná prostředky ROS2

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana X... Další informace mi poskytli... Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Daniel Onderka
22. prosince 2023

Poděkování

V této sekci je možno uvést poděkování vedoucímu práce a těm, kteří poskytli odbornou pomoc (externí zadavatel, konzultant apod.).

Obsah

Seznam obrázků

Kapitola 1

Úvod

Kapitola 2

Použitý Hardware

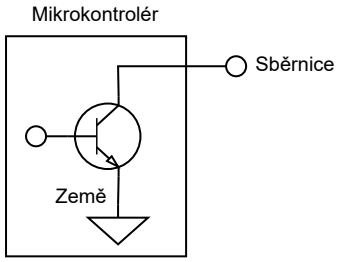
Celá práce je implementována nad existujícím hardwarem. Konkrétně se jedná o stavebnici Adeept AWR 4WD. Její součástí jsou všechny použité motory, serva, čidla a další periferie. Mozkem na kterém poběží software zajišťující ovládání těchto komponent bude mikropočítač Raspberry Pi 4.

HW Technologie

Nejprve budou představeny obecné hardwarové technologie které jsou následně využívány některými z představovaných periferií.

I2C

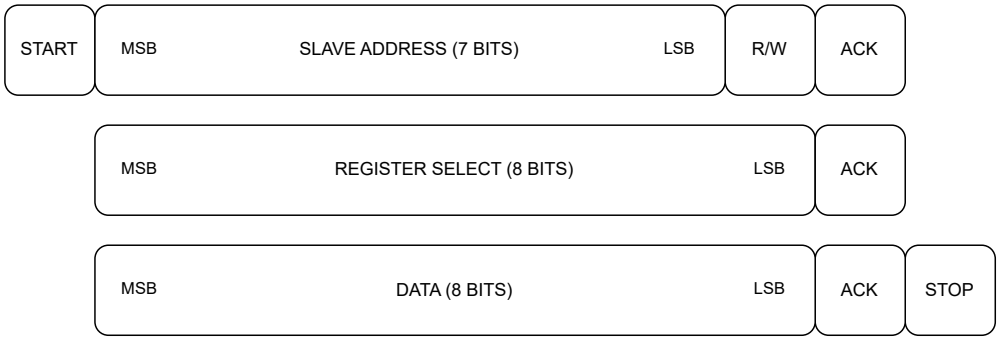
Je synchronní sběrnice, která se vyznačuje svou jednoduchostí a nízkou cenou. Využívá dva vodiče SDA (serial data) a SCL (serial clock). Oba vodiče jsou připojeny k napájecímu napětí pomocí pull-up rezistoru a bez vlivu jiného hardwaru zůstávají v logické jedničce. Zařízení která jsou na tuto sběrnici připojeny využívají open drain k úpravě aktuální napěťové úrovně. I2C pak pracuje s dvěma druhy zařízení, master a slave. Master zahajuje, řídí a ukončuje komunikaci na vodiči SDA a po dobu průběhu komunikace generuje hodinový signál na SCL. Typicky se jedná o mikrokontroler. Slave jsou pak ostatní zařízení s nimiž může master komunikovat, typicky různé periferie. [1]



Obrázek 2.1: Open drain

Přenos jednoho datového rámce zahájí master zařízení přivedením datové sběrnice do nuly. Následující komunikace se skládá z odeslání rámce o délce osmi bitů a potvrzení o úspěšném přenosu dat od přijímajícího zařízení. Toto potvrzení se nazývá ACK a je provedeno podržením datové sběrnice v hodnotě nula po dobu jednoho taktu. Opačný stav

se nazývá NACK a indikuje že nastala chyba. Ukončení přenosu je provedeno navrácením datové sběrnice na hodnotu jedna. [2]



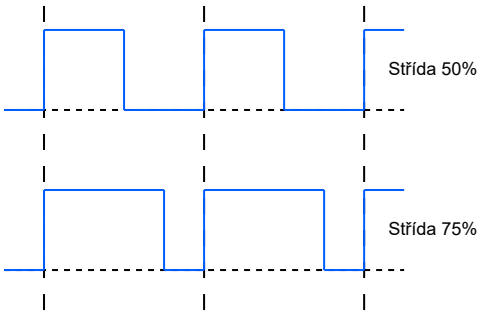
Obrázek 2.2:

Na obrázku lze vidět, jak může vypadat přenos jednoho datového slova. V prvním rámci je přenesena sedmi bitová adresa, identifikující slave zařízení se kterým chce máster navázat komunikaci. Osmý bit datového rámce indikuje směr, kterým budou posílány data. V druhém rámci dojde k adresaci konkrétního registru na slave zařízení. A ve třetím, případně dalších, již probíhá samotné posílání dat mezi zařízeními. [5]

Pulzně šířková modulace

Jedná se o techniku, která umožňuje vytvořit pseudo-analogový výstup s použitím číslicových pinů. Mikrokontroléry jsou digitální zařízení a chtěly by tedy s okolním světem komunikovat pomocí číslicových pinů. Reálný svět však nefunguje pouze na úrovni jedniček a nul a proto je často potřeba převádět výstup z mikrokontroléru na analogový signál. Problém je v tom, že převod digitálního signálu na analogový je relativně dlouhá a neefektivní operace. Proto vznikla pulzně šířková modulace (PWM), která umožňuje relativně jednoduše simulovat analogový výstup. [1]

PWM využívá toho, že člověk nedokáže rozpoznat rychlé změny, například led dioda blikající na frekvenci 5000 Hz se člověku jeví jako by svítila permanentně. Mechanická zařízení také mívají relativně velkou latenci a dc motoru tedy nevadí, že místo konstantního analogového napětí dostává periodický číslicový signál.



Obrázek 2.3:

Při pohledu na klasický digitální signál který rovnoměrně střídá vysokou a nízkou úroveň by šlo říci, že se jedná o PWM signál se střídou 50%. Střída (duty cycle) udává poměr času,

kdy je signál v logické jedničce, ku času, kdy je v nule. Součet těchto hodnot se musí rovnat délce jedné periody. Úpravou tohoto poměru lze simulovat analogový signál.

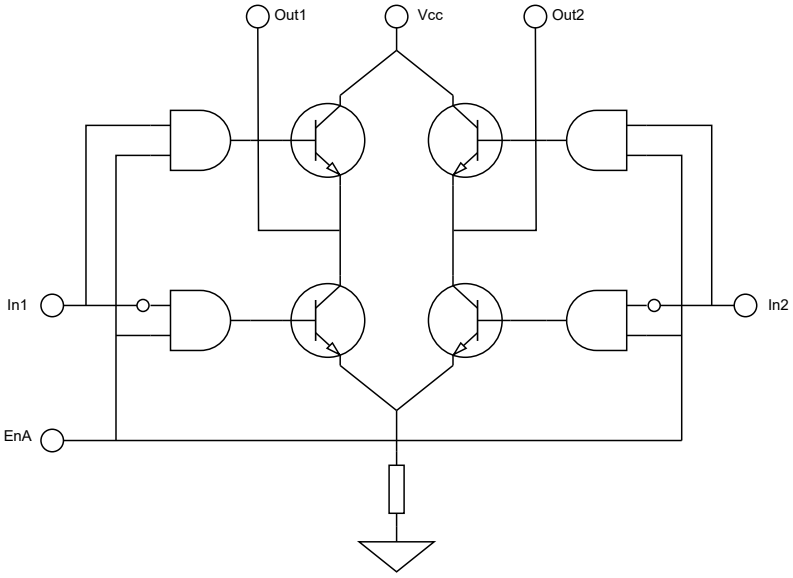
Adeept AWR 4WD

Pohyb tohoto robota zajišťují čtyři pevná kola a zatáčení je tedy realizováno diferenciálním způsobem. To znamená, že jednotlivá kola se mohou otáčet svou vlastní rychlostí a zpomalením kol na jedné straně oproti té druhé lze provést zatáčení robota. Velkou výhodou tohoto přístupu je možnost otáčení robota na místě, naopak nevýhodou pak budou složitější výpočty pokud je potřeba zatočit pod konkrétním úhlem.
Následující stránky popisují jednotlivé komponenty tohoto robota.

Robot HAT

HAT(hardware attached on top) je hardwarová deska, která slouží k rozšíření mikrokontroléru o další funkcionalitu. Tato konkrétní se k Raspberyy Pi připojuje pomocí GPIO(General purpuse input outpu) pinů. Deska jako taková obsahuje rozšiřující čipy a rozhraní sloužící k ovládání připojených periferií.

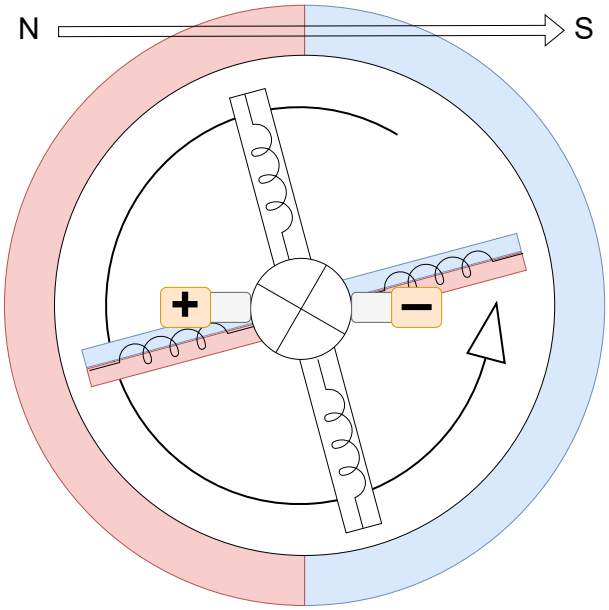
- PCA9685 [3]
  - generátor PWM signálu
  - 16 kanálů
  - 12 bitů rozlišení střídy (4096 možných hodnot)
  - je ovládaný přes I2C sběrnici
- L298P
  - ovladač pro řízení dc motoru
  - obsahuje full bridge obvod (viz následující obrázek)
  - umožňuje roztočit motor oběma směry
  - pomocí PWM lze ovládat rychlost motorů
  - připojuje motor na externí napájení
- další rozhraní pro připojení periferií (line tracking)



Obrázek 2.4: Full bridge konfigurace pro ovládání motoru. In1 a In2 určují směr otáčení. EnA je PWM signál určující rychlost otáčení. [6]

DC Motor

Pohyb celého autíčka zajišťují čtyři stejnosměrným proudem(direct current) napájené motory. Motor driver L298P sloužící k ovládání motorů je umístěný na Robot HAT.



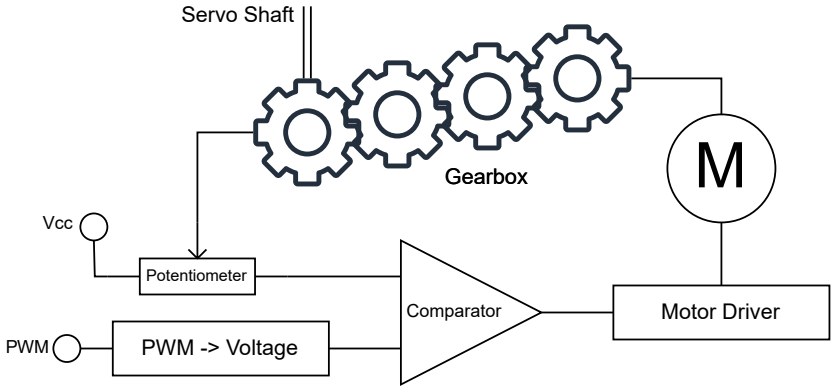
Obrázek 2.5: Dc motor

Elektrický DC motor se skládá ze dvou hlavních částí, stator a rotor. Stator je statická, vnější část, a typicky se jedná o permanentní magnet. Uvnitř statoru se pak nachází rotor, ten se skládá z elektromagnetů, které při zapnutí reagují se statorem(póly se odpuzují a přitahují) a dojde tak k částečnému pootočení. Při správném spínání a vypínání těchto mag-

netů lze motor rozběhnout. To zajišťuje komutátor, jedná se o prstenec složený z několika navzájem odizolovaných částí, které jsou připojeny na vývody elektromagnetu. S povrchem prstence jsou pomocí pružin v kontaktu dva kartáče, které jsou připojeny na napájení a zemi. Komutátor se otáčí společně s rotorem a při tomto pohybu se kartáče postupně dotýkají různých částí komutátoru a spínají tak jednotlivé elektromagnety, ty vedou k pootočení rotoru a sepnutí následujícího magnetu. [1]

Servo

Servo je komponenta podobná DC Motoru, jejíž hlavní výhodnou je možnost přesně určit úhel natočení. Na rozdíl od něj se však neotáčí celých 360 stupňů, ale bývá omezena na cca 180 stupňů.



Obrázek 2.6:

Při pohledu na vnitřní zapojení serva lze zjistit, že se prakticky jedná o klasický dc motor připojený na převodovku a rozšířený o elektroniku na jeho řízení. K nastavení úhlu serva se využívá PWM signál. Ten je první přeložený na napěťovou úroveň, která je porovnána s aktuálním natočením serva a výsledek udává směr, kterým se bude otáčet motor. Aktuální natočení serva je získáno využitím potenciometru zapojeného na výstupní hřídel serva. [1]

Adeept AWD 4WD využívá pouze jedno servo, a to na ovládání úhlu natočení kamery. Konkrétně se jedná o model Adeept AD002. Generování PWM signálu pro ovládání zajišťuje čip PCA9685 který je součástí Robot HAT.

Ultrazvukový senzor hloubky

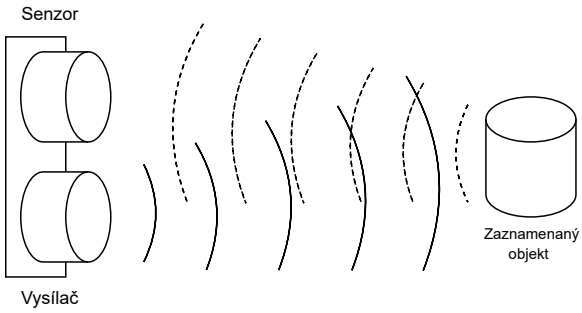
Slouží k určení vzdálenosti mezi robotem a překážkou. Funguje na principu radaru. Vyšle ultrazvukovou vlnu na frekvenci 40Khz a uloží si časovou značku. Následně poslouchá než se mu vrátí odražená vlna a opět si uloží značku. Pro výpočet vzdálenosti lze využít následující vzorec:

$$S = (T_2 - T_1) * V_S/2$$

Kde T_1 je moment kdy byla vyslána vlna T_2 kdy byla vlna přijata a V_S rychlost šíření zvuku ve vzduchu (cca 340m/s). Výsledek se pak dělí dvěma, protože doba $T_2 - T_1$ je rovna času k překážce a zpět.

Konkrétně se jedná o model hc-sr04, který dokáže změřit vzdálenost od 2cm do 400cm s přesností na 3mm. Ovládání senzoru je pak realizováno pomocí dvou jeho vývodů a to trig

a echo. Mikrokontroler vyšle pulz na trig vodiči. Ten zaktivuje senzor, který zahájí měření. To je realizováno osmi čtyřicetihercovými pulzy ze kterých vypočítá výslednou vzdálenost. Po dokončení výpočtu nastaví echo vodič do hodnoty jedna na dobu rovnou času mezi odesláním a zachycením ultrazvukového signálu. [1]



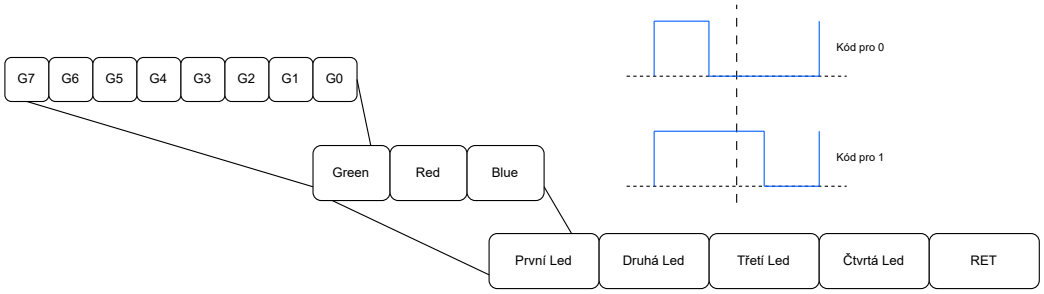
Obrázek 2.7: Ultrasonic

Třícestný senzor pro sledování čáry

Modul využívá fakt, zě intenzita světla odraženého od povrchu je závislá na barvě dané plochy. Například černá barva pohltí téměř veškeré světlo, naopak bílá téměř vše odrazí. Používáno je infračervené záření, protože není ovlivněno okolními zdroji světla, odráží se od velkého množství materiálů a je přesné. Jedná se o třícestný modul a skládá se tedy ze setu tří vysílačů a senzorů. Pokud vysílač svítí a senzor nezaznamenává dostatečnou intenzitu odraženého světla, znamená to, že byla nalezena černá čára. Komunikace s mikrokontrolerem je realizována pomocí GPIO vstupů, na které jsou připojeny tři vývody z modulu. Každý z nich reprezentuje jeden set vysílače a senzoru. Jednička na výstupu indikuje, že čára byla detekována.

WS2812 RGB LED

WS2812 je druh adresovatelných LED diod. Pojmem adresovatelných je myšleno, že není každá dioda připojena k mikrokontroleru zvláště, ale sdílejí jeden datový vodič pro nastavování barev. Prakticky to znamená, že pásek, který může obsahovat i stovky diod je připojený jen pomocí tří vodičů. Těmito vodiči jsou napájení, země a vstupní data.



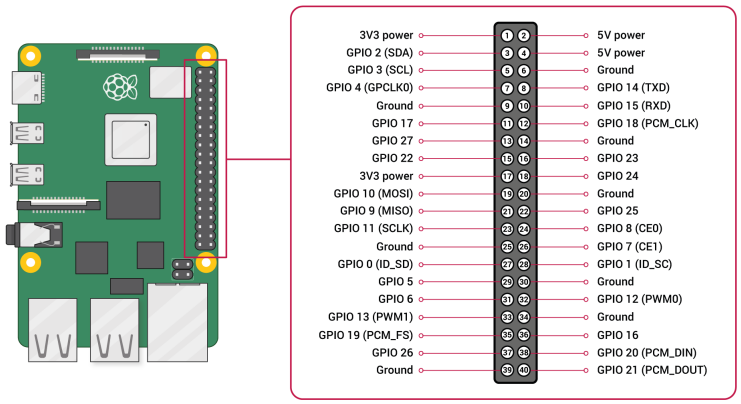
Obrázek 2.8: Ultrasonic

Diody jsou na pásku zapojeny „seriově“, mají DIN a DO porty a přijatá data, která nejsou určena dané diodě přeposílá dále. Komunikace pak vždy začíná klidovým stavem,

datový vodič je v nule. Datové slovo se skládá z 24 bitových bloků pro každou diodu. Blok obsahuje tři osmi bitové hodnoty, jednu pro každou barevnou složku a MSB je posíláno první. Diody pak fungují tak, že přijmou prvních 24 bitů podle kterých nastaví svou barvu. Tuto část odeberou z datového slova a zbytek přeposílají na výstup.

Raspberry Pi 4b

Jako mozek celého systému je použit mikropočítač Raspberry Pi. V porovnání s běžně používanými mikrokontroléry sloužícími pro řízení vestavěných systémů se jedná o výkonnější hardware, který zvládá i komplexnější operace jako běh plnohodnotného operačního systému a zpracování obrazu. Konkrétně se jedná o verzi 4 model B s operační pamětí o velikostí čtyř gigabajtů. Tato verze obsahuje 64bitový procesor, ten je potřeba pro spuštění 64 bitového Ubuntu serveru, který je doporučeným operačním systémem pro požití ROS2 na raspberry pi. Komunikace s většinou použitých periferií je uskutečněna pomocí General Purpuse Input Output(GPIO) pinů. Jedná se o číslicové vývody, které podle potřeby můžou fungovat jako vstup i výstup ze zařízení. Některé z nich pak mají ještě speciální funkce, například GPIO 2 a 3 můžou pracovat jako SDA a SCL připojení pro I2C komunikaci.



Obrázek 2.9: GPIO pinout

Camera

Přímo k rapsberry pi je připojena oficiální rpi camera v3. Tento modul dokáže nahrávat video až v rozlišení 2304 × 1296 pixelů a 56 snímcích za vteřinu. Avšak pro zpracování, případně analýzu obsahu videa v reálném čase s těmito specifikacemi nemá rpi dostatečný výkon. Prakticky budou využity nižší rozlišení a snímkovací frekvence.

Kapitola 3

Software pro řízení robota

Jak už bylo řečeno software poběží na Raspberry Pi. Jako primární programovací jazyk byl zvolen python, protože je jedním z oficiálních jazyků podporovaných ROS2 a také jsou v něm implementovány potřebné knihovny pro ovládání periferií.

3.1 Aktuální software

Robot Adeept AWR 4WD je dodáván s ukázkovým softwarem. Ten je implementován v jazyce Python a využívá knihovny třetích stran sloužící k nízkoúrovňovému ovládání hardwarových komponent. Aby byl robot responzivní je celá implementace řešena s použitím python modulů pro realizaci multithreadingu.

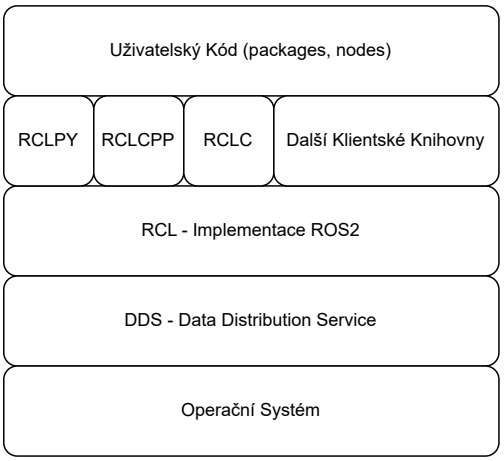
3.2 Seznámení s ROS2

ROS2 je middleware sloužící k vývoji a řízení robotů. Middleware je softwarová vrstva běžící nad operačním systémem. Jejím úkolem je rozšíření operačního systému o další funkcionalitu. Typickou součástí middlewaru bývají knihovny, ovladače, vývojové a monitorovací nástroje. Může také specifikovat doporučené metodologie pro vývoj. ROS2 je již druhá verze tohoto softwaru, která rozšiřuje a opravuje neduhy první verze. Původní ROS1 je považován za de-facto standart pro vývoj robotických aplikací. Tato práce využívá ROS2 distribuci jménem iron. Distribuce v ROS2 lze popsat jako set operačního systému, knihoven a dalších aplikací, které jsou otestovány a je zaručeno, že jsou navzájem kompatibilní. Velkou výhodou ROS je fakt, že se jedná o open source projekt. Díky tomu kolem něj vznikla velká komunita vývojářů, ale i firem a dalších institucí, které tvoří mnoho souvisejícího obsahu. Existuje tedy velké množství knihoven, dokumentací a návodů které usnadňují vývojářům práci. [4]

Vrstvy ROS2

Na nejvyšší úrovni, se nachází programátor který interaguje s klientskými knihovnami pro vývoj ROS2 aplikací. Tyto knihovny jsou oficiálně dvě a to rclpy pro python a rclcpp pro C++. Existují také implementace pro další programovací jazyky (rclc, java, C#), ty jsou však udržovány komunitně. Všechny klientské knihovny pak využívají RCL. RCL je jádro celého ROS implementované v jazyce C. RCL implementuje hlavní ros2 funkcionalitu, kterou následně poskytuje pomocí rozhraní jednotlivým klientským knihovnám. Díky tomuto

se funkcionalita implementovaná v pythonu bude chovat stejně jako ta implementovaná v c++. Taky pak kód implementovaný v c++ může komunikovat s tím v pythonu. Poslední vrstvou je data distribution service. DDS je komunikační vrstva implementována na UDP protokolu sloužící k předávání informací mezi procesy. Má charakteristiky systémů reálného času, zajišťuje kvalitu a zabezpečení komunikace. Také umožňuje vyhledávání uzlů bez potřeby centralizovaného serveru (využívá k tomu multicast, komunikace mezi uzly je následně unicast). [4]



Obrázek 3.1: Layers

3.2.1 Vývoj v ROS2

Nejvyšší organizační jednotkou v ROS2 je workspace. Jedná se o složku, která slouží k organizaci zdrojových souborů, jejich instalaci a následné spouštění. ROS2 instalace je také workspace a před použitím je potřeba ji nejprve aktivovat. K tomu v linuxu slouží příkaz `source`. Aktivace workspace je akumulativní a v jeden moment tedy může být aktivních několik workspace. Typicky se první aktivuje základní ros2 instalace, která tvoří takzvanou underlay vrstvu. Vývojový workspace aktivovaný jako druhý se pak nazývá overlay. Pokud má overlay nějaké závislosti, měly by být uspokojeny v underlay. Zdrojové soubory v rámci workspace jsou pak organizovány do packages. Package může obsahovat zdrojové soubory,

knihovny a definice zpráv. Packages na sobě můžou navzájem záviset (např: package která potřebuje interface závisí na jiné která tento interface definuje). [4]

```
Workspace
├── build       #soubory používané při kompilaci
├── install     #výsledky kompilace a další soubory potřebné ke spuštění
├── log         #logy z kompilace
├── launch      #launch soubory
└── src         #packages
    └── package_name     #příklad jak vypadá python package
        ├── package_name     #zdrojové python soubory
        ├── resource
        ├── test
        ├── package.xml   #metadata infromace o package
        ├── setup.cfg     #konfigurace pro manuální spouštění uzlu
        └── setup.py      #instrukce pro kompilátor jak nainstalovat package
```

Node

Celý ROS2 systém je složený z uzlů (Node), které mezi sebou navzájem komunikují. Každý uzel je vlastní výpočetní jednotka, která by měla plnit jeden specifický úkol. Tento přístup je podobný objektově orientovanému návrhu a ROS2 jej také využívá. Implementačně je tedy uzel objekt, který dědí ze třídy `Node`. Uzly v ROS2 většinou nepotřebují běžet permanentně, ale pouze v monentě, kdy nastane nějaká událost kterou je potřeba obsloužit. Z toho důvodu existuje v ROS2 metoda `spin()`, která uspí vykonávání uzlu, dokud jej není potřeba opět využít. Aby ROS2 šetřil výpočetní prostředky, využívá dva přístupy k určení, kdy bude potřeba uzel vzbudit. Prvním je iterative execution, ten se používá u uzlů, které vykonávají svou činnost pravidelně na nějaké frekvenci. Například se může jednat o výpočetní uzel, který pravidelně každých x mikrosekund provede výpočet podle hodnot senzorů a odešle výsledek. Druhý je event oriented execution. Zde dochází k vyvolání řídícího cyklu jako důsledek nějaké události, typicky se jedná od příchozí zprávu z subscription, service nebo action. Frekvenci spouštění těchto uzlů pak lze odvodit od frekvence příchozích zpráv. Typicky se může jednat o uzel přijímající snímky z kamery na kterých provede výpočet a vrátí odpověď. Frekvence výpočtu je dána příchozími snímky, pokud snímky přestanou přicházet, uzel se nebude spouštět. [4]

Topic

Je základním a také nejčastěji používaným způsobem pomocí kterého spolu ROS2 uzly komunikují. Topic si lze představit jako analogii hardwarové sběrnice. Prakticky se jedná o přesně pojmenované místo, do kterého může n uzlů posílat data (Publish) a m poslouchat

Algoritmus 1: Definice a použití Node objektu

```
1: class CustomNode(Node):
2:     def __init__(self):
3:         super().__init__('node_name')
4: def main(args):
5:     rclpy.init(args=args)
6:     node = CustomNode()
7:     rclpy.spin(node)
8:     node.destroy_node()
9:     rclpy.shutdown()
```

co bylo posláno (Subscribe). Zprávy posílané do topicu mají přesný formát určený pomocí Interface a jsou posílány asynchronně. Typickým příkladem použití může být topic, do nějž posílá data uzel ovládající kameru a několik dalších uzlů které tyto data potřebují jej mohou číst. [4]

Algoritmus 2: Subsrciber Node

```
1: self.create_subscription(Interface, "topic_name",
   self.callback_function, queue_size)
2: def callback_function(self, msg):
3:     value = msg.item
```

Tento kód ukazuje, jak se může uzel přihlásit v odebírání zpráv z topicku. Nejprve je potřeba (typicky v konstruktoru třídy) zavolat zděděnou metodu sloužící k inicializaci nějaké ROS2 funkcionality. V tomto případě se jedná o `create_subscription`. Jako parametry potřebuje jméno, interface, callback funkci a délku fronty. Inteface definuje formát zpráv a délka fronty je použita v případě, že uzel nezvládá přijímat zprávy dostatečně rychle. Callback funkce je pak zavolána pokaždé když do topicku přijde nová zpráva. Druhým parametrem callback funkce je předán objekt, který ve svých atributech obsahuje hodnoty dané zprávy.

Algoritmus 3: Publisher Node

```
1: self.publisher = self.create_publisher(Interface, "topic_name",
   queue_size)
2: output = Interface()
3: output.item = some_value
4: self.publisher.publish(output)
```

Odesílání zpráv do topicu demonstruje tento kód. Jeho struktura je podobná předchozímu příkladu. Odeslání zprávy demonstrují řádky 2–4. Nejprve dojde k inicializaci objektu interface, stejného datového typu jako ten, který používá topic. Tento objekt je následně naplněn daty a pomocí metody `publish()` předem vytvořeného publisheru odeslán.

Service

Sevice funguje stejně jako klient – server komunikace známá z počítačových sítí . Jedná se tedy o synchronní komunikaci kde jedna node poskytuje nějakou službu a ostatní si na ni mohou poslat požadavek. Od service se typicky předpokládá okamžitá odpověď aby nedošlo k narušení (control cycle) volajícího uzlu. [4]

Algoritmus 4: Service server

```
1: self.srv = self.create_service(Interface, "service_name",
   self.callback_function)
2: def callback_function(self, request, response):
3:     value = request.item
4:     response.item = some_value
5:     return response
```

Změnou oproti předchozím příkladům je přidání nového parametru do callback funkce. Tato funkce má dva důležité parametry, request a response. Request obsahuje konkrétní hodnoty požadavku na server a response je potřeba naplnit výsledky a vrátit z funkce.

Algoritmus 5: Service client

```
1: self.cli = self.create_client(Interface, "service_name")
2: while not self.cli.wait_for_service(timeout_sec=1.0):
3:     pass
4: def send_request(self):
5:     self.req = Interface.Request()
6:     self.req.item = some_value
7:     self.future = self.cli.call_async(self.req)
8:     rclpy.spin_until_future_complete(self, self.future)
9:     response = self.future.result()
10:    value = response.item
```

Service client narozdíl od topic subscriberu závisí na tom, aby existoval server, který je schopný odpovídat na jeho požadavky. Tato podmínka vyplývá z faktu, že service server by měl odpovídat na dotazy téměř okamžitě a service klient tedy předpokládá, že vždy dostane odpověď. Pokud by neexistoval server, klient by při požadavku skončil v nekonečném čekání. Proto je hned v konstruktoru implementována kontrola, které nedovolí vytvoření uzlu dokud není přítomen server. Čekání na odpověď od serveru, je pak implementována pomocí funkce `spin_until_future_complete()`

Action

Jedná se o rozšířenou verzi service. Akce z pravidla vykonává déle trvající požadavek. Například provedení řídícího manévru robota, který je prováděn v reálném světě a jeho provedení není tedy krátkodobá záležitost. Akce pak na rozdíl od service dokáže v průběhu vykonávání této činnosti odesílat průběžné aktualizace o aktuálním stavu provádění zpět volající node.

Implementačně funguje akce jako dva service a jeden topic. Cílový (goal) service slouží k zaslání požadavku na server a jeho potvrzení. Výsledkový (result) pak vrací výsledek operace. V průběhu akce pak server posílá aktualizace do topicu.

Algoritmus 6: Action server

```
1: self.action_server = ActionServer(self, Interface, "action_name",
   self.callback_function)
2: def callback_function(self, goal_handle):
3:     goal_handle.request.item
       // odeslání zpětné vazby volajícímu
4:     feedback = Interface.Feedback()
5:     feedback.item = some_value
6:     goal_handle.publish_feedback(feedback)
       // úspěšné ukončení požadavku
7:     goal_handle.succeed()
8:     result = Interface.Result()
9:     result.item = some_value
10:    return result
```

Action server využívá stejné postupy jako předchozí ukázky, pouze jich kombinuje více dohromady. V rámci callback funkce může odesílat průběžně zpětnou vazbu a nakonec jako návrat z funkce předá výsledek.

Při pohledu na implementaci action klienta lze dobře vidět vnitřní implementace akcí. První funkce `send_goal()` vypadá podobně jako service client. Dojde zde k zaslání požadavku na server a následné čekání na odpověď. Součástí požadavku je předání feedback topicu a čekání na odpověď není aktivní ale pomocí callback funkce. Druhá `response_callback_function` pak zpracuje výsledek požadavku (přijmutí nebo zamítnutí) a pošle požadavek na result service, který nakonec vrátí výsledek.

Interface

Interface slouží k určení přesného formátu jednotlivých zpráv, které jsou posílány mezi uzly. ROS2 obsahuje mnoho již vytvořených a vývojáři po celém světě používaných formátů. Tento přístup podporuje znovupoužitelnost vytvořeného kódu a šetří práci. Díky tomu může být software pro ovládání konkrétního kusu hardware naimplementován pouze jednou s využitím standartního rozhraní a všichni ostatní jej pak mohou využít ve svých systémech. Pokud však standartní interface nevyhovuje potřebám, lze si naimplementovat vlastní. K definici konkrétního formátu slouži tři druhy souborů.

Prvním jsou `.msg` zprávy. Tento formát je využívám topicy. Skládá se ze seznamu dvojic datový typ a název (případně ještě komentář).

```
int32 angle #comment
string direction
```

Druhým je `.srv`. Slouží pro definici request/response zpráv pro komunikaci se servicem. Tento soubor obsahuje dvě části, požadavek a odpověď, každá je tvořena seznamem položek a jsou odděleny řádkem `---`.

Algoritmus 7: Action client

```
self.action_client = ActionClient(self, Interface, "action_name")

// zaslaní požadavku na server
def send_goal(self):
    goal_msg = Servo.Goal()
    goal_msg.item = some_value
    self.action_client.wait_for_server()
    self.goal_future = self.action_client.send_goal_async(goal_msg,
    self.feedback_callback_function)
    self.goal_future.add_done_callback(self.response_callback_function)


// reakce na přijmutí nebo zamítnutí požadavku
def response_callback_function(self, future):
    goal_handle = future.result()
    if not goal_handle.accepted:
        return
    self.result_future = goal_handle.get_result_async()
    self.result_future.add_done_callback(self.result_callback_function)


def feedback_callback_function(self, msg):
    feedback = msg.feedback
    value = feedback.item
def result_callback_function(self, future):
    result = future.result().result
    value = result.item
```

```
int32 a
int32 b
---
int64 sum
```

Poslední je `.action` soubor. Slouží pro komunikaci s action serverem. Definice se skládá ze tří seznamů, jeden pro požadavek, druhý pro odpověď a poslední pro stavové aktualizace.

```
float32 goal_angle
---
bool response
---
float32 current_angle
```

Parametry

ROS2 uzly lze spouštět s parametry. Typicky slouží k nastavení hodnot (předání konfiguračního souboru) za běhu programu bez potřeby zásahu do zdrojových kódů. Příkladem může uzel, sloužící k obsluze periferního zařízení a parametrem jsou mu předány čísla GPIO pinů na které je dané zařízení připojeno.

Algoritmus 8: PARAMETERS

```
  // deklarace parametru, typicky v konstruktoru
1: self.declare_parameter('parameter_name', 'default_parameter_value')

  // získání hodnoty parametru
2: param =
  self.get_parameter('parameter_name').get_parameter_value().string_value


  // nastavení hodnoty parametru
3: new_param = rclpy.parameter.Parameter(
4:     'parameter_name',
5:     rclpy.Parameter.Type.STRING,
6:     'default_parameter_value'
7: )
8: new_param_list = [new_param]
9: self.set_parameters(new_param_list)
```

Launch File

ROS2 systém se skládá z velkého množství navzájem komunikujících uzlů, a protože spouštění každého uzlu zvlášť by bylo pracné a zdlouhavé, existují launch soubory, které tuto práci usnadňují. Tyto soubory můžou být napsány v pythonu, yaml nebo xml. Čtyři hlavní úkoly které launch soubory plní jsou, spouštění uzlu, volání dalšího launch soubory, nastavení parametrů a proměnných prostředí. Prakticky se launch soubory píší minimálně na dvou úrovních. Na nižší úrovni se využívají jako součást package, kde slouží k spouštění jednotlivých uzlů. Jejich úkolem je nastavit uzel tak, aby nebylo potřeba modifikovat zdrojový kód. Typicky tak nastavuje správný namespace, předávají parametry, konfigurační soubory a případně dochází k přemapování jména topicu. Launch soubory vyšších úrovní pak slouží ke spuštění několika uzlů zároveň a využívají k tomu launch soubory nacházející se v packages.

Knihovny

Algoritmus 9: Launch file

```
def generate_launch_description():
    return LaunchDescription([
        // spuštění konkrétního uzlu
        Node(
            package='package_name',
            executable='node_name',
            namespace='namespace_name',
            parameters=[{
                'param_name' : param_value,
            }]
        ),
        Node(
            package='package_name',
            executable='node_name',
            remappings=[
                ('topic_name', 'different_topic_name'),
            ]
        ),
        // zavolání dalšího launch souboru
        IncludeLaunchDescription(
            PythonLaunchDescriptionSource([
                PathJoinSubstitution([
                FindPackageShare('package_name'),
                    'launch',
                    'node_name.py'
                ])
            ]),
        )
    ])
```

Kapitola 4

Implementace

Kapitola 5

Závěr

Literatura

[1] Braunl, T. *Embedded Robotics: From Mobile Robots to Autonomous Vehicles with Raspberry Pi and Arduino*. Fourth edition. Springer, 2022. ISBN 9811608032.

[2] *I2C-bus specification and user manual* [online]. um10204. NXP Semiconductors [cit. 2023-11-12]. Dostupné z: https://www.nxp.com/docs/en/user-guide/UM10204.pdf.

[3] NXP Semiconductors. *PCA9685: 16-channel, 12-bit PWM Fm+ I2C-bus LED controller* [online]. 2009 [cit. 2023-11-12]. Dostupné z: https://www.nxp.com/docs/en/data-sheet/PCA9685.pdf.

[4] Rico, F. M. *A concice introduction to robot programming with ROS2*. First edition. CRC Press, 2023. ISBN 978-1-032-26465-3.

[5] Salhuana, M. *Sensor I2C Setup and FAQ* [online]. an4481. Freescale Semiconductor, Inc, červenec 2012 [cit. 2023-11-12]. Dostupné z: https://www.nxp.com/docs/en/application-note/AN4481.pdf.

[6] STMicroelectronics. *L298: Dual Full-Bridge Driver*. 2000 [cit. 2023-11-12].

Příloha A

prilohy