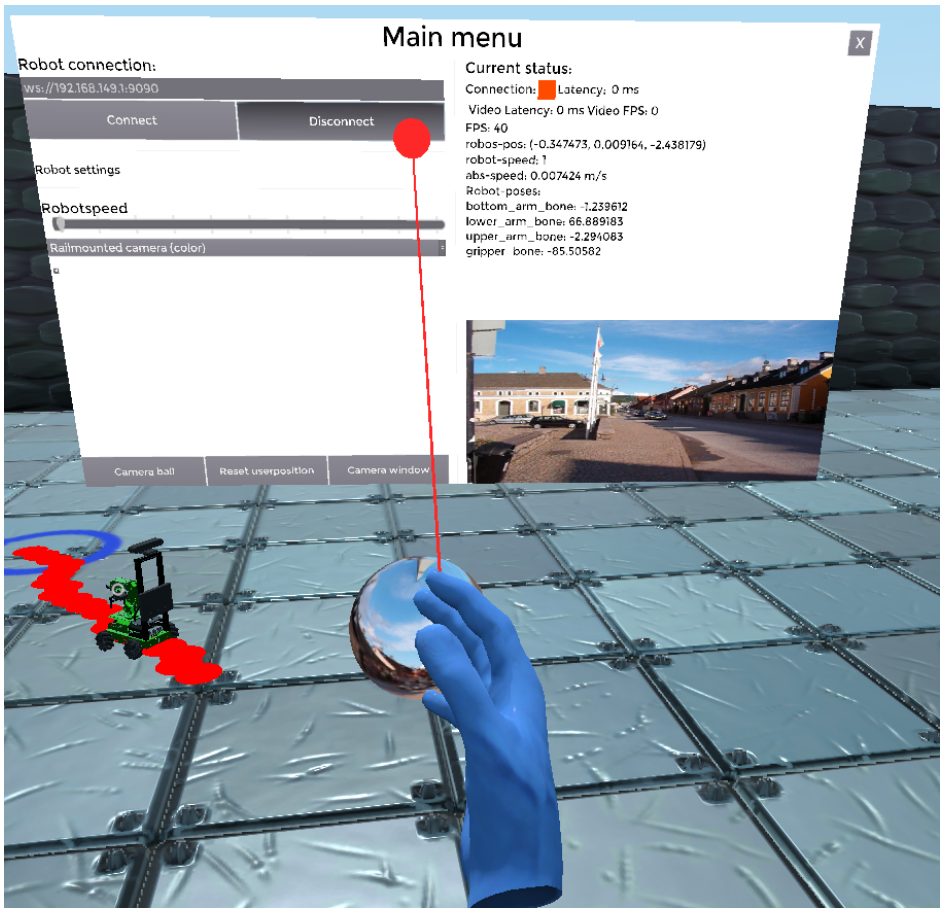


Maskinstyrning via VR

Utveckling av VR-applikation med OpenXR och Godot

Examensarbete inom Data- och Informationsteknik

Felix Holmström
Alen Mukaca

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2023
www.chalmers.se

EXAMENSARBETE 2023

Maskinstyrning genom VR

Utveckling av VR-applikation med OpenXR och Godot

Felix Holmström
Alen Mukaca

GÖTEBORGS
UNIVERSITET

CHALMERS

Institutionen för Data- och Informationsteknik
CHALMERS TEKNISKA HÖGSKOLA
GÖTEBORGS UNIVERSITET
Göteborg, Sverige 2023

Maskinstyrning genom VR
utveckling av VR-applikation med OpenXR och Godot
Felix Holmström
Alen Mukaca

Uppdragsgivare: Combitech
Handledare på Chalmers Tekniska Högskola: Morten Fjeld
Teknisk handledare på Combitech: Henrik Hellström och David Brandberg
Examinator: Jonas Duregård, CS, CSE Examensarbete 2023

Institutionen för Data- och Informationsteknik
Chalmers tekniska högskola
Göteborgs universitet
SE-412 96 Göteborg
Telephone +46 31 772 1000

Typeset in LaTeX
Printed by Chalmers Reproservice
Göteborg, Sverige 2023

Machine remote control trough VR
Felix Holmström
Alen Mukaca
Institution of Computer and Information technology
Chalmers University of Technology
Gothenburg University

Abstract

As the number of robots and drones used in society increases, new ways to manouver and control said units needs to be investigated. The evergrowing field of 'Extended reality' holds great potential on how to manouver and control robots in a more intuitive manner. The aim of this study was to create a VR application through which a user in an interactive manner could control and recieve data from a HiWonder JetAuto SLAM robot.

Development was done in the GodotEngine utilizing the OpenXR plugin to interact with different Headmounted Displays. The final application allows a user to control the robot by interacting with a virtual representation inside VR. Connection to the robot was achieved through a preconfigured selfhosted Wifi and instructions were sent utilizing a websocket in the format of JSON, in accordance to the ROSBridge API specification. Proper pathfinding was achieved through the implementation of an A* algorithm. The equipped robot arm could be controlled directly by the user grabbing and moving the arm inside VR. The user can also receive a stream of pictures from the mounted cameras yielding a videofeed at a high update frequency, either projected on a wall or on the inside of an orb. Although there might be more intuitive ways to recieve and view videodata through VR the resulting application serves as a good starting point to expand further upon. Further investigation and integration of data sources from the robot could further improve the utility of the application.

Keywords: VR, Godot, ROS, Robot, OpenXR, Human Robot Interaction, Websocket.

Maskinstyrning genom VR
Felix Holmström
Alen Mukaca
Institutionen för Data- och Informationsteknik
Chalmers tekniska högskola
Göteborgs universitet

Sammanfattning

Allteftersom robotar och drönare är mer förekommande inslag i samhället behövs mer nytänkade sätt att styra dessa enheter. 'Extended reality' erbjuder många nya möjligheter för en användare att på ett mer intuitivt vis styra en robot. Syftet och målet med denna studie var att uveckla en VR-applikation genom vilken användare bereds möjlighet att styra och ta emot data från en HiWonder JetAuto SLAM robot. Applikationen byggdes i spelmotorn Godot med ramverket OpenXR för att sammankoppla applikationen med olika VR-headset. För uppkoppling till robot nyttjades en av roboten förkonfigurerad Wifi-hotspot genom vilken instruktioner och data utbyttes mellan roboten och applikationen i formen av JSON i enlighet med ROSBridge API-specifikation.

Applikationen möjliggör för en användare att ta kontroll över en robot och skicka instruktioner för förflyttning av hela roboten genom att interagera med roboten i en VR-miljö. Instruktioner för förflyttning genererades med hjälp av en A*-algoritm. Då den valda roboten var utrustad med en robotarm gavs användaren möjlighet att flytta på denna genom att interagera och dra i den virtuella representationen av robotarmen. Videodata från robotens olika kameralägen återfås inuti VR i formen av en virtuell skärm eller på insidan av en kamerasfär i realtid. Även om det kan tänkas finnas fler och mer intuitiva sätt att ta emot och se video inuti VR utgör applikationen ett bra utgångsläge för vidare efterforskning och utveckling. Fortsatt integration av övriga sensorer och datakällor hade kunnat nyttjas för att förbättra nyttovärdet av applikationen.

Nyckelord: VR, Godot, ROS, Robot, OpenXR, Människa Robot Interaktion, Websocket.

Förord

Vi skulle vilja tacka Ingvar Andersson på Combitech för att ha gett oss möjligheten att få arbeta med detta projekt samt våra två handledare vid Combitech Henrik Hellström och David Brandberg för all feedback och hjälp längs med vägen.

Vi skulle även vilja tacka vår handledare vid Chalmers Morten Fjeld för flertalet förslagsrika och konstruktiva möten under projektets gång.

Felix Holmström,
Alen Mukaca,
Göteborg, Maj 2023

Förkortningslista

A*	A-star
AR	Augumented Reality
FPS	Frames Per Second
HMD	Headmounted display
MVP	Minimal viable product
PUB	Publish
ROS	Robot Operating System
SUB	Subscribe
VR	Virtual Reality
XR	Extended Reality
WS	Websocket

Nomenklatur

Nedan presenteras den nomenklatur och terminologi som används för olika ramverk, gränssnitt och andra ej vedertagna variabler som förekommer i denna rapport.

3D-modellering

Mesh	En 3D-modell med färdiga punkter och ytor men inkluderar inte en textur.
Textur	En bild som appliceras på ytan av en mesh.
Transform	En 3x4-matris innehållande värden för vektorerna $\hat{\imath}$, $\hat{\jmath}$ och $\hat{k}$ samt en vektor för ursprungspunkt för att modifiera och förflytta ett objekt i 3D.
Skelett	Definierar ett kollektiv av "ben" som är kopplade med varandra med förälder och barn relation som förhindrar dem från att separeras. Bitar av 3D-modeller kan associeras med ett eller flera ben för att följa dess rörelser. Skelett används främst för animering av 3D-modeller.
Riggning	Att sammankoppla ett konstruerat 3D-skelett med en 3D-modell för att representera enskilda delar och hur de sitter ihop.
Inverterad Kinematik	Det är processen för att beräkna alla leders rotation i skelett från en mål position där skelettets ände ska försöka nå.
Keyframe Animation	En metod att animera ett skelett eller ett 3D objekt genom definiera specifka poser som rotation och plats på vissa tid punkter. Sedan interpolerar modellens Transform mellan de punkterna.

Godot

Nod	Basen för alla olika funktioner i Godot vilket motsvarar ett objekt i andra högnivåspråk.

Process	Huvudloopen i varje skript som körs så ofta som möjligt vilket brukar vara varje genererad bildruta. Uppdaterar den grafiska representationen.
Physics process	En for-loop som hanterar beräkning av alla fysikrelaterade funktioner oberoende av process.
Move and slide	En funktion för att förflytta ett objekt i 3D med hjälp av physics process.
Delta	En räknare som håller värdet på mängden tid som passerats i sekunder sedan senaste uppdatering av 'process' respektive 'physics process'.

Protokoll och Verktyg

HTTP	Ett protokoll som används i webben där en klient kan skicka ett meddelande i form av en begäran och sedan få ett svar. Protokollet kan användas för att ladda upp och ner data från en server. Dessutom är alla begäran oberoende av tidigare begäran och svar.
Websocket	Ett protokoll för att kunna sporadiskt skicka meddelande fram och tillbaka. Tillskillnad från HTTP bibehålls TCP kopplingen längre än ett meddelande vilket tillåter för sporadisk utskick och mottagning av data. Dessutom etableras Websocket-kopplingen via en HTTP-uppgraderingsrubrik vilket tillåter identifiering och användning av en HTTP-proxy.
JSON	Ett data-utbytesformat hämtat från JavaScript som numera används som ett språkagnostiskt format att spara och överföra data.
APK	Ett packeterinsformat för applikationer till mobiloperativsystemet Android vilket brukar användas för installation. Formatet är numera ersatt av Android App Bundle (AAB).

VR-kontrollknappar

För att se en visuell representation av knapparna hänvisas till Appendix A.

Greppknapp	En knapp på sidan av kontrollen som brukar användas för att greppa objekt i VR-världen.
Trigger	En knapp på kontrollernas baksida som brukar användas för att välja eller klicka inom ett UI eller för att aktivera en funktion på ett objekt som hålls.

Joystick	Majoriteten av alla VR-kontroller har en joystick, en analog spak som används till olika saker såsom förflyttning. Vive-kontrollen har en pekplatta istället vilken fyller samma som funktion som en joystick.

Innehåll

Figurer

1
Inledning

I följande kapitel belyses bakgrunden till projektet såväl som syfte och mål som implementationen syftar uppnå. För att minimera onödig tidspillan på ej kritiska komponenter sattes avgränsningar vilka även beskrivs nedan.

1.1 Bakgrund

Robotar och drönare är en allt vanligare förekomst i det moderna samhället. Att undersöka hur dessa robotar kan användas för att minimera den mänskliga faktorn från kritiska situationer kan vara mänskligheten behjälpligt[1]. Exempel på detta skulle vara inom gruvindustrin där möjligheten att använda sig av robotik för att kunna inspektera skador som kan uppstå i samband med gruvras utforskas [2]. Även i samband med thoraxkirurgi har assistans av robotar visat potential att snabba på den postoperativa återhämtningen som en följd av snabbare och mer precisa ingrepp [3]. Med ökad tillgänglighet av smart teknologi som kan kopplas upp mot lokala nätverk och även samspela med annan elektronik ökar efterfrågan på fler möjligheter att kunna styra och interagera med hårdvara genom styrdon bortom de som historiskt funnits tillhanda såsom tangentbord och en mus [4].

Virtuell verklighet (VR) hade sitt intåg på marknaden med flertalet fortsatt aktiva företag såsom Meta, Valve och Microsoft vars olika produkter ger möjligheten för användaren att kunna interagera och röra sig i en simulerad virtuell miljö där alltifrån huvud- och handrörelser registreras [5]. Detta ger upphov till frågor såsom hur man kan uppnå låg latens i kommunikationen mellan VR och den styrda maskinen samt hur användarupplevelsen skall se ut vid användning.

1.2 Syfte

Syfte med projektet var att utveckla en applikation som nyttjar möjligheterna som VR för med sig avseende styrning av extern hårdvara, såsom en robot. Då många moderna VR-system använder sig av positioneringssystem av huvud och händer ger detta möjligheten att mer intuitivt och interaktivt flytta och interagera med objekt i den virtuella världen genom att spegla användarens rörelser. Projektet kan även vara en bra demo för att visa upp konceptet.

1.3 Mål

Skapa ett grafiskt gränssnitt i VR genom vilket en operatör kan styra ett virtuellt gränssnitt för en robot och därigenom en fysisk robot av modellen HiWonders JetAuto Slam.

1.4 Avgränsningar

För att simulera en komplett miljö i VR behöver fysik och grafisk representation av olika objekt hanteras. Detta är inget som kommer skapas inom ramen för detta arbete utan den etablerade grafikmotorn Godot kommer användas. Numera finns flertalet företag som erbjuder VR-headsetlösningar men i detta projekt kommer en HTC Vive Pro med medföljande två handkontroller användas. Då OpenXR, ett multikompatibelt gränssnitt för kommunikation mellan VR-headset och applikationen, används leder detta till att olika modeller av VR-headset kommer kunna köra projektet men utveckling av applikationen fokuserar på och testas med en HTC Vive Pro och med en Meta Quest 2.

För att styra roboten så kommer gränssnittet RoboticOS (ROS) att användas för att simulera och styra hårdvaran. Även om ROS i egenskap av gränssnitt kan användas för att interagera olika sorters robotar kommer slutprodukten inte innebära ett generiskt stöd för robotar utan fokusera att styra den i projektet valda roboten. Utöver ovan nämnda kommer nedanstående punkter inte heller beröras:

- Human-Robot-Interaction användarupplevelse
- Tolkning av videodata i syfte för objektigenkänning
- Dynamisk generering av VR-miljö baserat på data från robotens Lidar och djupkamera

2
Metod

Projektets planerade tillvägagångssätt beskrivs i detta kapitel.

2.1 Planeringsfas

Tid motsvarande 20 timmar per vecka planerades in mellan datumen 17/1-2023 och 3/6-2023. För kravställning rörande applikationens samt interaktionen mellan roboten definierades en lista med kriterier för en MVP.

2.2 Arbetssätt och arbetsfördelning

Majoritet av projektets arbete kommer ta plats i en visuell editor given av spelmotorn för att modifiera utseende och funktioner i 3D omgivningen. Mer specifika beteenden för 3D objekt använder sig av skript som skrivs i en kodredigerare. Vid framtagande av nya funktioner i VR applikationen användes ett externt bibliotek som möjliggjorde att styra rörelser med hjälp av tangentbord och mus, varvid tillgång till VR-utrustningen ej var ett explicit krav för att testa ny kod.

Inom ramen för projektet applicerades ett agilt arbetssätt i formen av scrum med tvåveckors-sprints. Uppgifter och önskade funktioner definierades för att delas upp i mindre uppgifter vilka sedan slutfördes av en eller båda projektmedlemmarna. Uppgifter och deluppgifter bokfördes med hjälp av hemsidan Trello och dess Kanban-kortsystem. Stadig och kontinuerlig utveckling inom projektet säkerställdes genom dagliga såväl som veckovisa avstämningar inom projektgruppen. Varje sprint avslutades med ett kortare möte med produktägare under vilka projektets status, vilka framsteg som skett samt vilka uppgifter som låg till grund för kommande sprint. Samt ritades ett par diagram för att visuellt representera projektets preliminära struktur vilket är länkad med kanban brädan som togs fram under samma tid.

2.3 Informationssökning

Efterforskning och litteratursökning skedde nyttjandes Google Scholar samt Chalmersbibliotekets egna sökfunktion. Robotens mjukvara undersöktes för att hitta möjliga sedan tidigare inbyggda funktioner och gränssnitt För samtliga programvaror, gränssnitt och ramverk kommer tillhörande dokumentation på respektive hemsida eller annan medföljande dokumentation användas. I de fall när programvara saknade

komplett dokumentation kommer flera tillvägagångsätt nyttjas för att komplettera denna såsom "paketsniffning" med hjälp av Wireshark, meddelande-infångning, emulering och disassemblering av robotens medföljande mobilapplikation med hjälp av vertyget APK-tool.

En översiktlig genomgång av tillgängliga spelmotorer kommer att genomföras med gallringskriterier såsom OpenXR-stöd samt tillgänglighet för nybörjare där Godot framstår som en lovande kandidat. Vidare undersökning kommer att utgå ifrån hur man implementerar projektets preliminära design i spelmotorn. Där ytligare undersökning rörande ROS och dess medföljande paket kan visa hur man bäst kommunicerar med robotar som implementerar ROS.

3
Teknisk bakgrund

För utformning av en applikation där en robot skall kunna styras genom Virtuell verklighet (VR) nyttjades flertalet program, ramverk och gränsnitt vilka presenteras nedan.

3.1 XR, AR och VR

Extended reality (XR) är den gemensamma paraplyterm som används för att gemensam beröra allt inom ramen för Augumented reality (AR) såväl som VR. AR innebär att användaren presenteras ett datorgenererat lager ovanpå den verkliga omgivningen. Ett exempel på detta är hur trafikinformation kan presenteras för en förare projicerat som ett lager på vindrutan för att öka förarens möjlighet till trafikrelaterad information utan att behöva titta ner på instrumentbrädan eller annan extern enhet [6]. AR har även visat potential i att öka säkerheten för industriarbetare [7].

VR går ut på att användaren presenteras visuellt en helt datorgenererad virtuell miljö som användaren kan interagera med. Ett VR-kit består av en huvudmonterad display (HMD) genom vilken användaren ser den virtuella världen samt handkontroller för att interagera med menyer och objekt. Vidare externa tillbehör kan bestå av hörlurar, en kontroll per hand samt väggmonterade sensorer för att detektera den relativa positionen av användarens HMD i det verkliga rummet [8]. En skillnad mellan AR och VR är miljön inuti VR inte direkt kan påverkas av den fysiska miljön.

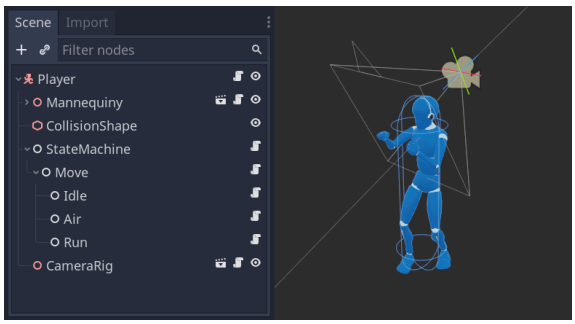
I en AR-miljö kan objekt ges en virtuell representation i programmet vilket gör att det finns en koppling mellan objektet i AR och i verkligheten, varför eventuell förflyttning av det fysiska objektet kan då ge en liknande förflyttning av det virtuella objektet. AR kan även användas för att placera mer och tydligare eller tydligare information till slutanvändaren genom användning av 'Quick Response codes' [9].

3.2 Godot

Godot är en spelmotor som utvecklas som ett projekt med öppen källkod under Software Freedom Conservancy [10]. Spelmotorn kan användas för att utöver spel även utveckla konventionella program för att köra dessa på såväl Windows, macOS,

Linux, Android, iOS och web. Programmeringsspråk som stöds i Godot är C#, C++, VisualScript, de egenutvecklade skriptspråken GDScript och mer via GDNative API:et.

Den minsta byggstenen i ett Godot-projekt är 'noden' där varje nod representerar en funktion som t.ex. en videouppspelare, en menyknapp med bakomliggande logik samt även fysiska objekt i spelvärlden såsom en vägg eller en ljuskälla. För att jämföra med spelmotorn Unity så motsvarar en nod till en modul och ett spelobjekt. En uppsamling av interagerande noder som gemensamt bidrar till en komplett funktion brukar sampaketeras till en så kallad scen [11]. Scener kan sedan på ett



Figur 3.1: Exempel på ett nodträ hos en scen från [11]. CC-BY 3.0.

modulärt sätt importeras och återanvändas i andra projekt eller delar av projekt som en nod. Sammantaget ger detta att projekt i Godot består av ett träd av noder där vissa ges en visuell representation i spelet medans andra existerar som bärare för bakomliggande funktioner, beräkningar och annan logik (se figur 3.1).

Godot som motor använder sig av två huvudsakliga trådar per nod, process och physics process, för att hålla koll på förfluten tid för programmet såväl som alla fysikberäkningar. Vissa funktioner och metoder kallas med återkommande intervall som en del av process däremot nyttjar de funktioner som berör förflyttning sig av physics process då denna kör i en högre frekvens. Vid implementering av mer beräkningskrävande funktioner samt nätverkskommunikation där eventuell risk för fördröjning är oönskad kommer nya dedikerade trådar att skapas.

3.3 OpenXR

OpenXR är ett plattformsoberoende AR/VR-bibliotek vilket möjliggör kommunikation mellan mjukvara och olika HMD. Där en applikation som integrerar biblioteket har automatisk stöd för flera olika VR-kit på samma maskin. OpenXR har en unik position då den tidigt anammades av Microsoft och Oculus, numera Meta, varför samtliga större företag kring AR/VR låtit utveckla egna drivrutiner som möjliggör sömlöst användande av OpenXR [8]. OpenXR implementeras även av andra spelmotorer såsom Unreal Engine och Godot samt har funnit stöd hos andra stora företag inom underhållning, hårdvara och telefoni [8].

3.4 Insomnia

Insomnia är ett open-source program som syftar möjliggöra för en användare att skicka data via en websocket till en annan enhet och läsa eventuella svar. Resulterande svar kan sedan användas för att testa och debugga kommandon samt läsa underliggande svarskoder [12]. Då bl.a. robotens kameror nås via HTTP-server kan användbar data insamlas och testas med hjälp av Insomnia. Manuella kommandon kan även skickas till roboten för att testa hur roboten förflyttar sig.

3.5 ROS och ROS-bridge

Robot Operativsystem (ROS) är ett mjukvaruutvecklings-kit för att abstrahera integrationen till robothårdvara. Detta tillåter mjukvaran att vara platformsoberoende samt att simulera beteende hos utvecklad kod innan denna körs på den tilltänkta hårdvaran. ROS-bridge ger ett användargränssnitt för icke-ROSkompatibla applikationer att skicka data i formen av JSON via en websocket för att ge instruktioner till en robot som använder sig av ROS [13].

3.6 Roboten

Vald robot för detta projekt är JetAuto Pro ROS SLAM Robot (se figur 3.2) producerad och utgiven av Shenzhen Hiwonder Technology Co., Ltd. beläget i Shenzhen, Kina. Roboten är utrustad med en skärm för debuggning, en 41cm lång griparm med tillhörande 480p vidvinkelkamera, 180° täckande framåtriktad Lidar och en 1024p djupkamera monterad på ett stativ. Då roboten går på

Figur 3.2: Roboten som använts inom ramen för projektet.

batterier och har fyra hjul betyder detta att robotens räckvidd endast begränsas av batteritiden och det wifi som sammankopplar roboten och styrande enhet. Roboten styrs av en mikrodator av typen Nvidia Jetson Nano vilket liknar en Raspberry Pi men är mer utformad för AI processering genom sitt inbyggda grafikkort [14].

3.7 Blender

Blender är ett 3D-modelleringsverktyg som är populär inom animation, 3D-skulptering och modellering då det bland annat är byggt på öppen källkod. Blender kan skapa 3D modeller genom att manuellt placera, addera och subtrahera olika former för att uppnå en slutgiltig dimension på ett objekt med ett mer organiskt utseende. Blender används allt oftare inom industrin och högre utbildning för dess många 3D-modellerings- och animationsmöjligheter [15][16].

4
Genomförande

I detta kapitel beskrivs genomförandet av projektet.

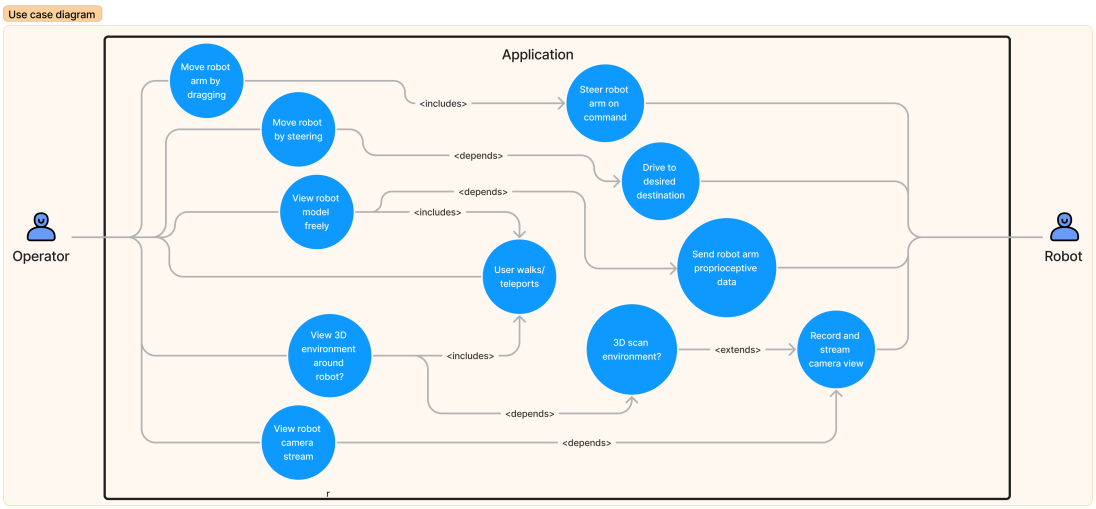
4.1 Planering, målsättning och MVP

Under första veckan gavs möjlighet för produktägaren att vidare definiera de önskvärda funktioner beskrivna i det ursprungliga projektdokumentet. MVP definierades till:

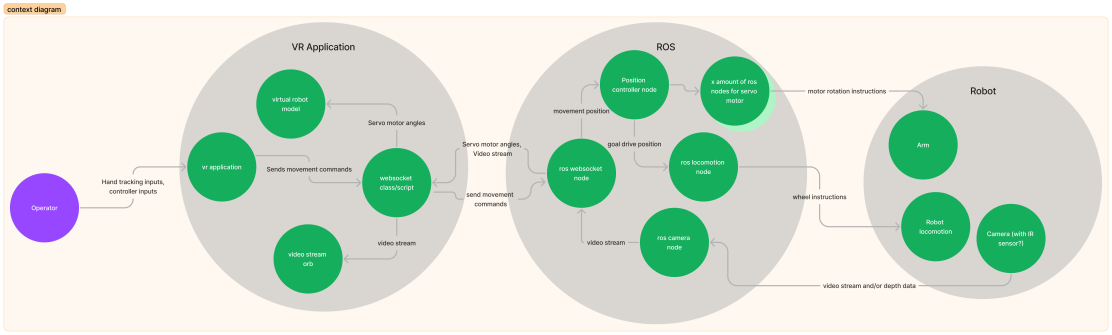
- Användaren skall ha tillgång till en VR-miljö i formen av ett rum.
- Användaren skall se en virtuell representation av en robot.
- Användaren skall kunna ge instruktioner till roboten att förflytta sig mellan två punkter.
- Roboten skall kunna ges instruktioner att ändra griparmens läge genom att interagera med den virtuella roboten.
- Användaren skall kunna ta emot videodata från roboten och få denna representerad i VR.

Litteratursökning rörande Godot och ROS genomfördes varefter en planeringsyta av kanban-modell skapades och tillhörande arbetskort med olika epiks och användarhistorier togs fram. Förslag på robot togs fram och möjliga sätt att nyttja videoströmmar samt annan sensordata undersöktes. I början av februari togs ett det fram ett par diagram för att ge en översikt för hur projektet struktur och interaktioner mellan olika aktörer kring projektet. Först skapades det en Use-case diagram som visar hur olika aktörer interagerar med VR applikation vilket i detta fall är användaren och roboten (se figur 4.1). Då varje element känt som en ”use-case” är en interaktion användaren eller roboten har med applikationen samt hur den interaktionen relaterar till andra interaktioner. Den andra diagrammet som skapades är ett Container diagram som fungerar som ett Kontext diagram som visar hur data använts och överförs mellan aktörer eller noder. Medan skillnaden här är att aktörerna är expanderade där de visar sina interna noder och dess interaktioner (se figur 4.2). Aktörerna i diagrammet är roboten (hårdvara), ROS (robotens mjukvara), VR applikationen och användaren. Båda diagrammen utgår ifrån MVP:en med undantag för funktionen för 3D skanning

som ansågs vara en extra funktion om det skulle finnas tid över.



Figur 4.1: Use-case diagram som beskriver aktörernas interaktioner med projektets preliminära struktur (Större version finns i Appendix B).



Figur 4.2: Container diagram som beskiver data utbytet över programmets preliminära struktur (Större version finns i Appendix C).

4.2 VR-miljö

En scen med fem noder som sammansatt gav ett kvadratiskt golv med en diameter på 20 meter samt tillhörande fyra tre meter höga väggar skapades. Denna scen kom att agera som huvudscen och alla andra objekt inklusive användaren själv var enskilda scener importerades som noder till huvudscenen. Nya scener utvecklades för varje objekt i VR-applikationen. Gränssnitt och grundläggande logik för kontrollmenyn, videoströmningsskärmen, kamerasfären och kommunikations-gränssnittet för roboten togs fram. De olika kontrollobjekt, för olika funktioner och hantering av data, utvecklades separat och importerades därefter till VR omgivningen för att testas.

Under tiden roboten inte var tillgänglig så användes en video som ett exempel för strömmen under testning. När roboten levererades, skapades en

websocket-nod skapades genom vilken uppkoppling mot roboten etablerades. Genom websocket-noden kunde olika instruktionsmeddelanden skickas för att hämta information om roboten eller styra den. Under mars månad skapades logik för varje kontrollobjekts funktion i huvudscenen såsom automatisk förflyttning av menyn och kamerasfären så att dessa följde sina relevanta objekt.

4.3 Visualisering av roboten i VR

Robtens källkod innehöll STL-filer som består av flera meshes av dela olika delar i robten och URDF/XML-filer som beskrev hur de olika mesharna placerades men deras fysiska egenskaper. Dessa erhölls från leverantören varefter de importerades till Blender för att sammansättas till en komplett representation av roboten inklusive riggning av skelett för animation. Robotens rörliga delar såsom armen och gripklon riggades med virtuella ben och länkar för att rotation och övriga rörelser skulle kunna simuleras i VR.

Den färdiga robotmodellen exporterades från Blender till Godot-formatet ESCN varefter den importerades in till huvudscenen. Robotarmens skelett konfigurerades till att använda sig av Inverterad Kinematik tillsammans med skriptade begränsningar för kunna animera armen. Klons animation använder sig av Keyframe Animation eftersom klons skelett har för många paralella ben för kunna nyttja Inverterad Kinematik. För förflyttning av roboten i VR användes den inbyggda metoden “move and slide“. Detta då denna funktion utför en korrekt förflyttning av den virtuella roboten oberoende av eventuella fördröjningar i rendering av scenen. Roboten följer den linje som genererats av punkterna framtagna av A*-algoritmen tills det den nått sitt mål.

4.4 Styrning av roboten i VR

Roboten levererades med en färdig applikation genom vilken användaren kan använda sig telefon för att ge instruktioner till roboten. På grund av tidspress och problem att starta upp ROS och MoveIt på arbetsdatorerna dekompilerades den medföljande WonderAi Android applikationen för att extrahera information kringom vilka kommandoanrop som roboten lystrade till. Vidare utforskades robotens medföljande API och bakomliggande styrlogik i den medföljande mobilapplikationen genom att roboten kopplades upp mot en dator körandes Wireshark och mobilappen för att se de utformade meddelandena som skickades till roboten.

Senare upptäcktes det att roboten använder sig av ROS Bridge, ett väldefinierat API. Detta gav insikt till hur robotens medföljande källkod kommunicerade via dess Websocket API varefter det gick att identifiera och testa nya kommandon via verktyget Insomnia. Då roboten vid uppstart startar ett eget wifinätverk och en websocketserver under en fördefinierad IP-adress kopplades arbetsdatorn upp mot detta nätverk för att interagera och utforska robotens funktioner. En omfattande dokumentation i form av ett AsyncAPI-dokument över robotens medföljande

implementation av ROS och ROS Bridge skapades för att ligga till grund för instruktioner att skicka till roboten för att genomföra önskade förflyttningar av såväl själva roboten som robotens arm.

En A*-algoritm nyttjades för att spara ner samtliga punkter som roboten behövde passera för att nå en annan punkt i rummet. Föreslagen rutt-data bearbetades av programmet och rutten visualiserades med hjälp av röda cirklar. När Websocket-noden utvecklats kom den att implementeras på flera områden i Godot-projektet såsom att skicka hastighetskommandon när A* algoritmen kollar efter nästa punkt i rutten. Vidare skickades även rotationskommandon till roboten för förflyttning av armen när användaren släpper robotarmen med hjälp av websocket-noden.

När användaren greppar armens topp då rörs den med handen med toppen riktad mot handen. Initialt skedde videoöverföring med hjälp av videospelar-noden i Godot men på grund av prestandaproblem i form av hög latens på videoöverföring byttes denna ut mot en mesh med tillhörande textur. När applikationen kördes fanns ett skript som kontinuerligt uppdaterade denna textur med en ny bild från kopplad kamera.

5

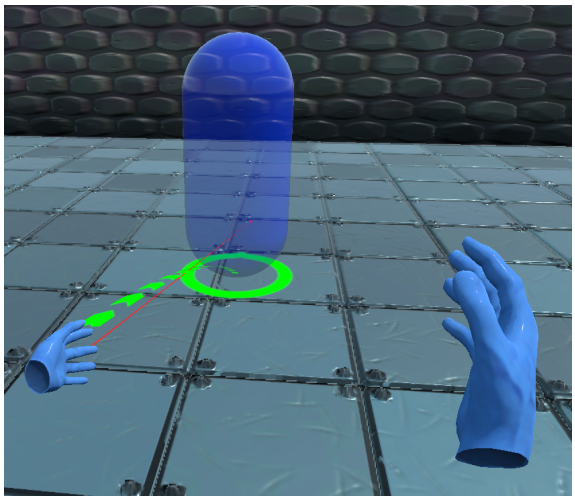
Resultat

I detta projekt har en VR-applikation med tillhörande styrlogik för en extern robot skapats. Applikationen kan kompileras till att köra på Windows, Mac-OS, Linux och Android. Applikationen körs i över 90 fps på Windows men har endast kunnat uppnå 35 fps på Android. Projektet är testat för att vara fullt fuktionellt i Windows medan Android versionen är funktionell med undantaget att själva uppkopplingen till roboten inte fungerar.

5.1 VR-applikation

I VR har användare möjligheten att förflytta sig endera genom inbyggd teleportering eller faktisk förflyttning i det egna rummet. Det skapade kvadratiska rummet har en yta på $400m^2$ där varje ruta på golvet är en kvadratmeter.

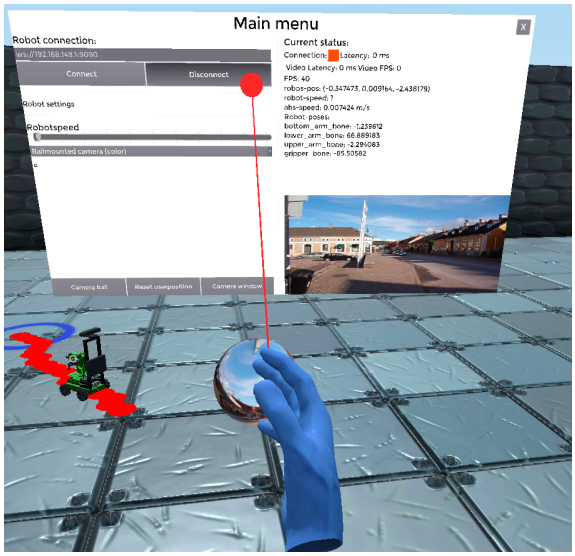
För vägvalsproblematik för uppkopplad robot har ett kvadratiskt rutnät med sidor på 0,1u, vilket motsvarar 0,1m, skapats och placerats ovanpå golvet. Genom olika knapptryck kan användaren teleportera (se figur 5.1), peka på objekt, ge instruktioner för att flytta på roboten samt skapa en meny med inställningar och olika knappar.



Figur 5.1: Användaren väljer plats för att teleportera.

Tillgängliga knappinteraktioner är:

- Greppknappen, i direkt kontakt med hand eller genom laserpekaren, greppar tag i kamerasfär eller robotarm.
- Applikationsmenyknappen visar eller döljer menyn.
- Triggerknappen kan användas för att:
  - Peka och klicka på menyelement i menyn.
  - Peka och klicka på golvet för att flytta målmarkören vilket inleder en förflyttning av roboten.
  - När robotarmen är greppad då kan klon styras med hjälp av triggerknappen. Klon stänger sig i motsvarande grad till hur nedtryckt triggerknappen är och stannar i valt läge när robotarmen inte längre är greppad.

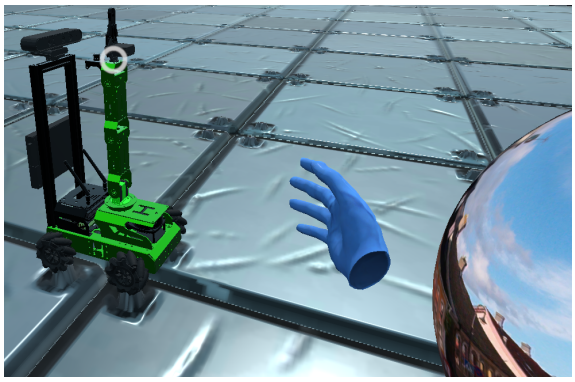


Figur 5.2: Användaren klickar på en knapp i menyn. Exempel stadsbild från [17]. CC-BY.

I menyn ges användaren möjlighet att koppla upp sig mot en robot och visa status för om kontakt etablerats med roboten (se figur 5.2). De olika knapparna, skjutreglagen och fälten i menyn gör det möjligt för användaren att:

- Ange en IP-adress att ansluta till
- Koppla upp mot angiven IP-adress
- Frånkoppla sig från roboten
- Reglage för att ange hastigheten på robotens rörelser
- Välja kamerainput att visa
- Se en kameraström från vald kamera
- Gå in i debugläge för att se mer information:
  - Uppkopplingstatus
  - FPS
  - Latens på bildström
  - Värdet på hastighetsreglaget

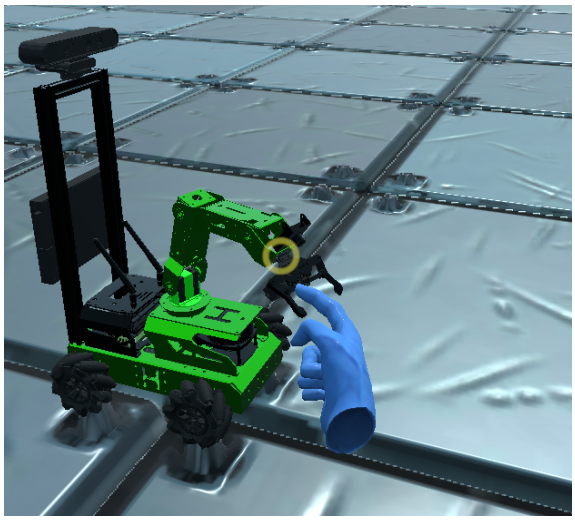
  - Aktuell hastighet på roboten i VR
  - Vinklar på samtliga av robotarmens motorer för armen
- Frammana en kamerasfär till scenytan
- Nollställa användarens virtuella position i rummet
- Ta bort en videoström från en vägg i rummet



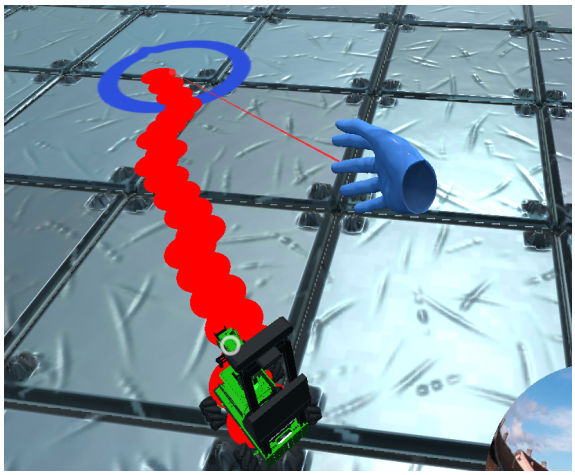
Figur 5.3: Virtuella roboten i startposition.

5.2 Interaktion mellan robot och VR

I menyn kan användaren skriva in IP/värdnamn genom vilket uppkoppling mot roboten är tillgänglig, standard vid tomt fält är 192.168.149.1 vilket är robotens fasta IP när den har ett eget wifi-nätverk. När uppkoppling till roboten etablerats över Websocket kan användaren interagera med en virtuell representation av roboten in i det virtuella rummet och få eventuella positionsförändringar efterspeglade av den fysiska roboten (se figur 5.3). Genom att greppa på robotens arm kan användaren ta kontroll över den virtuella roboten arm i helhet förflytta denna till önskat läge (se figur 5.4). Robotarmen kan greppas tag i och förflyttas eller vrida sig i alla dess



Figur 5.4: Robotarmen blir dragen av handen med öppen klo.



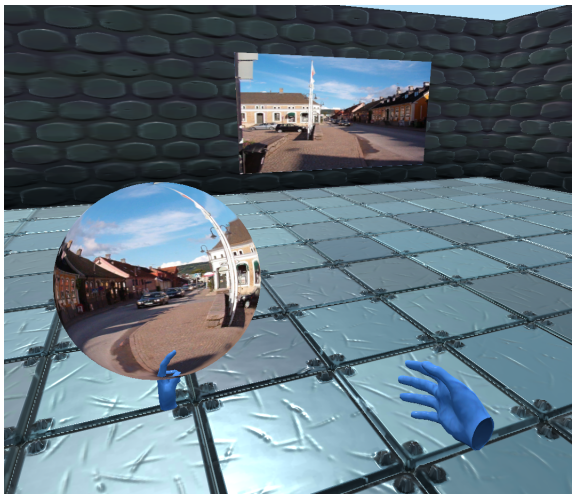
Figur 5.5: Roboten åker till den markerade punkten.

tillgängliga vinklar inuti VR-applikationen. När användaren släpper armen skickas det en instruktion till roboten att matcha samma ställning som i den virtuella roboten. Instruktion att förflytta roboten initialiseras genom att peka på marken och trycka på trigger-knappen för att placera en målmarkör med en pil för att indikera riktning på målmarkören. Användaren kan välja att hålla nere knappen för målmarkören och rikta pilen genom att dra sin hand i valfri riktning i förhållande till där målmarkören placerats för att peka i den valda riktningen. Efter att användaren släppt knappen roterar roboten för att motsvara riktningen på målmarkörens pil.

Applikationen beräknar och målar därefter ut en rutt längs det etablerade kvadratiska rutnätet med hjälp av en A*-algoritm för att ta sig till målmarkören. Efter att föreslagen rutt målats ut förflyttar sig roboten längs denna rutt (se figur 5.5). Hastigheten kan justeras under förflyttning genom menyns robothastighets-reglage. De tillåtna hastigheterna för roboten är på en skala 0-100 där ’100’ motsvarar ca: 0.4m/s i verkligheten och ’25’ ger 0.1m/s. Vid djupare inspektion av robotens källkod hittades det en parameter som tyder på att det finns en inbyggd vägmätningsfunktion som dock var inaktiverad vid leverans och eventuella försök att aktivera denna misslyckades.

5.2.1 Representation av sensordata från roboten i VR

När roboten är aktiv och uppkopplad mot applikationen så kan användaren frammana en kamerasfär genom menyn. I menyn finns en flervalsmeny i vilken användaren kan växla mellan de olika kamerorna för att prenumerera på bilduppdateringar. Denna kamerasfär har en textur som kontinuerligt uppdateras med komprimerade bilder från den i menyn valda kameratyp och läge. Om kamerasfären kommer i kontakt med en vägg visas bildflödet från kameraströmmen som ett fönster på väggen (se figur 5.6). Kameraströmmen överförs med en latens på 32 millisekunder och i en uppdatering på 30 FPS med upplösningen 480p. Ingen nämnvärd förlust av bilddata i form av artefakter noterades under körning.



Figur 5.6: Kamerasfären och tavlan med en kameraström. Exempel stadsbild från [17]. CC-BY.

5.3 Dokumentation av Robotens gränssnitt

När robotens gränssnitt undersöktes skrevs det en AsyncAPI dokument som dokumenterar möjliga interaktioner med robotens Websocket API (se Appendix D). Dokumentationen innefattar:

- Rörelsekommadon som tar emot hastigheter.
- Rotationskommando för varje individuell servomotor i robotarmen som tar emot en vinkel och tid.
- Prenumeration på servomotorernas aktuella läge.
- Prenumeration på kamerornas bildström i RGB, IR och djup.
  - Information om kamerorna, såsom upplösning.
  - Möjlighet att ändra kamerainställningar.
  - En HTTP-variant för att kameradata via en Multi-part-ström.
- Prenumeration på data från lidar-skannern.
- Inbyggda API:er inom ROS som detaljeras i paketet ROS-API vilket tillåter att få information av alla möjliga ändpunkter och tjänster som är tillgängliga att kallas via Websocketkopplingen.
- Vilka IP adresser roboten är tillgänglig under när den har en WIFI Hotspot aktiv.

Dock blev inte alla möjliga meddelanden dokumenterade men robotens källkod visade spår av flera andra funktioner såsom en vägmätnings funktion och en kartläggnings funktion som inte är aktiverade för tillfället.

6
Etik och hållbarhet

Detta kapitel förklarar projektets fördelar och nackdelar utifrån ett etik- och hållbarhetsperspektiv.

6.1 Etik

Projektet syftar till att tjäna som ett proof-of-concept i hur man kan ge operatörer ett verktyg att i förlängningen kunna nyttja VR för att styra en robot, i första hand trådlöst lokalt men även att öppna möjligheten för trådlös styrning över längre distanser. Gällande den utvecklade mjukvaran är den i nuläget, i enlighet med avgränsningarna, mycket nischad att endast fungera med Hiwonders JetAuto SLAM.

Denna robot är både begränsad i vad gäller storlek och greppstyrka varför roboten på sin höjd hade kunnat inverka på enskilds integritet och hälsa genom dess förmåga att filma och skicka film över internet. Om så en illasinnad användare skulle önska omforma applikationen för att manövrera robotar eller drönare med större förmåga skulle det krävas en sådan omfattande refaktorering av grundläggande logik och noder till den grad att det skulle kunna likställas med att skriva ett helt nytt program från grunden.

6.2 Hållbarhet

I det läge om projektet skulle vidareutvecklas till där trådlös hantering av kopplad maskin är möjlig skulle detta eventuellt kunna bidra till ett minskat behov av transporter för maskinoperatörer vilket skulle kunna ha en marginell påverkan på miljön i och med att behovet av persontransporter minskar.

I valet av programvara, ramverk och gränssnitt har projektet försökt nyttja mjukvara med öppen källkod för att bidra till ökad transparens kring projektets ingående delar och hur dessa fungerar. Roboten är även modulärt byggd vilket gör det möjligt och enkelt för slutanvändaren att byta ut och reparera enskilda delar över tid istället för att behöva kasta hela roboten bara för att en enskild komponent gått sönder.

7
Diskussion

Detta kapitel kommer reflektera över projektets resultat såsom dess utvecklingspotential och eventuella felkällor.

7.1 VR-applikationen

VR applikationen var den mest tidskrävande och största biten av projektet och utveckling skedde kontinuerligt under arbetets gång.

7.1.1 Utveckling av VR-applikationen

Storleken på rummet landade vad som motsvarar 400 kvm ett exakt mått som kan anses vara ovanligt att stöta på i verkligheten. Valet på detta mått handlade mer om utmaningen om att förstå hur man skulle kunna dynamiskt skala om rummet under körning. En indirekt begränsning som valet av storlek på rummet förde med sig var att etablerandet av rutnätet, som användes vid vägvalsproblematik genom A*, tog mycket lång tid att generera.

Ett försök att generera ett mer finmaskigt rutnät, t.ex. 0,01u/0,01m, gav laddningstider överstigande 10 minuter vid kallstart vilket är långt bortom vad som kan anses vara acceptabelt. Den slutgiltiga upplösningen på 0,1u/0,1m kan upplevas lite väl grovmaskigt med tanke på att roboten är ganska liten och kan önskas förflyttas mindre avstånd i verkligheten än i inkrement av 0,1m. En annan begränsning som finns med VR-applikationen är att den yta som roboten förflyttar sig på förväntas vara perfekt platt vilket skulle kunna ge en imperfekt representation i en situation när rummet i vilket roboten förflyttar sig varierar i lutning.

Med tillgång till en djupkamera torde det vara möjligt att ge en mer komplett representation av det rum som roboten befinner sig i verkligheten i vilket skulle inkludera eventuella ojämnheter i underlaget. För styrning av roboten används en form av 'peka-och-klicka'metodik medans förflyttning av armen sker genom att direkt dra i armen till önskat läge.

7.1.2 Framtidsutsikter för VR-applikationen

Vid eventuell vidare utveckling av applikationen skulle man kunna nyttja data från tillhörande djupkamera för att göra en grov uppskattning av var rummets väggar

under körning. Alternativt att rummets totala yta genereras och expanderas vid behov under körning med ett ännu mer finmaskigt rutnät för mer exakta och jämna rörelser som effekt. Under projektets gång släpptes en stor uppdatering Godot 4.0 vilken bl.a. kommer med en uppdaterad A*-algoritm vilket hade förmodats kunna leda till bättre prestanda kring generering och under körning modifikation av rutnätet för vägvalsproblematik.

Vidare innebär Godot 4 en övergång till vulkan-stöd vilket innebär många förbättringar kring nyttjandet av grafikkort för beräkningar vilket hade kunnat innebära bättre prestanda i VR-applikationen i allmänhet men speciellt om applikationen avses köras på en mindre kraftfull enhet som kör android, såsom Meta Quest 2. I den nuvarande versionen av VR-applikationen skapas ett fördefinierat rum med tillhörande rutnät för vägvalsproblematik.

I en framtida iteration hade man kunnat överväga att låta roboten sköta delar av genererandet av rummet såväl som varje nytillkommen rumsdels rutnät för att till högre del nyttja den relativt starka beräkningskraft som robotens Jetson Nano har. Fördelen med detta eventuella fördelande av ansvar mellan roboten och VR-enheten hade kunnat möjliggöra körandet av roboten på ännu svagare hårdvara. Det finns även möjlighet att vidare utforska olika sätt för en användare att ge instruktioner till roboten, t.ex. skulle man kunna låta användaren att lyfta roboten i VR och placera den i en annan del av rummet varefter applikationen överför instruktioner till roboten för att utföra denna förflyttning.

7.2 Interaktion mellan Robot och VR

I den nuvarande implementationen av styrning av roboten så sker rotation och förflyttning av roboten i ett obrutet och direkt flöde efter varandra. Detta kan leda till att användaren oavsiktligen råka förflytta roboten när endast rotation är önskvärt. I en vidareutveckling av användargränssnittet hade detta kunnat åtgärdas genom att implementera ett enskilt rotationsläge och ett vanligt förflyttningsläge alternativt en möjlighet för användaren att acceptera eller avböja föreslagen förflyttning av roboten efter roboten roterat klart eller använda joysticken för ändamålet.

Då de av robotföretaget bifogade 3D-modellen och logiken för skelettet ej kunde användas utan manuell riggning av roboten skedde genom Blender kvarstår viss problematik med 'self-collisions' mellan t.ex. arm och metallramen på vilken djupkameran sitter samt golvet. Om mer tid hade funnits skulle mer logik kunnat läggas till för att minimera risker för dessa oavsiktliga kollisioner genom införandet av fler begränsningar gällande vart den virtuella armen kan placeras i förhållande till roboten genom att expandera eller ändra robotens kollisionsarea.

7.2.1 Representation av sensordata från roboten i VR

Nyttjande data från robotens djup- och IR-kamera skulle det finnas möjlighet att dynamiskt kunna generera en mesh i den virtuella miljön utifrån djupdata från

kameran. På så vis kan det genereras olika former som roboten kan manipulera och förflytta. Detta kan vara ett bättre sätt för användaren att hålla koll på omgivningen tillskillnad från den nuvarande implementationen där kameraströmmen visas på en vägg eller på kamersfären som rullar på golvet eftersom användaren måste kolla ner på roboten när den manipuleras istället för att kolla på väggen.

Vidare kan robotens lidar användas för att bedöma avstånd vilket kan användas till förbättring av bedömning på avståndet roboten har åkt men även som en säkerhetsfunktion att stoppa den från att krocka med väggar och andra objekt. Lidar-data skulle även kunna användas för att bygga en uppdaterad karta för att ge användaren möjligheten att få en överblick över sin omgivning inuti VR. Roboten har även en vägmätningsfunktion som var avstängd som standard vilken hade kunnat nyttjas för mer exakta beräkningar av exakta rörelser.

8

Slutsats

I detta projekt har vi utforskat ett möjligt sätt för en användare att kunna kontrollera en Hiwonder JetAuto SLAM robot genom en VR-applikation. Användaren ges möjligheten att nyttja delar av de fördelar som den ökade närvaron som virtuell verklighet erbjuder. Applikationen kopplar upp sig mot roboten och möjliggör videoöverföring och skickade av instruktioner med en fördröjning nedåt 20 ms. Via VR kan användaren ges en uppsjö av användbar data på ett sätt som är mer greppbart och med riktiga proportioner till skillnad från vad en skärm kan återge. Roboten kan förflyttas genom att användaren pekar och klickar på marken medens robotarmen förflyttas genom att användaren greppar tag i denna och drar den till önskat läge.

Roboten som använts i detta projekt har, utöver de två kamerorna som använts, även en framåtriktad Lidar med 180° täckning. I ett framtida projekt skulle data från denna lidar kunna nyttjas för att ge en ännu mer representativ bild av det rum i vilket roboten rör sig såsom dynamisk generering av hinder och konsekvent deaktiverande av delar av rutnätet som täcks av objekt för att undvika kollisioner. Vidare utveckling och forkning krävs även för att utröna vilka gester och kontrollupplägg som kan ge ett mer intuitivt sätt att interagera med objekt och ge styrinstruktioner till roboten i den virtuella världen. Implementationsmöjligheterna är flera och beroende på vilken robot eller inriktning ett framtida projekt skulle beröra finns flertalet möjligheter till vidareutveckla den framtagna applikation.

Litteraturförteckning

[1] J. P. West and J. S. Bowman, "The domestic use of drones: An ethical analysis of surveillance issues," *Public Administration Review*, vol. 76, no. 4, pp. 649–659, 2016.

[2] G. Zhai, W. Zhang, W. Hu, and Z. Ji, "Coal mine rescue robots based on binocular vision: A review of the state of the art," *IEEE Access*, vol. 8, pp. 130561–130575, 2020.

[3] J. Gofus, S. Cerny, Y. Shahin, Z. Sorm, M. Vobornik, P. Smolak, A. Sethi, S. Marcinov, M. Karalko, J. Chek, *et al.*, "Robot-assisted versus conventional midcab: A propensity-matched analysis," *Frontiers in Cardiovascular Medicine*, p. 2414, 2022.

[4] P. Grimm, W. Broll, R. Herold, J. Hummel, and R. Kruse, "Vr/ar input devices and tracking," in *Virtual and Augmented Reality (VR/AR) Foundations and Methods of Extended Realities (XR)*, pp. 107–148, Springer, 2022.

[5] G. Buckingham, "Hand tracking for immersive virtual reality: opportunities and challenges," *Frontiers in Virtual Reality*, p. 140, 2021.

[6] R. Schroeter and F. Steinberger, "Pokémon drive: towards increased situational awareness in semi-automated driving," in *Proceedings of the 28th australian conference on computer-human interaction*, pp. 25–29, 2016.

[7] D. Tatić and B. Tešić, "The application of augmented reality technologies for the improvement of occupational safety in an industrial environment," *Computers in Industry*, vol. 85, pp. 1–10, 2017.

[8] "OpenXR Ecosystem Update — July 2020," The Khronos Group Inc. 2020 . [PowerPoint] Tillgänglig: https://www.khronos.org/assets/uploads/apis/OpenXR-EcoSystem-Update_Jul20.pdf.

[9] Z. Makhataeva and H. A. Varol, "Augmented reality for robotics: A review," *Robotics*, vol. 9, no. 2, p. 21, 2020.

[10] Software Freedom Conservancy, "Current projects," 2023. [Online] Hämtad: 2023-02-09. Tillgänglig: https://sfconservancy.org/projects/current/.

[11] A. M. Juan Linietsky and the Godot community, "Nodes and scenes," 2023. [Online] Hämtad: 2023-05-18
Tillgänglig: https://docs.godotengine.org/en/3.5/getting_started/step_by_step/nodes_and_scenes.html.

[12] Insomnia by Kong, Version 2023.2.0 for Windows, [Software], San Fransisco, CA, 2023.

[13] J. Mace, "Rosbridge summary," 2013. [Online] Hämtad: 2023-04-19
Tillgänglig: http://wiki.ros.org/rosbridge_library?distro=noetic.

[14] NVIDIA Jetson Nano System-on-Module, Santa Fe, USA NVidia Corp. [Online] Hämtad: 2023-05-18
Tillgänglig: https://developer.download.nvidia.com/assets/embedded/secure/jetson/Nano/docs/JetsonNano_DataSheet_DS09366001v1.1.pdf?neCl9gkfylB2St1YLJk0G0MsP8sSHFNBX9t-LuJtlK9Cb5-1xrBrYl9pXGcq_kYh9qaKxjyZDvRg_4ZMaJHPa8cSFwYy4qR8s6-w20eYYPQ4ij7xZT0AqjGfPxlh4C8xh_b2rergXgVatRi13ewO7l6mqzVkFcg39EX9nsMjjfeFYWgwVWYWFGH37RDUzQ==&t=eyJscyI6ImdzZW8iLCJsc2QiOiJodHRwczovL3d3dy5nb29nbGUuY29tLyJ9.

[15] Enlyft, "Companies using blender," Unknown. [Online] Hämtad: 2023-05-11
Tillgänglig: https://enlyft.com/tech/products/blender.

[16] E. O. Karaoğlu, D. Tükel, and B. Arthaya, "Vr based visualization of robotic workcells using cryengine," in *2019 International Conference on Mechatronics, Robotics and Systems Engineering (MoRSE)*, pp. 118–121, IEEE, 2019.

[17] Fred J, "File:båstad, sweden, a street adjacent to market square.jpg." [Elektronisk Bild] Hämtad: 2023-05-22
Tillgänlig: https://commons.wikimedia.org/wiki/File:B%C3%A5stad,_Sweden,_a_street_adjacent_to_market_square.jpg.

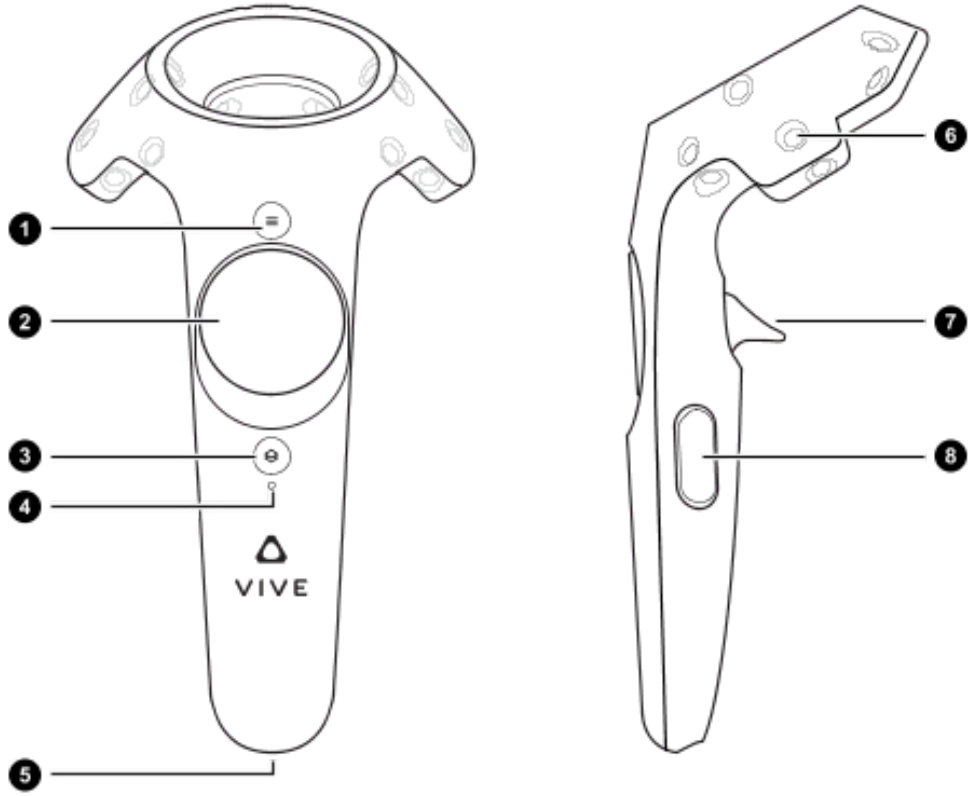
[18] HTC Vive, "About the vive controllers." [Elektronisk Bild] Hämtad: 2023-05-18
Tillgänlig: https://www.vive.com/us/support/vive/category_howto/about-the-controllers.html.

[19] Arc8ngel på r/oculus Reddit, "Oculus touch controller diagrams." [Elektronisk Bild] Hämtad: 2023-05-18
Tillgänlig: https://www.reddit.com/r/oculus/comments/8ycgov/oculus_touch_controller_diagrams/.

A

Appendix - Kontroller Layout

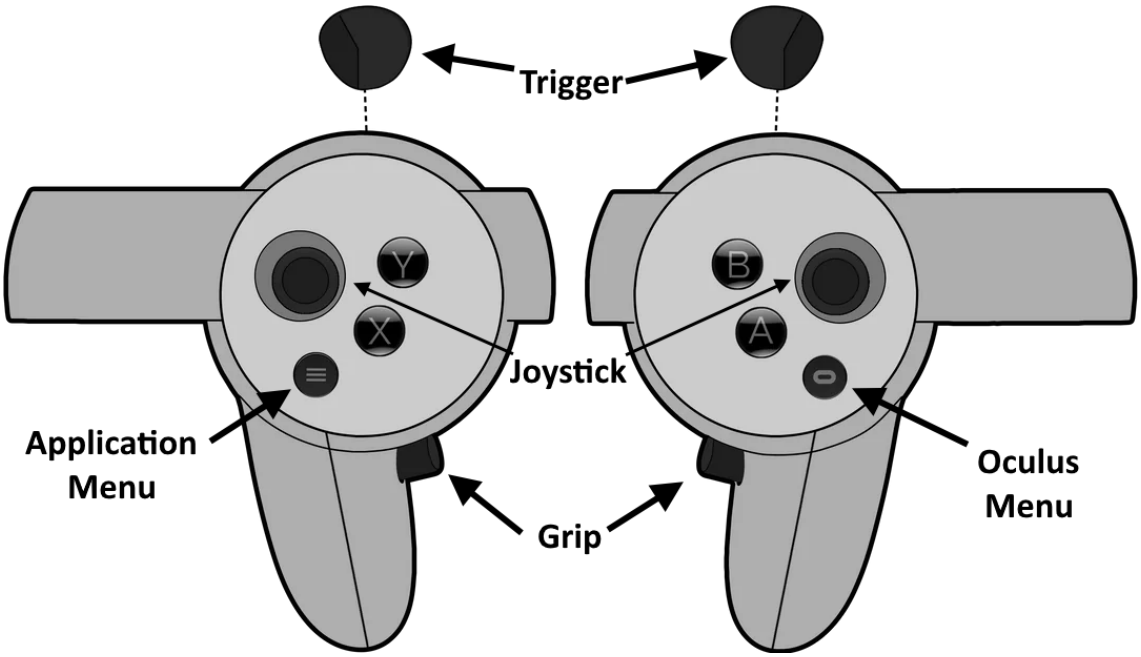
A.1 HTC Vive Kontroller



Ritning av kontroller för HTC Vive. Från [18].

1. Application Menu
2. Touchpad
3. System Menu
4. Battery Indicator
5. Charging Port
6. Tracking Sensor
7. Trigger
8. Grip

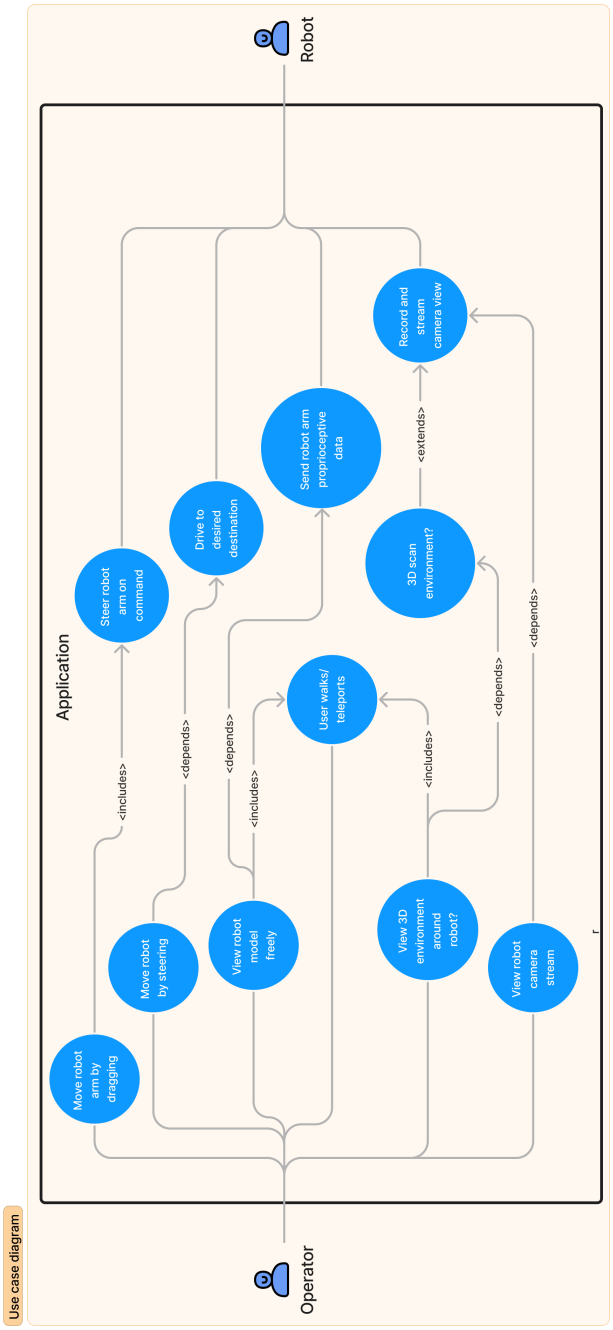
A.2 Oculus Touch Kontroller



Ritning av Oculus Touch kontroller. Från [19]. Modifierad med tillåtelse.

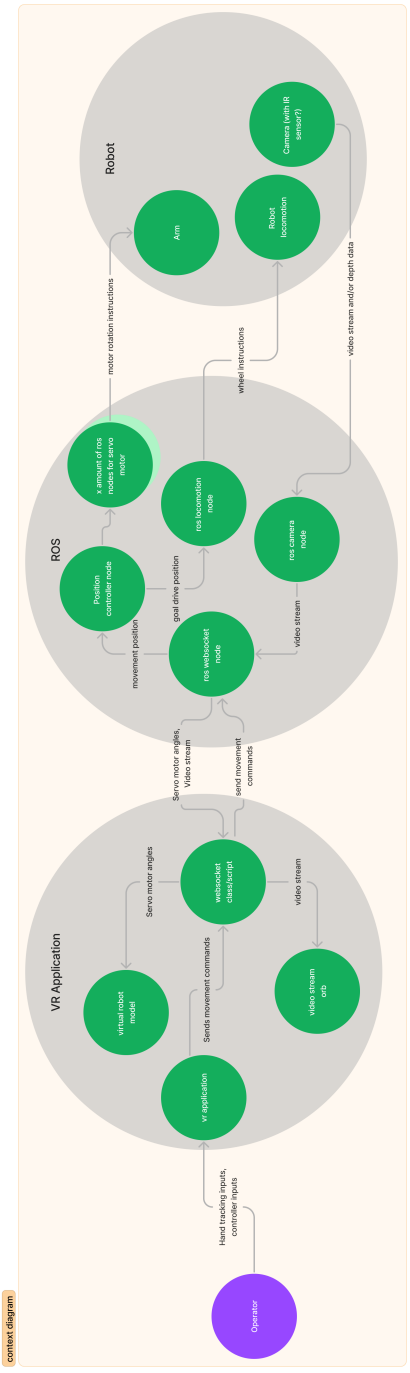
B

Appendix - Use Case Diagram



C

Appendix - Container Diagram



D

Appendix - Robot API

D.1 ROS-Bridge WebSocket API 1.0.0 documentation

The robot api uses a websocket created by the rosbridge-library and all paths/topics are specified by the topic/service field in messages.

D.1.1 Table of Contents

- PUB /rosapi/services
- SUB /rosapi/services
- PUB /rosapi/service_request_details
- SUB /rosapi/service_request_details
- PUB /rosapi/service_response_details
- SUB /rosapi/service_response_details

D.1.2 Servers

D.1.2.1 `wifi-direct` Server

- URL: `192.168.149.1:9090`
- Protocol: `Websocket`

The IP adress of the robot when it hosts its own WIFI network. An websocket connection can be stablished on port 9090.

D.1.2.2 `wifi-direct-stream` Server

- URL: `192.168.149.1:8080`
- Protocol: `HTTP`

The IP adress the robot uses when it host its own wifit network. To do HTTP request it can be done on port 8080. It may be used to receieve camera streams.

D.1.2.3 `usb` Server

- URL: `192.168.55.1:9090`
- Protocol: `Weboscket`

When connected to USB with NDIS configured then the websocket can be established trough this IP.

D.1.3 Operations

D.1.3.1 PUB `/{ros_topic}` Operation

The template for communicating with ros bridge.

D.1.3.1.1 Parameters

Name	Type	Description	Value	Constraints	Notes
ros_topic	string	-	-	format (`path`)	**required**

D.1.3.1.2 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

Accepts **one of** the following messages:

D.1.3.1.3 Message `rosbridgePublish` *Message to publish data to a ros topic.*

D.1.3.1.3.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"publish"`)	-	-
topic	string	The name of the ROS topic to publish to.	-	format (`path`)	-
msg	object	The message to be published to the ROS topic.	-	-	**additional properties are allowed**

Examples of payload *(generated)*

```
{
  "op": "publish",
  "topic": "/ros_topic",
  "msg": {}
}
```

D.1.3.1.4 Message `rosbridgeSubscribe`

Message to subscribe to a ROS topic through ROS-Bridge.

D.1.3.1.4.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"subscribe"`)	-	-
topic	string	The name of the ROS topic to subscribe to.	-	format (`path`)	-

Examples of payload *(generated)*

```
{
  "op": "subscribe",
  "topic": "/ros_topic"
}
```

D.1.3.1.5 Message `rosbridgeUnsubscribe`

Message to unsubscribe from a ROS topic through ROS-Bridge.

D.1.3.1.5.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**

Name	Type	Description	Value	Constraints	Notes
op	string	The operation to be performed.	allowed (`"unsubscribe"`)	-	-
topic	string	The name of the ROS topic to unsubscribe from.	-	format (`path`)	-

Examples of payload *(generated)*

```
{
  "op": "unsubscribe",
  "topic": "/ros_topic"
}
```

D.1.3.2 SUB `/{ros_topic}` Operation

The template for communicating with ros bridge.
When subscribed to a topic then the client recieves any publish messages from other nodes.

D.1.3.2.1 Parameters

Name	Type	Description	Value	Constraints	Notes
ros_topic	string	-	-	format (`path`)	**required**

D.1.3.2.2 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-

Name	Type	Description	Value	Constraints	Notes
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

D.1.3.2.3 Message `rosbridgePublish` *Message to publish data to a ros topic.*

D.1.3.2.3.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"publish"`)	-	-
topic	string	The name of the ROS topic to publish to.	-	format (`path`)	-
msg	object	The message to be published to the ROS topic.	-	-	**additional properties are allowed**

Examples of payload *(generated)*

```
{
  "op": "publish",
  "topic": "/ros_topic",
  "msg": {}
}
```

D.1.3.3 PUB `/{ros_service}` Operation

The template for a calling a service in ros bridge. The service may send responses.

D.1.3.3.1 Parameters

Name	Type	Description	Value	Constraints	Notes
ros_service	string	-	-	format (`path`)	**required**

D.1.3.3.2 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

D.1.3.3.3 Message `rosbridgeService` *Message to call a ROS service through ROS-Bridge.*

D.1.3.3.3.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"call_service"`)	-	-

Name	Type	Description	Value	Constraints	Notes
service	string	The name of the ROS service to call.	-	format (`path`)	-
args	object	The arguments to be passed to the ROS service.	-	-	**additional properties are allowed**

Examples of payload *(generated)*

```
{
  "op": "call_service",
  "service": "/ros_service",
  "args": {}
}
```

D.1.3.4 SUB `/{ros_service}` Operation

The template for a calling a service in ros bridge. The service may send responses.

D.1.3.4.1 Parameters

Name	Type	Description	Value	Constraints	Notes
ros_service	string	-	-	format (`path`)	**required**

D.1.3.4.2 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-

Name	Type	Description	Value	Constraints	Notes
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

D.1.3.4.3 Message `rosbridgeServiceResponse` *Message containing the response from a ROS service call through ROS-Bridge.*

D.1.3.4.3.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"service_response"`)	-	-
service	string	The name of the ROS service that was called.	-	format (`path`)	-
result	boolean	-	-	-	-
values	object | string	The response values returned by the ROS service.	-	-	**additional properties are allowed**

Examples of payload

Successful service call

```
{
  "op": "service_response",
  "service": "/ros_service",
  "result": true,
  "values": {
    "some_data": null
  }
}
```

Service not found

```
{
  "op": "service_response",
  "service": "/ros_service",
  "result": false,
  "values": "Service /ros_service does not exist"
}
```

D.1.3.5 PUB `/jetauto_controller/cmd_vel` Operation

Command to set a speed for the robot to move in a direction.

D.1.3.5.1 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

D.1.3.5.2 Message `vehicleVelocityCommand` *Movement velocity message to the robot.*

D.1.3.5.2.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**

Name	Type	Description	Value	Constraints	Notes
op	string	The operation to be performed.	allowed (`"publish"`)	-	-
topic	string	The name of the ROS topic to publish to.	allowed (`"/jetauto_controller/cmd_vel"`, `"/cmd_vel"`)	-	-
msg	object	The message to be published to the ROS topic.	-	-	**additional properties are allowed**
msg.linear	object	Contains speeds for moving forawrds or sideways.	-	-	**additional properties are allowed**
msg.linear.x	number	The speed for the robots forward/-backward movement. Positive results in forward movement while negative means backward movement.	-	[-0.5 .. 0.5]	-

Name	Type	Description	Value	Constraints	Notes
msg.linear.y	number	Decides the sideways speeds. Positive values makes the robot strafe to the left while negative values strafe to the right.	-	[-0.5 .. 0.5]	-
msg.angular	object	Contains rotation speeds.	-	-	**additional properties are allowed**
msg.angular.z	number	Decides the robot's rotation speed. Positve values make it rotate counter-clockwise while negative values make it rotate clockwise.	-	[-1 .. 1]	-

Examples of payload *(generated)*

```
{
  "op": "publish",
  "topic": "/jetauto_controller/cmd_vel",
  "msg": {
    "linear": {
      "x": 0,
      "y": 0
    },
    "angular": {
      "z": 0
    }
  }
```

```
}
```

D.1.3.6 PUB `/{joint}_controller/command_duration` Operation

Command to rotate a specifc joint with the speed determined by a duration paramter.

D.1.3.6.1 Parameters

Name	Type	Description	Value	Constraints	Notes
joint	string	The specific joint to control. Joint1 rotates the arm horizontally while joint2 and 3 rotates the arm vertically from bottom to top. Joint4 rotates claw itself vertically while r_joint opens or closes the claw.	allowed (`"joint1"`, `"joint2"`, `"joint3"`, `"joint4"`, `"r_joint"`)	-	**required**

D.1.3.6.2 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**

Name	Type	Description	Value	Constraints	Notes
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

D.1.3.6.3 Message `jointRotateCommand` *Message to set rotation of arm joints.*

D.1.3.6.3.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"publish"`)	-	-

Name	Type	Description	Value	Constraints	Notes
topic	string	The name of the ROS topic to publish to. Joint1 rotates the arm horizontally while joint2 and 3 rotates the arm vertically from bottom to top. Joint4 rotates claw itself vertically while r_joint opens or closes the claw.	allowed (`"/joint1_controller/command_duration"`, `"/joint2_controller/command_duration"`, `"/joint3_controller/command_duration"`, `"/joint4_controller/command_duration"`, `"/r_joint_controller/command_duration"`)	-	-
msg	object	The message to be published to the ROS topic.	-	-	**additional properties are allowed**
msg.data	number	The value to rotate to in radians. Some joints have constraints which results in that some joints won't rotate past a point.	-	-	-

Name	Type	Description	Value	Constraints	Notes
msg.duration	integer	The duration in millise-conds on how long the movement should take between it's starting pose and final pose.	default (1000)	>= 1	-

Examples of payload *(generated)*

```
{
  "op": "publish",
  "topic": "/joint1_controller/command_duration",
  "msg": {
    "data": 0,
    "duration": 1000
  }
}
```

D.1.3.7 PUB `/joint_states` Operation

Subscribe to a stream of states of the joints in the arm.

D.1.3.7.1 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
path	string	The HTTP path to start the websocket connection on.	default ("/")	format (path)	-

Name	Type	Description	Value	Constraints	Notes
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

Accepts **one of** the following messages:

D.1.3.7.2 Message `jointStateSubscribe` *Message to subscribe to Joint States.*

D.1.3.7.2.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"subscribe"`)	-	-
topic	string	The name of the ROS topic to subscribe to.	allowed (`"/joint_states"`)	format (`path`)	-

Examples of payload *(generated)*

```
{
  "op": "subscribe",
  "topic": "/joint_states"
}
```

D.1.3.7.3 Message `jointStateUnsubscribe` *Message to unsubscribe from Joint States.*

D.1.3.7.3.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"unsubscribe"`)	-	-
topic	string	The name of the ROS topic to unsubscribe from.	allowed (`"/joint_states"`)	format (`path`)	-

Examples of payload *(generated)*

```
{
  "op": "unsubscribe",
  "topic": "/joint_states"
}
```

D.1.3.8 SUB `/joint_states` Operation

Subscribe to a stream of states of the joints in the arm.

D.1.3.8.1 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

D.1.3.8.2 Message `jointStateResponse` *Message recieved when subscribed to joint states topic.*

D.1.3.8.2.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"publish"`)	-	-
topic	string	The name of the ROS topic to published from.	allowed (`"/joint_states"`)	format (`path`)	-
msg	object	The message to be published to the ROS topic.	-	-	**additional properties are allowed**
msg.header	object	-	-	-	**additional properties are allowed**
msg.header.stamp	object	-	-	-	**additional properties are allowed**
msg.header.stamp.secs	integer	-	-	format (`Unix Timestamp`)	-
msg.header.stamp.nsecs	integer	-	-	format (`timestamp`)	-
msg.header.frame_id	string	-	default (`"base_link"`)	-	-
msg.header.seq	integer	-	-	-	-

Name	Type	Description	Value	Constraints	Notes
msg.velocity	array	The velocity for each of the joints. The order int the array should match the name found in the names array.	-	-	-
msg.velocity (single item)	number	-	-	-	-
msg.effort	array	The effort for each of the joints if they're currently moving. The order should match the names in the names array.	-	-	-
msg.effort (single item)	number	-	-	-	-
msg.name	array	The names of all the joints available. The order should match the rest of the arrays.	default (`[joint1,joint2,joint3,joint4,r_joint]`)	-	-
msg.name (single item)	string	-	-	-	-

Name	Type	Description	Value	Constraints	Notes
msg.position	array	The current rotation in radians for each of the joints. The order should match the names array.	-	-	-
msg.position (single item)	number	-	-	-	-

Examples of payload *(generated)*


```
{
  "op": "publish",
  "topic": "/joint_states",
  "msg": {
    "header": {
      "stamp": {
        "secs": 0,
        "nsecs": 0
      },
      "frame_id": "base_link",
      "seq": 0
    },
    "velocity": [
      0,
      0,
      0,
      0,
      0
    ],
    "effort": [
      0,
      0,
      0,
      0,
      0
    ],
    "name": [
      "joint1",
      "joint2",
      "joint3",
```

```
      "joint4",
      "r_joint"
    ],
    "position": [
      0,
      0,
      0,
      0,
      0
    ]
  }
}
```

D.1.3.9 PUB `/{camera_source}/image_raw` Operation

Delivers a video feed from a camera. It can be from the Astra camera which provies rgb and depth versions or the usb camera mounted on the arm which has only a rgb mode.

D.1.3.9.1 Parameters

Name	Type	Description	Value	Constraints	Notes
camera_source	string	-	allowed (`"astra_cam/rgb"`, `"astra_cam/depth"`, `"astra_cam/ir"`, `"usb_cam"`)	-	**required**

D.1.3.9.2 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-

Name	Type	Description	Value	Constraints	Notes
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

Accepts **one of** the following messages:

D.1.3.9.3 Message `astraCamSubscribe` *Message to subscribe to a camera feed.*

D.1.3.9.3.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"subscribe"`)	-	-
topic	string	The name of the ROS topic to subscribe to.	allowed (`"/astra_cam/rgb/image_raw"`, `"/astra_cam/depth/image_raw"`, `"/astra_cam/ir/image"`, `"/usb_cam/image_raw"`, `"/astra_cam/rgb/image_raw/compressed"`, `"/astra_cam/depth/image_raw/comprssed"`, `"/astra_cam/ir/image/compressed"`, `"/usb_cam/image_raw/compressed"`)	format (`rgb`)	-

Examples of payload *(generated)*

```
{
  "op": "subscribe",
  "topic": "/astra_cam/rgb/image_raw"
}
```

D.1.3.9.4 Message `astraCamUnsubscribe` *Message to unsubscribe from a camera feed.*

D.1.3.9.4.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed ("unsubscribe")	-	-
topic	string	The name of the ROS topic to unsubscribe from.	allowed ("/astra_cam/rgb/image_raw", "/astra_cam/depth/image_raw", "/astra_cam/ir/image", "/usb_cam/image_raw", "/astra_cam/rgb/image_raw/compressed", "/astra_cam/depth/image_raw/comprssed", "/astra_cam/ir/image/compressed", "/usb_cam/image_raw/compressed")	format (path)	-

Examples of payload *(generated)*

```
{
  "op": "unsubscribe",
  "topic": "/astra_cam/rgb/image_raw"
}
```

D.1.3.10 SUB /{camera_source}/image_raw Operation

Delivers a video feed from a camera. It can be from the Astra camera which provies rgb and depth versions or the usb camera mounted on the arm which has only a rgb mode.

D.1.3.10.1 Parameters

Name	Type	Description	Value	Constraints	Notes
camera_source	string	-	allowed ("astra_cam/rgb", "astra_cam/depth", "astra_cam/ir", "usb_cam")	-	**required**

D.1.3.10.2 ws Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

D.1.3.10.3 Message `astraCamFeed` *Message to publish data to ROS-Bridge.*

D.1.3.10.3.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"publish"`)	-	-
topic	string	The name of the ROS topic to published from.	allowed (`"/astra_cam/rgb/image_raw"`, `"/astra_cam/depth/image_raw"`, `"/astra_cam/ir/image"`, `"/usb_cam/image_raw"`, `"/astra_cam/rgb/image_raw/compressed"`, `"/astra_cam/depth/image_raw/comprssed"`, `"/astra_cam/ir/image/compressed"`, `"/usb_cam/image_raw/compressed"`)	format (`path`)	-

Name	Type	Description	Value	Constraints	Notes
msg	object	The message to be published to the ROS topic.	-	-	**additional properties are allowed**
msg.header	object	-	-	-	**additional properties are allowed**
msg.header.stamp	object	-	-	-	**additional properties are allowed**
msg.header.stamp.secs	integer	-	-	format (`Unix Timestamp`)	-
msg.header.stamp.nsecs	integer	-	-	format (`timestamp`)	-
msg.header.frame_id	string	-	allowed (`"astra_cam_rgb_optical_frame"`, `"usb_cam"`)	-	-
msg.header.seq	integer	-	-	-	-
msg.data	string	The image data encoded in base64. May need to be converted to binary before it can be used.	-	format (`base64`)	-
msg.height	integer	The height of the image.	-	-	-
msg.encoding	string	The color encoding of the image.	allowed (`"rgb8"`, `"16UC1"`)	-	-
msg.format	string	-	-	-	-
msg.step	integer	-	-	-	-
msg.width	integer	-	-	-	-

msg.is_bigendian	integer	-	-	format (`boolean`)	-

Examples of payload *(generated)*

```
{
  "op": "publish",
  "topic": "/astra_cam/rgb/image_raw",
  "msg": {
    "header": {
      "stamp": {
        "secs": 0,
        "nsecs": 0
      },
      "frame_id": "astra_cam_rgb_optical_frame",
      "seq": 0
    },
    "data": "string",
    "height": 480,
    "encoding": "rgb8",
    "format": "rgb8; jpeg compressed bgr8",
    "step": 1920,
    "width": 640,
    "is_bigendian": 0
  }
}
```

D.1.3.11 SUB `/stream` Operation

Starts a stream from the Astra cam by first doing a HTTP get request which returns a multipart response. The request uses query parameters so a url make look like the following, `/stream?topic=/{camera_source}/image_raw&type=ros_compressed`

D.1.3.11.1 Parameters

Name	Type	Description	Value	Constraints	Notes
topic	string	-	allowed (`"/astra_cam/rgb/image_raw"`, `"/astra_cam/depth/image_raw"`, `"/astra_cam/ir/image"`, `"/usb_cam/image_raw"`)	-	**required**
type	string	-	allowed (`"ros_compressed"`, `""`)	-	**required**

D.1.3.11.2 `http` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
port	integer	-	default (`80`)	-	-

D.1.3.11.3 Message `<anonymous-message-11>`

D.1.3.11.3.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	string	It sends the encoded jpg image data as a http multipart stream.	-	format (`binary`)	-

Examples of payload *(generated)*

```
"string"
```

D.1.3.12 PUB `/scan` Operation

Subscribe to a stream of distances scanned from the lidar sensor.

D.1.3.12.1 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

Accepts **one of** the following messages:

D.1.3.12.2 Message `lidarScanSubscribe` *Message to subscribe to a camera feed.*

D.1.3.12.2.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"subscribe"`)	-	-
topic	string	The name of the ROS topic to subscribe to.	allowed (`"/scan"`)	format (`path`)	-

Examples of payload *(generated)*

```
{
  "op": "subscribe",
  "topic": "/scan"
}
```

D.1.3.12.3 Message `lidarScanUnsubscribe` *Message to unsubscribe from a camera feed.*

D.1.3.12.3.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"unsubscribe"`)	-	-

Name	Type	Description	Value	Constraints	Notes
topic	string	The name of the ROS topic to unsubscribe from.	allowed (`"/scan"`)	format (`path`)	-

Examples of payload *(generated)*

```
{
  "op": "unsubscribe",
  "topic": "/scan"
}
```

D.1.3.13 SUB `/scan` Operation

Subscribe to a stream of distances scanned from the lidar sensor.

D.1.3.13.1 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

D.1.3.13.2 Message `lidarScanFeed` *Message to unsubscribe from a camera feed.*

D.1.3.13.2.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"publish"`)	-	-
topic	string	The name of the ROS topic to unsubscribe from.	allowed (`"/scan"`)	format (`path`)	-
msg	object	The message to be published to the ROS topic.	-	-	**additional properties are allowed**
msg.header	object	-	-	-	**additional properties are allowed**
msg.header.stamp	object	-	-	-	**additional properties are allowed**
msg.header.stamp.secs	integer	-	-	format (`Unix Timestamp`)	-
msg.header.stamp.nsecs	integer	-	-	format (`timestamp`)	-
msg.header.frame_id	string	-	allowed (`"lidar_frame"`)	-	-
msg.header.seq	integer	-	-	-	-
msg.angle_min	number	-	-	-	-
msg.angle_max	number	-	-	-	-
msg.angle_increment	number	-	-	-	-
msg.scan_time	number	-	-	-	-
msg.range_min	number	-	-	-	-
msg.range_max	number	-	-	-	-
msg.time_increment	number	-	-	-	-
msg.ranges	array	-	-	-	-

Name	Type	Description	Value	Constraints	Notes
msg.ranges (single item)	number	-	-	-	-
msg.intensities	array	-	-	-	-
msg.intensities (single item)	number	-	-	-	-

Examples of payload *(generated)*


```
{
  "op": "publish",
  "topic": "/scan",
  "msg": {
    "header": {
      "stamp": {
        "secs": 0,
        "nsecs": 0
      },
      "frame_id": "lidar_frame",
      "seq": 0
    },
    "angle_min": -3.1415927410125732,
    "angle_max": 3.1415927410125732,
    "angle_increment": 0.005482709966599941,
    "scan_time": 0.0735989585518837,
    "range_min": 2.9119999408721924,
    "range_max": 2.9159998893737793,
    "time_increment": 0.00006422247679438442,
    "ranges": [
      [
        2.9159998893737793,
        2.9159998893737793,
        2.9119999408721924,
        2.9119999408721924,
        2.9519999027252197,
        2.9519999027252197,
        3.003999948501587,
        3.003999948501587,
        null,
        null
      ]
    ],
    "intensities": [
      [
        47,
```

```
        47,
        47,
        47,
        47,
        47,
        47,
        47,
        0,
        0
      ]
    ]
  }
}
```

D.1.3.14 PUB `/lidar_app/{operation}` Operation

A built in application in the robot when started the robot marches and stop when an object is too close based on the lidar data.

D.1.3.14.1 Parameters

Name	Type	Description	Value	Constraints	Notes
operation	string	-	allowed (`"enter"`, `"exit"`, `"heartbeat"`, `"set_parameters"`)	-	**required**

D.1.3.14.2 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-

Name	Type	Description	Value	Constraints	Notes
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

Accepts **one of** the following messages:

D.1.3.14.3 Message `lidarAppEnter`

Message to call a ROS service through ROS-Bridge.

D.1.3.14.3.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"call_service"`)	-	-
service	string	The name of the ROS service to call.	allowed (`"/lidar_app/enter"`)	format (`path`)	-
args	object	The arguments to be passed to the ROS service.	-	-	**additional properties are allowed**

Examples of payload *(generated)*

```
{
  "op": "call_service",
  "service": "/lidar_app/enter",
  "args": {}
}
```

D.1.3.14.4 Message `lidarAppSetParams`

Message to call a ROS service through ROS-Bridge.

D.1.3.14.4.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"call_service"`)	-	-
service	string	The name of the ROS service to call.	allowed (`"/lidar_app/set_parameters"`)	format (`path`)	-
args	object	The arguments to be passed to the ROS service.	-	-	**additional properties are allowed**
args.data	array	-	default (`[0,0,0]`)	-	-
args.data (single item)	number	-	-	-	-

Examples of payload *(generated)*

```
{
  "op": "call_service",
  "service": "/lidar_app/set_parameters",
  "args": {
    "data": [
      0,
      0,
      0
    ]
  }
}
```

D.1.3.14.5 Message `lidarAppHeartbeat` *Message to call a ROS service through ROS-Bridge.*

D.1.3.14.5.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"call_service"`)	-	-
service	string	The name of the ROS service to call.	allowed (`"/lidar_app/heartbeat"`)	format (`path`)	-
args	object	The arguments to be passed to the ROS service.	-	-	**additional properties are allowed**
args.data	boolean	-	-	-	-

Examples of payload *(generated)*

```
{
  "op": "call_service",
  "service": "/lidar_app/heartbeat",
  "args": {
    "data": true
  }
}
```

D.1.3.14.6 Message `lidarAppExit` *Message to call a ROS service through ROS-Bridge.*

D.1.3.14.6.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"call_service"`)	-	-

Name	Type	Description	Value	Constraints	Notes
service	string	The name of the ROS service to call.	allowed (`"/lidar_app/exit"`)	format (`path`)	-
args	object	The arguments to be passed to the ROS service.	-	-	**additional properties are allowed**

Examples of payload *(generated)*

```
{
  "op": "call_service",
  "service": "/lidar_app/exit",
  "args": {}
}
```

D.1.3.15 SUB `/lidar_app/{operation}` Operation

A built in application in the robot when started the robot marches and stop when an object is too close based on the lidar data.

D.1.3.15.1 Parameters

Name	Type	Description	Value	Constraints	Notes
operation	string	-	allowed (`"enter"`, `"exit"`, `"heartbeat"`, `"set_parameters"`)	-	**required**

D.1.3.15.2 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**

Name	Type	Description	Value	Constraints	Notes
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

D.1.3.15.3 Message `lidarAppResponse` *Message containing the response from a ROS service call through ROS-Bridge.*

D.1.3.15.3.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"service_response"`)	-	-
service	string	The name of the ROS service that was called.	allowed (`"/lidar_app/heartbeat"`, `"/lidar_app/enter"`, `"/lidar_app/set_parameters"`, `"lidar_app/exit"`)	format (`path`)	-
values	object	The response values returned by the ROS service.	-	-	**additional properties are allowed**
values.message	string	-	-	-	-
values.success	boolean	-	-	-	-
result	boolean	-	-	-	-

Examples of payload *(generated)*

```
{
  "op": "service_response",
  "service": "/lidar_app/heartbeat",
  "values": {
    "message": "string",
    "success": true
  },
  "result": true
}
```

D.1.3.16 PUB `/rosapi/topics` Operation

Access all available topics on the ROS server.

D.1.3.16.1 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

D.1.3.16.2 Message `rosapiTopicsService` *Message to call a ROS service through ROS-Bridge.*

D.1.3.16.2.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"call_service"`)	-	-
service	string	The name of the ROS service to call.	allowed (`"rosapi/topics"`)	format (`path`)	-
args	object	The arguments to be passed to the ROS service.	-	-	**additional properties are allowed**

Examples of payload *(generated)*

```
{
  "op": "call_service",
  "service": "rosapi/topics",
  "args": {}
}
```

D.1.3.17 SUB `/rosapi/topics` Operation

Access all available topics on the ROS server.

D.1.3.17.1 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-

Name	Type	Description	Value	Constraints	Notes
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

D.1.3.17.2 Message `rosapiTopicsServiceResponse` *Message containing the response from a ROS service call through ROS-Bridge.*

D.1.3.17.2.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"service_response"`)	-	-
service	string	The name of the ROS service that was called.	allowed (`"rosapi/topics"`)	format (`path`)	-
values	object	The response values returned by the ROS service.	-	-	**additional properties are allowed**
values.topics	array	-	-	-	-
values.topics (single item)	string	-	-	-	-
values.types	array	-	-	-	-
values.types (single item)	string	-	-	-	-

Examples of payload *(generated)*

```
{
  "op": "service_response",
  "service": "rosapi/topics",
  "values": {
    "topics": [
      "string"
    ],
    "types": [
      "string"
    ]
  }
}
```

D.1.3.18 PUB `/rosapi/topic_type` Operation

Get the type that a ROS topic uses.

D.1.3.18.1 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

D.1.3.18.2 Message `rosapiTopicTypeService` *Message to call a ROS service through ROS-Bridge.*

D.1.3.18.2.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"call_service"`)	-	-
service	string	The name of the ROS service to call.	allowed (`"rosapi/topic_type"`)	format (`uri`)	-
args	object	The arguments to be passed to the ROS service.	-	-	**additional properties are allowed**
args.topic	string	-	-	-	-

Examples of payload *(generated)*

```
{
  "op": "call_service",
  "service": "rosapi/topic_type",
  "args": {
    "topic": "string"
  }
}
```

D.1.3.19 SUB `/rosapi/topic_type` Operation

Get the type that a ROS topic uses.

D.1.3.19.1 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**

Name	Type	Description	Value	Constraints	Notes
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

D.1.3.19.2 Message `rosapiTopicTypeServiceResponse` *Message containing the response from a ROS service call through ROS-Bridge.*

D.1.3.19.2.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"service_response"`)	-	-
service	string	The name of the ROS service that was called.	allowed (`"rosapi/topic_type"`)	format (`path`)	-
values	object	The response values returned by the ROS service.	-	-	**additional properties are allowed**
values.type	string	-	-	-	-

Examples of payload *(generated)*

```
{
  "op": "service_response",
```

L

```
  "service": "rosapi/topic_type",
  "values": {
    "type": "string"
  }
}
```

D.1.3.20 PUB `/rosapi/subscribers` Operation

Get the available subscriber nodes for a specific topic.

D.1.3.20.1 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

D.1.3.20.2 Message `rosapiSubscribersService` *Message to call a ROS service through ROS-Bridge.*

D.1.3.20.2.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"call_service"`)	-	-

Name	Type	Description	Value	Constraints	Notes
service	string	The name of the ROS service to call.	allowed (`"rosapi/subscribers"`)	format (`path`)	-
args	object	The arguments to be passed to the ROS service.	-	-	**additional properties are allowed**
args.topic	string	-	-	-	-

Examples of payload *(generated)*

```
{
  "op": "call_service",
  "service": "rosapi/subscribers",
  "args": {
    "topic": "string"
  }
}
```

D.1.3.21 SUB `/rosapi/subscribers` Operation

Get the available subscriber nodes for a specific topic.

D.1.3.21.1 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-

Name	Type	Description	Value	Constraints	Notes
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

D.1.3.21.2 Message `rosapiSubscribersServiceResponse`

Message containing the response from a ROS service call through ROS-Bridge.

D.1.3.21.2.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"service_response"`)	-	-
service	string	The name of the ROS service that was called.	allowed (`"rosapi/subscribers"`)	format (`uri`)	-
values	object	The response values returned by the ROS service.	-	-	**additional properties are allowed**
values.subscribers	array	-	-	-	-
values.subscribers (single item)	string	-	-	-	-

Examples of payload *(generated)*

```
{
  "op": "service_response",
  "service": "rosapi/subscribers",
  "values": {
    "subscribers": [
```

```
      "string"
    ]
  }
}
```

D.1.3.22 PUB `/rosapi/publishers` Operation

Get the available publisher nodes for a topic.

D.1.3.22.1 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

D.1.3.22.2 Message `rosapiPublishersService` *Message to call a ROS service through ROS-Bridge.*

D.1.3.22.2.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"call_service"`)	-	-

Name	Type	Description	Value	Constraints	Notes
service	string	The name of the ROS service to call.	allowed (`"rosapi/publishers"`)	format (`path`)	-
args	object	The arguments to be passed to the ROS service.	-	-	**additional properties are allowed**
args.topic	string	-	-	-	-

Examples of payload *(generated)*

```
{
  "op": "call_service",
  "service": "rosapi/publishers",
  "args": {
    "topic": "string"
  }
}
```

D.1.3.23 SUB `/rosapi/publishers` Operation

Get the available publisher nodes for a topic.

D.1.3.23.1 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-

Name	Type	Description	Value	Constraints	Notes
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

D.1.3.23.2 Message `rosapiPublishersServiceResponse` *Message containing the response from a ROS service call through ROS-Bridge.*

D.1.3.23.2.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"service_response"`)	-	-
service	string	The name of the ROS service that was called.	allowed (`"rosapi/publishers"`)	format (`uri`)	-
values	object	The response values returned by the ROS service.	-	-	**additional properties are allowed**
values.publishers	array	-	-	-	-
values.publishers (single item)	string	-	-	-	-

Examples of payload *(generated)*

```
{
  "op": "service_response",
  "service": "rosapi/publishers",
  "values": {
    "publishers": [
```

```
      "string"
    ]
  }
}
```

D.1.3.24 PUB `/rosapi/services` Operation

Get the available services in a ROS server.

D.1.3.24.1 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

D.1.3.24.2 Message `rosapiServicesService` *Message to call a ROS service through ROS-Bridge.*

D.1.3.24.2.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"call_service"`)	-	-

Name	Type	Description	Value	Constraints	Notes
service	string	The name of the ROS service to call.	allowed (`"rosapi/services"`)	format (`path`)	-
args	object	The arguments to be passed to the ROS service.	-	-	**additional properties are allowed**

Examples of payload *(generated)*

```
{
  "op": "call_service",
  "service": "rosapi/services",
  "args": {}
}
```

D.1.3.25 SUB `/rosapi/services` Operation

Get the available services in a ROS server.

D.1.3.25.1 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

D.1.3.25.2 Message `rosapiServicesServiceResponse` *Message containing the response from a ROS service call through ROS-Bridge.*

D.1.3.25.2.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"service_response"`)	-	-
service	string	The name of the ROS service that was called.	allowed (`"rosapi/services"`)	format (path)	-
values	object	The response values returned by the ROS service.	-	-	**additional properties are allowed**
values.services	array	-	-	-	-
values.services (single item)	string	-	-	-	-

Examples of payload *(generated)*

```
{
  "op": "service_response",
  "service": "rosapi/services",
  "values": {
    "services": [
      "string"
    ]
  }
}
```

D.1.3.26 PUB `/rosapi/service_request_details` Operation

Get the type used when sending a request to a service.

D.1.3.26.1 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

D.1.3.26.2 Message `rosapiServiceRequestDetailsService`

Message to call a ROS service through ROS-Bridge.

D.1.3.26.2.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"call_service"`)	-	-
service	string	The name of the ROS service to call.	allowed (`"rosapi/service_request_details"`)	format (`path`)	-
args	object	The arguments to be passed to the ROS service.	-	-	**additional properties are allowed**
args.type	string	-	-	-	-

Examples of payload *(generated)*

```
{
  "op": "call_service",
  "service": "rosapi/service_request_details",
  "args": {
    "type": "string"
  }
}
```

D.1.3.27 SUB `/rosapi/service_request_details` Operation

Get the type used when sending a request to a service.

D.1.3.27.1 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

D.1.3.27.2 Message `rosapiServiceRequestDetailsServiceResponse` *Message containing the response from a ROS service call through ROS-Bridge.*

D.1.3.27.2.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**

Name	Type	Description	Value	Constraints	Notes
op	string	The operation to be performed.	allowed ("service_response")	-	-
service	string	The name of the ROS service that was called.	allowed ("rosapi/service_request_details")	format (path)	-
values	object	The response values returned by the ROS service.	-	-	**additional properties are allowed**
values.typedefs	array	-	-	-	-
values.typedefs.type	string	-	-	-	-
values.typedefs.fieldnames	array	-	-	-	-
values.typedefs.fieldnames (single item)	string	-	-	-	-
values.typedefs.fieldtypes	array	-	-	-	-
values.typedefs.fieldtypes (single item)	string	-	-	-	-
values.typedefs.fieldarraylen	array	-	-	-	-
values.typedefs.fieldarraylen (single item)	integer	-	-	format (int32)	-
values.typedefs.examples	array	-	-	-	-
values.typedefs.examples (single item)	string	-	-	-	-
values.typedefs.constnames	array	-	-	-	-
values.typedefs.constnames (single item)	string	-	-	-	-
values.typedefs.constvalues	array	-	-	-	-
values.typedefs.constvalues (single item)	string	-	-	-	-

Examples of payload *(generated)*


```
{
  "op": "service_response",
  "service": "rosapi/service_request_details",
```

```
  "values": {
    "typedefs": [
      {
        "type": "string",
        "fieldnames": [
          "string"
        ],
        "fieldtypes": [
          "string"
        ],
        "fieldarraylen": [
          0
        ],
        "examples": [
          "string"
        ],
        "constnames": [
          "string"
        ],
        "constvalues": [
          "string"
        ]
      }
    ]
  }
}
```


D.1.3.28 PUB `/rosapi/service_response_details` Operation

Get the type of the response that service may return.

D.1.3.28.1 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-

Name	Type	Description	Value	Constraints	Notes
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

D.1.3.28.2 Message `rosapiServiceResponseDetailsService` *Message to call a ROS service through ROS-Bridge.*

D.1.3.28.2.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"call_service"`)	-	-
service	string	The name of the ROS service to call.	allowed (`"rosapi/service_response_details"`)	format (`path`)	-
args	object	The arguments to be passed to the ROS service.	-	-	**additional properties are allowed**
args.type	string	-	-	-	-

Examples of payload *(generated)*

```
{
  "op": "call_service",
  "service": "rosapi/service_response_details",
  "args": {
    "type": "string"
  }
}
```

D.1.3.29 SUB `/rosapi/service_response_details` Operation

Get the type of the response that service may return.

D.1.3.29.1 `ws` Operation specific information

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
path	string	The HTTP path to start the websocket connection on.	default (`"/"`)	format (`path`)	-
timeout	integer	The time in seconds to wait for a WebSocket connection to be established.	default (`5`)	[1 .. 86400]	-

D.1.3.29.2 Message `rosapiServiceResponseDetailsServiceResponse`

Message containing the response from a ROS service call through ROS-Bridge.

D.1.3.29.2.1 Payload

Name	Type	Description	Value	Constraints	Notes
(root)	object	-	-	-	**additional properties are allowed**
op	string	The operation to be performed.	allowed (`"service_response"`)	-	-
service	string	The name of the ROS service that was called.	allowed (`"rosapi/service_response_details"`)	format (`path`)	-

Name	Type	Description	Value	Constraints	Notes
values	object	The response values returned by the ROS service.	-	-	**additional properties are allowed**
values.typedefs	array	-	-	-	-
values.typedefs (single item)	string	-	-	-	-
values.typedefs.fieldnames	array	-	-	-	-
values.typedefs.fieldnames (single item)	string	-	-	-	-
values.typedefs.fieldtypes	array	-	-	-	-
values.typedefs.fieldtypes (single item)	string	-	-	-	-
values.typedefs.fieldarraylen	array	-	-	-	-
values.typedefs.fieldarraylen (single item)	integer	-	-	format (`int32`)	-
values.typedefs.examples	array	-	-	-	-
values.typedefs.examples (single item)	string	-	-	-	-
values.typedefs.constnames	array	-	-	-	-
values.typedefs.constnames (single item)	string	-	-	-	-
values.typedefs.constvalues	array	-	-	-	-
values.typedefs.constvalues (single item)	string	-	-	-	-

Examples of payload *(generated)*

```
{
  "op": "service_response",
  "service": "rosapi/service_response_details",
  "values": {
    "typedefs": [
      {
        "type": "string",
        "fieldnames": [
          "string"
        ],
        "fieldtypes": [
```

```
          "string"
        ],
        "fieldarraylen": [
          0
        ],
        "examples": [
          "string"
        ],
        "constnames": [
          "string"
        ],
        "constvalues": [
          "string"
        ]
      }
    ]
  }
}
```

Institutionen för Data- och Informationsteknik
Chalmers tekniska högskola
Göteborgs universitet
Göteborg, Sverige
www.chalmers.se

GÖTEBORGS
UNIVERSITET

CHALMERS