UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO
POSGRADO EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

AGENTES INTELIGENTES TELE-OPERADOS UTILIZANDO AMBIENTES VIRTUALES

TESIS
QUE PARA OPTAR POR EL GRADO DE
MAESTRO EN CIENCIAS (COMPUTACIÓN)

PRESENTA:
DANIEL RUELAS MILANÉS

Dr. JESÚS SAVAGE CARMONA
FACULTAD DE INGENIERÍA

CIUDAD UNIVERSITARIA, CDMX, ENERO 2020

Agradecimientos

A la Universidad Nacional Autónoma de México, por el privilegio de ser parte de ella.

Al Consejo Nacional de Ciencia y Tecnología (CONACYT) por el apoyo económico recibido durante mis estudios.

A DGAPA UNAM por el apoyo recibido a través del proyecto PAPIIT IG100818

Índice general

Índice de figuras

Resumen

Resumen

En lugares de difícil acceso, situaciones de riesgo como el manejo de material peligroso o en el caso de contar con alguna discapacidad es útil un robot que pueda realizar las tareas requeridas. La navegación y reconocimiento de objetos puede presentar problemas para los robots actualmente, a comparación de un humano, esto puede tomar tiempo o ser impreciso. Este problema se puede resolver mediante la creación de un robot teleoperado y autónomo que realice navegación y tareas sencillas de manera automática, mientras que en escenarios más complicados, la navegación y las tareas sean realizadas de manera remota por un humano. Uno de los retos que presenta la teleoperación es la velocidad de transmisión de red o latencia. Este trabajo explora la creación de un robot virtual que represente un robot real teleoperado, utilizando una interfaz virtual inmersiva de manera que se puedan realizar ciertas tareas a través del ambiente virtual que mantiene los cuadros por segundo de manera más estable y rápida en un ambiente con visión de 360°, a la vez de reducir la latencia en la posición del robot y los objetos en el ambiente al enviar y recibir menos información por la red.

Abstract

In places of difficult access, risk situations such as the handling of hazardous material or in the case of having a disability, a robot that can perform the required tasks is useful. Navigation and object recognition can present problems for robots today, compared to a human, this can take time or be inaccurate. This problem can be solved by creating a teleoperated and autonomous robot that performs navigation and simple tasks automatically, while in more complicated scenarios, navigation and tasks are performed remotely by a human. One of the challenges that teleoperation presents is the network transmission speed or latency. This work explores the creation of a virtual robot that represents a real teleoperated robot, using an immersive virtual interface so that certain tasks can be performed through the virtual environment that keeps the frames per second more stable and fast in an environment with 360 ° vision, while reducing latency in the position of the robot and objects in the environment by sending and receiving less information over the network.

Capítulo 1

Introducción y Antecedentes

1.1. Definición y formulación del problema

En lugares de difícil acceso, situaciones de riesgo como el manejo de material peligroso o en el caso de contar con alguna discapacidad es útil un robot que pueda realizar las tareas requeridas. La navegación y reconocimiento de objetos puede presentar

(a) Interfaz remota (b) Robots de asistencia

Figura 1.1: Robots e interfaz remota[Gat08]

problemas para los robots actualmente, a comparación de un humano, esto puede tomar tiempo o ser impreciso. Este problema se puede resolver mediante la creación de un robot teleoperado y autónomo que realice navegación y tareas sencillas de manera automática, mientras que en escenarios más complicados, la navegación y las tareas sean realizadas de manera remota por un humano. Se requiere una interfaz inmersiva para poder observar el ambiente remoto de mejor manera y un modo de evitar o mejorar los problemas de latencia en la red.

1.2. Ubicación del proyecto

1.2.1. Área de investigación

Ciencia e Ingeniería de la Computación.

Las ciencias de la computación se centran en el entendimiento, diseño y desarrollo de programas y computadoras. En su núcleo, las ciencias de la computación se centran en la información, transformación de ésta y en algoritmos.[Uni19]

La ingeniería de la computación trata con el diseño, desarrollo y operación de sistemas computacionales. En su núcleo, la ingeniería en computación se concentra en dispositivos digitales y computadoras, además del software que los controla.[Uni19]

1.2.2. Materia de la investigación

Robótica y ambientes virtuales

Robótica

Un robot es una máquina diseñada para ejecutar una o más tareas de manera automática con velocidad y precisión. Los robots de asistencia son los que pueden realizan tareas repetitivas dentro o fuera del hogar, como barrer, limpiar, transportar objetos. Virbot, una arquitectura robótica, logra definir los comandos necesarios usando planificación de acciones de inteligencia artificial y comportamientos reactivos con una descripción del entorno de trabajo.

Figura 1.2: Robot HSR de Toyota[Toy18]

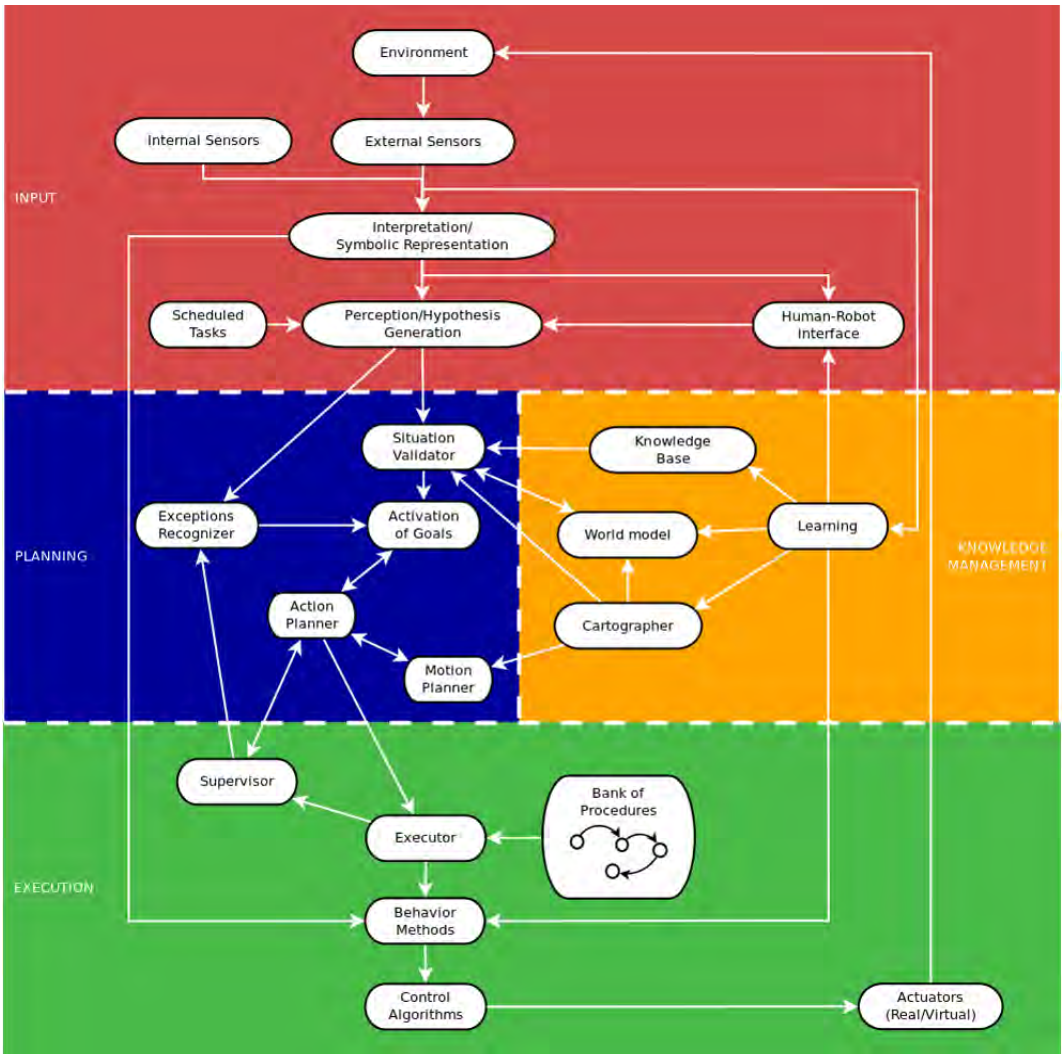


Figura 1.3: Arquitectura Virbot[Sav18]

Teleoperación y telepresencia

La teleoperación o telecontrol se refiere a el "Mando de un aparato o sistema, ejercido a distancia"[Esp19] Los inicios de los sistemas teleoperados datan de 1898 y el primer sistema considerado teleoperado fue un pequeño barco controlado mediante señales de radio creado por el ingeniero e inventor Nikola Tesla.

La teleoperación actualmente es utilizada para manipulación de robots en el espacio para la exploración planetaria, ejemplos actuales incluyen el Mars Exploration Rover y el Curiosity (Rover). Otro uso de la teleoperación es en vehículos marítimos que son usados para reparar plataformas petroleras alejadas de la costa, izar naves hundidas, la exploración de las ruinas del Titanic fue hecha utilizando estos sistemas. En medicina también se realiza investigación en dispositivos de este estilo para cirugías con sistemas menos invasivos, con este tipo de sistemas un médico puede realizar operaciones haciendo únicamente pequeños agujeros suficientemente grandes para el manipulador y sin necesidad de abrir una gran cavidad.[Wik19] Otro de sus usos es para manipulación de materiales peligrosos como los radioactivos.

La telepresencia, por otro lado, fue introducida por Marvin Minsky en 1980, y dando referencia a su visión de telepresencia al autor de ciencia ficción Robert A. Heinlein en su trabajo "Waldo"de 1948. En este trabajo, Minsky, presenta la idea de poder tener dispositivos y robots teleoperados en donde todas las experiencias sensoriales, visión, olor, color, tacto; puedan ser percibidas por la persona que ejerce el control a distancia y de esta manera poder tener una mayor experiencia del lugar donde se encuentra el dispositivo teleoperado y así lograr un mayor control y manejo, evitando posibles riesgos asociados a ambientes peligrosos o evitando el transporte de sujetos capacitados para las tareas necesarias [MAR80].

Realidad y Realidad Virtual

Definir lo que es real y no, es un tema demasiado complejo, por lo que para fines de este trabajo definiremos la realidad en términos de los objetos que tienen existencia de manera objetiva. La realidad virtual será definida en términos de los objetos creados mediante sistemas computacionales y que no tienen una existencia objetiva, solo en esencia o efecto.

Realidad Mixta

En el trabajo "Augmented Reality: A class of displays on the reality-virtuality continuum", Paul Milgram define el "Continuo Realidad-Virtualidad"[MK94] Esta clasifi-

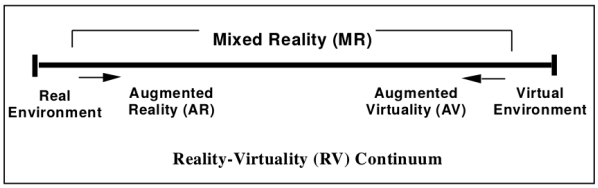


Figura 1.4: Continuo Realidad-Virtualidad[MK94]

cación la podemos ver en la figura 1.4. En el extremo izquierdo, los ambientes completamente reales, y en el extremo derecho los ambientes completamente virtuales. En la parte entre estos dos extremos, podemos ver la realidad mixta, la cual contiene objetos de ambientes reales y objetos de ambientes virtuales de manera conjunta.

Realidad Aumentada y Virtualidad Aumentada

Dentro de la realidad mixta, se pueden definir dos subcategorías, la Realidad Aumentada y la Virtualidad Aumentada. La Realidad Aumentada se refiere a ambientes en los cuales los objetos mostrados son predominantemente reales. La Virtualidad Aumentada se refiere a ambientes en los que los objetos mostrados son predominantemente virtuales. Aunque estas definiciones son útiles para contar con una taxonomía, mientras más avance la tecnología, podría llegar un punto en el que sea difícil saber si el ambiente es predominantemente real o virtual. [MK94]

Realidad Mixta y Robótica

Los ambientes virtuales en la robótica han sido utilizados principalmente para la localización del robot y sus partes móviles como brazos y cabeza. Es útil para los algoritmos de planeación contar con un ambiente virtual en el que se puedan realizar pruebas, previo a realizar las tareas en la realidad. También es útil para poder conocer y visualizar la información del ambiente en el que se encuentra el robot.

Figura 1.5: Simulador Gazebo[Tok19]

1.3. Objetivos de la investigación

Objetivo general

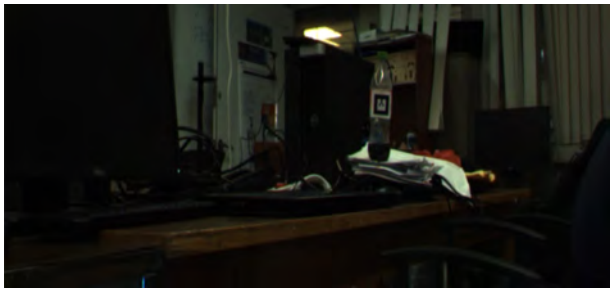
Crear un robot virtual que represente a un robot real remoto para realizar teleoperación sobre éste utilizando una interfaz inmersiva fácil de utilizar y exportar a otras plataformas, que requiera pocos recursos computacionales para reproducir el ambiente virtual, utilizando abstracción de datos para una comunicación más eficiente y rápida.

Objetivos específicos

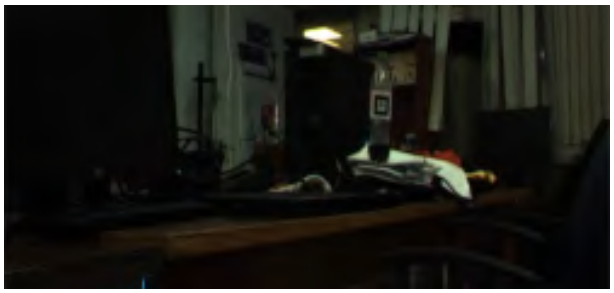
- Programación de la interfaz que se utilizará para la teleoperación.
- Programación de la interfaz de comunicación entre el robot en el ambiente real remoto y la interfaz virtual local.
- Creación del ambiente virtual sobre el que funcionará la interfaz.
- Creación dinámica de objetos en el ambiente virtual para objetos que pueden cambiar de posición en el ambiente real.

1.4. Hipótesis

La visualización de ambientes remotos se complica debido a la gran cantidad de información que se requiere enviar utilizando una señal de video, provocando intermitencia e interrupción en la comunicación, incluso utilizando algoritmos para optimización y compresión de datos, existe un límite para poder distinguir los detalles en las imágenes debido a la pérdida de información, esta misma información puede ser procesada por el robot y ser enviada de manera más eficiente y sin pérdida.



(a) Imagen 1920x898 pixeles



(b) Imagen 250x117 pixeles

Figura 1.6: Diferencia de definiciones

El reconocimiento automático de ciertos objetos y la manipulación de estos por un robot, aún con los avances en investigación es muy complicado. La utilización de un robot teleoperado puede ser útil para resolver este tipo de problemas, además de la utilización de un ambiente virtual para disminuir la cantidad de información que se requiere enviar, para un control más eficiente y rápido. Mediante la utilización del robot HSR de Toyota, el cual se encuentra en el laboratorio de Biorobótica del Posgrado de Ingeniería de la UNAM, y un simulador creado en Unity 3D,

Figura 1.7: Robot HSR de Toyota

se programaron y probaron los algoritmos necesarios para la teleoperación del mismo. El robot cuenta con cámaras, sensores de profundidad, llantas, láser, un brazo robótico y una pantalla que nos permiten interactuar y censar el ambiente. En cuanto al retraso

(a) Cámaras y sensores (b) Láser HSR

Figura 1.8: Sensores de HSR de Toyota

de la transmisión de video, se propone utilizar una mezcla entre esta transmisión y la utilización de un ambiente virtual local previamente hecho que represente el entorno real remoto, de tal manera que únicamente se transmita la posición del robot para su localización en el ambiente virtual, reduciendo considerablemente los problemas de comunicación. En el ambiente real puede haber elementos que cambien de posición

(a) Ambiente real (b) Ambiente virtual

Figura 1.9: Ambiente donde interactua robot y usuario

durante la navegación y estos cambios pueden no verse en el ambiente virtual, de

manera que hay que localizar su posición en el ambiente real y representarlos en el ambiente virtual en la posición adecuada. Utilizando etiquetas QR podemos encontrar la información de posición y diferenciar los objetos fácilmente. De esta manera

(a) Botella en ambiente real (b) Botella en ambiente virtual

Figura 1.10: Objetos no fijos

se pretende que la interfaz inmersiva virtual resuelva gran parte de los problemas de transmisión en red que puede haber en sistemas teleoperados y que a la vez sea útil para robots de asistencia.

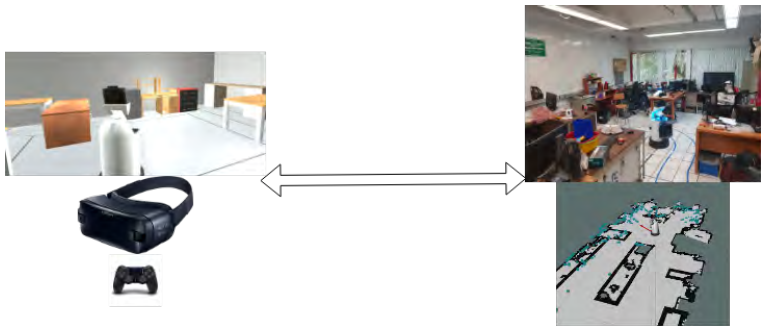


Figura 1.11: Configuración final del proyecto

1.5. Justificación

Debido a la gran cantidad de información que se requiere enviar para la transmisión de video y el tiempo de reacción del robot en la teleoperación, ésta puede ser lenta y de mala calidad. El presente trabajo se enfoca en la telepresencia y teleoperación de un robot utilizando un ambiente virtual para mejorar la velocidad con la que el usuario puede teleoperar e interactuar con el ambiente donde se encuentra el robot.

1.6. Organización de la tesis

- **Antecedentes:** Conocer las tecnologías existentes actualmente y saber que características debe cumplir un sistema para poder innovar.
- **Metodología:** Hacer un análisis de las tecnologías disponibles, herramientas y planeación del funcionamiento del sistema.
- **Implementación:** Con base en la metodología y el análisis de antecedentes, proceder a la creación del sistema procurando corregir detalles no previstos.
- **Pruebas:** Realizar pruebas para corroborar que el sistema funciona adecuadamente y que su desempeño sea óptimo.

Capítulo 2

Estado del Arte

2.1. Robots teleoperados

Al operar un robot a distancia es deseable tener retroalimentación acerca del ambiente remoto y es importante considerar el retraso que podría existir en la comunicación. Esta retroalimentación al usuario puede ser de manera auditiva, visual o táctil. Actual-

Figura 2.1: Robot controlado con gamepad[Syn11]

mente los robots pueden ser teleoperados utilizando *gamepads* u otros dispositivos de

control principalmente para realizar tareas imposibles para los robots hoy en día, esto se realiza de manera local o remota. En la actualidad existen diversos tipos de robots teleoperados para exploración espacial, marítima, manejo de materiales peligrosos, de uso médico, entre otros.

En cuanto a la exploración espacial, podemos citar diversos ejemplos como son: Mars Exploration Rover (MER), Curiosity, Dextre y Robonaut. MER y Curiosity son robots creados para la exploración de la superficie de marte, cuentan con navegación autónomo y con teleoperación en caso de ser requerida o para una exploración que la navegación autónoma no sea capaz de realizar. Dextre son dos brazos robóticos capaces de hacer reparaciones en la nave o estación espacial cuando se requiere, previo a esto, los astronautas realizaban las reparaciones haciendo caminatas espaciales. Robonaut es un robot humanoide diseñado a diferencia de los vehiculos y brazos roboticos, para realizar tareas que requieren más destreza, fue diseñado para asistir a los astronautas en las misiones espaciales.

Figura 2.2: Mars Rover, Dextre y Robonaut

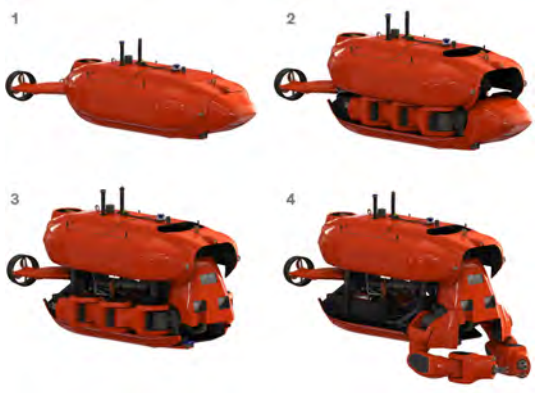


Figura 2.3: Aquanaut[Hou19]

En la exploración maritima existen robots como Aquanaut, un

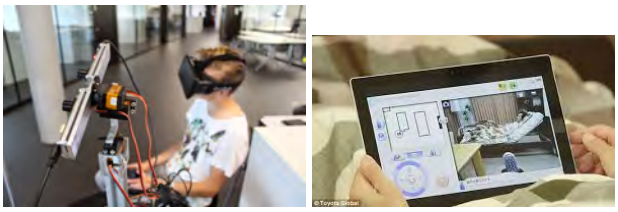
robot capaz de transformarse de una forma de submarino a una humanoide para poder realizar una navegación rápida y manipulación fina de objetos, su teleoperación dentro del agua permite mantener a salvo a los operadores de la presión del mar y otros peligros. La manipulación de materiales peligrosos como los radioactivos se utilizan brazos robóticos similares a Dextre que permiten a los operadores estar lejos de estos materiales. En cuanto a la operación para uso médico un ejemplo es el robot Da Vinci el cual permite reducir riesgos para el paciente como el temblor de la mano al operar, aumento de la capacidad de maniobra dentro del cuerpo y una operación menos invasiva.

Figura 2.4: Robot Da Vinci[Equ]

2.2. Telepresencia mediante dispositivos portátiles y teleoperación mediante cascos (HMDs).

Aun con los avances en investigación y tecnología, es difícil lograr que un robot reconozca ciertos objetos o que entienda algunas acciones que debe realizar. Una manera de resolver esto han sido los sistemas teleoperados y como parte de estos, es muy importante su interfaz. Una interfaz que nos permita ver mundo remoto de manera inmersiva es muy útil, ya que nos deja explorar completamente el ambiente y tomar mejores decisiones al momento de teleoperar. Las interfaces inmersivas de teleoperación existentes más actuales presentan un sistema que mapea los movimientos del usuario al robot teleoperado para realizar el movimiento correspondiente y ver lo mismo mediante el uso de cámaras.

Otras de las soluciones ha sido la utilización de dispositivos móviles para su uso de manera sencilla.



(a) Interfaz inmersiva[Uni15] (b) Interfaz a través de dispositivo móvil[GRA15]

Figura 2.5: Interfaces remotas actuales

Teleyes [WYTK18] es un sistema basado en visión estereoscópica y rastreo del movimiento de la cabeza. En el trabajo realizado sobre este sistema se encontró que el sistema puede mejorar la eficiencia y precisión al controlar sistemas de vehículos no tripulados, teleoperados. Teleyes sincroniza el movimiento de la cabeza del usuario con el dispositivo controlado. Mediante el uso de cámaras estéreo provee de una mejor percepción de la profundidad del ambiente al usuario.

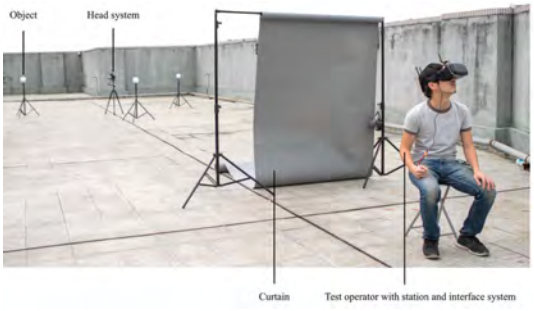
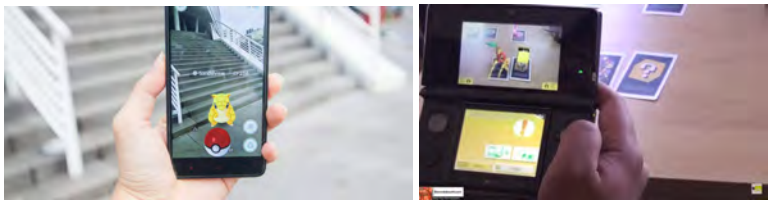


Figura 2.6: Sistema Teleyes en uso[WYTK18]

2.3. Realidad Mixta

Los dispositivos utilizados para realidad mixta son de una gran variedad, van desde dispositivos móviles, consolas de videojuegos, proyecciones en ambientes reales, hasta ”head mounted displays” o dispositivos tipo casco con pantallas. Estos dispositivos son capaces de reproducir objetos virtuales en alguna medida, o son apoyados de algún otro dispositivo para este efecto. También es posible que cuenten con cámaras para la captura de video de ambientes reales.

Figura 2.7: Head Mounted Displays para realidad virtual[Cat16]



(a) Teléfono inteligente[Pri18] (b) Consola de videojuegos[Boo11]

Figura 2.8: Dispositivos portátiles y Realidad Aumentada.

Los teléfonos inteligentes son capaces de reproducir modelos tri-dimensionales, y mediante el uso de la cámara también integrada a éstos, es posible la reproducción de contenido de realidad virtual y aumentada (realidad mixta). También es posible la reproducción de este tipo de contenido en otros dispositivos portátiles como consolas de videojuegos [figura 2.8].

Figura 2.9: Head Mounted Displays para Realidad Aumentada[LOP15]

Los HMD pueden ser utilizados para dar una experiencia completamente inmersiva donde el usuario únicamente puede observar el ambiente virtual y/o una reproducción indirecta de algún ambiente real. Existen HMD que son capaces de reproducir modelos tri-dimensionales virtuales sobre una pantalla transparente, lo cual permite ver el mundo real al mismo tiempo y sobre poner los objetos

virtuales [figura 2.9].

Figura 2.10: CAVE[MCR+17]

Existen, además, ambientes de realidad virtual inmersivos que utilizan proyectores, estos permiten tener una experiencia mucho más libre ya que no dependen de tener algún dispositivo manipulado por el usuario ó sobre su cabeza como en el caso de los HMD utilizados para la visualización [figura 2.10].

2.4. Video transmitido en vivo

La transmisión de contenido multimedia ha sido muy importante para la comunicación, y un punto importante a tratar ha sido la latencia que se refiere al tiempo de respuesta entre un estímulo y la reacción, en el caso del video es el tiempo en el que se obtiene una imagen y la entrega y/o visualización de esta por parte del receptor.

La transmisión de video presenta un gran reto debido a la cantidad de información que se requiere enviar a través de un medio como internet. Dado esto, se han logrado muchos avances utilizando codificadores para la compresión de información y que su envío sea más eficiente. En trabajos como Overhead and performance of low latency live streaming using MPEG-DASH [BCL14] se muestra un intento por reducir la latencia provocada por la red obteniendo tiempos de respuesta menores a 240 milisegundos en una red local y en otros softwares definiendo latencia ultra baja como menor a 1 segundo.[Deb18]

2.5. Sistemas teleoperados con ambientes virtuales

Actualmente existen algunos ambientes virtuales inmersivos para teleoperación como lo es VETO [WBM+18], este permite la navegación de un pequeño robot y utilizando la información sensorial del robot, generar un ambiente virtual que permita una mejor teleoperación.

Figura 2.11: Interfaz VETO[WBM+18]

VETO utiliza Unreal Engine y Oculus Rift para cumplir con su propósito, además de leap motion para la manipulación.

2.6. Trabajo presente

Los sistemas y robots teleoperados actualmente atacan solo algunos aspectos de la realización de tareas. Pueden llegar a ser muy específicos a su plataforma y a su control para la teleoperación, un control limitado y ser complicados de exportar a otras plataformas.

La transmisión de datos del ambiente puede ser difícil para algunos de estos sistemas debido a la gran cantidad de información que se requiere enviar para la visualización, además de contar con información limitada del ambiente real. La latencia depende no únicamente del sistema implementado, si no del tráfico y condiciones de la red, actualmente sistemas capaces de transmitir a grandes velocidades únicamente estan disponibles para redes locales, debido a esto, es importante que el robot cuente, además de la teleoperación, con un sistema inteligente que permita la operación automática del mismo para tareas sencillas, haciendolo un un agente inteligente con capacidad para teleoperación.

Aún con la codificación de la transmisión de video para reducir la latencia, esta tiene un límite en el que la pérdida de información al hacer la codificación ya no permite la correcta identificación y manipulación de objetos por parte del tele-operador. En este trabajo se exploran estas deficiencias creando el sistema de tal manera que sea fácil de utilizar y exportar a otras plataformas, así como necesitar pocos recursos computacionales para reproducir el ambiente, mejora de la transmisión de datos realizando mediciones de latencia e inclusión de aspectos del ambiente real en el ambiente virtual de manera dinámica para realizar una mejor teleoperación, uniendo esto con un agente inteligente que permita la realización de tareas sencillas.

Capítulo 3

Metodología

3.1. Hardware

3.1.1. Robot de servicio

La Organización Internacional de Normalización (ISO) define un robot de servicio como un robot que realiza tareas útiles para humanos o equipos, excluyendo aplicaciones de automatización industrial[ISO12]. Los robots de servicio son un área de aplicación emergente para tecnologías centradas en humanos. El auge de los robots domésticos y de asistencia personal pronostica una sociedad de colaboración humano-robot ya que pueden resolver muchas tareas como atención a pacientes con incapacidad para moverse y realizar tareas del hogar.[HBG+13]

Toyota HSR

El Toyota HSR (Human Support Robot) es un robot de servicio que fue creado como una herramienta que tiene dos funciones, trabajo físico y comunicación.

Figura 3.1: Robot HSR de Toyota[Toy15]

El robot cuenta con una base de 430 milímetros, una altura de 1 metro a 1.35 metros cuando el torso se encuentra en posición extendida. Cuenta con 11 grados de libertad y un peso de 37 kilogramos aproximadamente.

Las especificaciones son las siguientes:

Base móvil	Sistema de manejo	Mecanismo de movimiento omnidireccional
	Ambiente de uso	Interiores, paso de 5 milímetros, ángulo de subida 5 grados
	Sensores	Sensor de rango de medición láser (UST-20LX), IMU (6DOF) y un sensor magnético de paro.
	Velocidad máxima	0.22 m/s (0.8km/h)
	Capacidad de expansión	USB3.0 x1
Elevación	Capacidad de elevación	690 milímetros
	Características	Mecanismo telescópico, Mecanismo de compensación de peso
Brazo	Largo	Cerca de 600mm
	Rango movible (Altura)	0 1.350 mm
	Rango Movible (Profundidad)	450 mm (desde la cara del extremo de la base móvil)
	Carga útil (recomendada/máxima)	0.5 / 1.2kg
	Velocidad máxima de la mano	1.1m / s
	Sensores	Codificador de ángulo articular tipo absoluto, Sensor de fuerza de 6 ejes
	Caracteristicas	Mecanismo de compensación de peso, Par flexible detectable par x 2 ejes

Pinza	Velocidad de apertura y cierre	menor a 0.4s (en el rango máximo de apertura y cierre)
	Máxima fuerza de agarre	40N
	Rango máximo de apertura y cierre.	135mm
	Máxima fuerza de succión	5N
	Sensores	Potenciómetro, Sensor de fuerza de agarre, Cámara gran angular
	Caracteristicas	Mecanismo de puntas de dedos paralelas flexibles, Mecanismo de articulación flexible, Mecanismo de autobloqueo, Mecanismo de succión, Agarre de potencia compatible
Cabeza	Sensores	Codificador de ángulo articular tipo absoluto, Sensor RGB-D (Xtion PRO LIVE) x1, Cámara estéreo x1, Cámara gran angular x1, Conjunto de micrófonos x1
	Capacidad de expansión	USB2.0 x1, Salida 12V-0.5A x1

Monitor	tamaño	7.0 pulgadas
	Resolución	1024 x 600
Cuerpo	CPU	Intel Core i7 de 4a generación (16 GB de RAM, 256 GB de SSD)
	Tarjeta GPU integrada	NVIDIA Jetson TK1
	Altavoz	5W
	Batería	Batería de iones de litio 25V / 9.5Ah
	Tiempo de operacion	Alrededor de 3 horas
	Capacidad de expansión	USB3.0 x3, VGA x1, LAN x1, Serie x1, Salida 12V-0.5A x1, TK1 USB2.0 x1, TK1 Serie x1

Estas especificaciones son privadas en [Toy19]

Combinando tres grados de libertad de su base, cuatro de su brazo y uno de su torso, el HSR cuenta con ocho grados de libertad totales, lo cual permite movimientos flexibles para navegación y manipulación de objetos. El HSR también cuenta con diversos sensores, láser para medir distancias, micrófono, cámaras estéreo y de profundidad, entre otros.[YNK+]

3.1.2. Lentes de Realidad Virtual

Se pueden utilizar dispositivos montados en la cabeza para desplegar imágenes, llamados ”Head Mounted Displays”. Estos dispositivos constan de una o dos pantallas y un par de lentes. El campo de visión es expandido para que la imagen aparezca posicionada a muchos metros al frente.[Shi02] Los lentes de realidad virtual son ”Head Mounted Displays”que nos permiten ver un mundo virtual directamente en las pantallas que posee y permite una sensación inmersiva en el entorno virtual.

Figura 3.2: Modelos de HMD[MGMADGM17]

Oculus Rift

Oculus Rift son lentes de realidad virtual desarrollados y manufacturados por Oculus VR. Cuenta con dos pantallas PenTile OLED, con resolución de 1080x1020 por ojo, con una frecuencia de actualización de 90 Hz y un campo de visión de 110°. Cuenta con rastreo rotacional y posicional, así como audifonos integrados.[Wik18b]

Para su uso se requiere un equipo de cómputo compatible y con suficiente capacidad para desplegar las imágenes de manera rápida y eficiente, así como el sistema operativo Windows ya que solo existe soporte oficial para este. Las características de estos lentes de realidad virtual son útiles para la telepresencia inmersiva de este proyecto.

Figura 3.3: Oculus Rift con sensores y Touch Controllers[Unr18]

Samsung Gear VR

Samsung Gear VR son lentes de realidad virtual creados por Samsung Electronics en colaboración con Oculus VR[Wik18c]. Es compatible con algunos modelos de dispositivos móviles de Samsung Electronics, donde la pantalla de estos sirve para desplegar las imágenes y los acelerómetros y giroscopios con los que cuentan, para hacer rastreo de rotación. La característica de estos lentes de funcionar con dispositivos móviles permite que sean utilizados de una manera más sencilla en un ambiente doméstico. Para utilizar Samsung Gear VR se requiere un equipo Samsung con sistema operativo Android.

(a) Vista trasera[Sam18b] (b) Colocación de dispositivo móvil[Kar17]

Figura 3.4: Samsung Gear VR

3.1.3. Dispositivos Móviles

Los dispositivos móviles fueron diseñados para ser computadoras portátiles de mano, que ejecutan una variedad de aplicaciones. Pueden servir a propósitos adicionales como proveer de servicio telefónico[CE15]. Actualmente los teléfonos inteligentes, dispositivos móviles como se describe anteriormente, con servicio telefónico, cuentan con una alta capacidad de procesamiento, además de sensores y herramientas como acelerómetros, giroscopios y cámaras que permiten crear desarrollos más complejos para ellos.

Samsung Galaxy Note 8

El Samsung Galaxy Note 8 modelo SM-N950F es un dispositivo móvil creado por Samsung Electronics. Este cuenta con un procesador Exynos de 8 núcleos, 4x2.3 GHz y 4x1.7 GHz y memoria RAM de 6 Gb, una pantalla Super AMOLED con resolución QHD 2960x1440 con un tamaño de 6.3”(160.5 mm) en diagonal y diversos sensores como acelerómetro y giroscopio, entre otras características, lo cual lo hace óptimo para que, en conjunto con Samsung Gear VR reproduzcan un ambiente virtual de buena calidad. [Sam18a]

Figura 3.5: Samsung Galaxy Note 8[Ama18]

3.1.4. Control Remoto

Al estar en un ambiente virtual es útil contar con un control para navegar e interactuar con éste. El control puede ser con sensores de movimiento para facilitar la interacción en el ambiente o únicamente con botones.

Dualshock 4

Dualshock 4 es la cuarta iteración de una línea de ”gamepads”desarrollados por Sony Interactive Entertainment para la familia PlayStation. Cuenta con diversos botones para la entrada de comandos, además de utilización de bluetooth y usb para conectividad con otros dispositivos distintos a PlayStation. Esta característica permite utilizarlo con PC’s y dispositivos móviles con Android.

Figura 3.6: Control Dualshock 4[Son]

3.2. Software

3.2.1. Middleware para robótica

Crear robots de propósito general realmente robustos es difícil. Los problemas que parecen triviales desde una perspectiva humana a menudo varían demasiado entre diferentes tareas o ambientes. Lidiar con estas variaciones es demasiado complicado para que un solo individuo, laboratorio o institución sea capaz de lograrlo. Debido a esto es importante utilizar un middleware para robótica que permita realizar, utilizar y unificar ciertas tareas.

ROS

ROS (Robot Operating System) es un middleware para robótica, un conjunto de bibliotecas y herramientas de software que ayudan a construir aplicaciones robóticas. Estas herramientas van desde drivers hasta los últimos algoritmos creados. ROS fue diseñado para ser tan distribuido y modular como sea posible, por lo que provee servicios diseñados para clusters heterogéneos de computadoras como abstracción de hardware, control de dispositivos en bajo nivel, implementación de funcionalidades comúnmente usadas, procesos con paso de mensajes y manejo de paquetes. Así los usuarios pueden usar tanto o tan poco de ROS como se desee.

ROS utiliza paquetes que contienen nodos para realizar estas tareas : ”Los paquetes son la unidad de organización de software del código de ROS.

Cada paquete contiene bibliotecas, ejecutables, scripts y otros artefactos"[ROSa]. "Un nodo no es mucho más que un archivo ejecutable dentro de un paquete de ROS. Los nodos de ROS utilizan una biblioteca de cliente de ROS para comunicarse con otros nodos. Los nodos pueden publicar o suscribirse a un *Topic*. Los nodos también pueden proveer o utilizar un servicio"[ROSc]. "Los *Topics* son canales nombrados sobre los cuales los nodos intercambian mensajes. Los *Topics* tienen semánticas anónimas suscriptor/publicador, lo que desacopla la producción de información de su consumo. En general los nodos no están conscientes de con quién se están comunicando. En vez de esto, los nodos están interesados en la información que provee el *Topic* relevante; nodos que generan información, publican al *Topic* relevante. Puede haber múltiples suscriptores y publicadores de un *Topic*d[ROSb]. En el caso de este proyecto se utiliza ROS debido a su modularización y fácil manejo de bibliotecas y paso de mensajes para unir el dispositivo de realidad virtual, el robot virtual y el robot real. Así como la utilización de otras bibliotecas para la navegación. Este proyecto utiliza la versión Kinetic de ROS, ejecutándose en el Sistema Operativo Ubuntu (Linux) 16.04 LTS.

3.2.2. Motor de Videojuegos para Simulación

Un motor de videojuegos es un Software utilizado para la creación de videojuegos. Permite la importación de "assets" o archivos multimedia como son modelos 3D, archivos de audio e imágenes entre otros. Estos archivos pueden ser utilizados en "escenas de juego" donde programadores, diseñadores y artistas pueden integrar estos elementos para que puedan interactuar entre ellos para crear el juego[Unia]. Los motores de juego incluyen elementos para la simulación de física, iluminación, inteligencia artificial, rendering, red, de una manera muy general, así, las personas involucradas en la creación del juego pueden enfocarse en las particularidades de cada juego sin la necesidad de crear los elementos más básicos. Utilizando scripts para la programación de comportamientos dentro del juego, se puede dar forma a los juegos creados.

La utilización de motores de videojuegos permite un desarrollo más rápido, sencillo y enfocado en el contenido más que en otros aspectos de bajo nivel. Los motores de videojuegos han sido utilizados de manera amplia para simulación debido a esta facilidad para crear contenido que permite enfocarse en temas más específicos en el campo de la simulación. La calidad que puede llegar a lograrse por medio de estos motores de juego permite una simulación lo suficientemente

convincente para su utilización en este tipo de situaciones.

Unity 3D

Unity 3D es un motor de videojuegos que permite la creación de estos para diversos sistemas y dispositivos: PC, XBox Family, Play Station Family, consolas Nintendo, Android, iOS, Web. Esta característica, así como el fácil uso de su interfaz permite la creación de contenido una única vez y poder re-utilizarlo en otros sistemas. Unity 3D utiliza como lenguajes base C# y javascript para la creación de scripts que modelen el comportamiento de los elementos de juego, para los modelos 3D es posible utilizar diversos formatos como son .obj, .fbx y .mb. Este proyecto utiliza la versión versión 2018.2.21f1 de Unity 3D, ejecutándose en el Sistema Operativo Windows 10 Profesional.

3.2.3. Modelos 3D

Un modelo es una representación de una entidad concreta o abstracta. Esta representación puede ser de varios tipos. Modelos cuantitativos utilizan ecuaciones para representar y describir el comportamiento del sistema; modelos organizacionales usan jerarquías para representar esquemas de clasificación. Un modelo de computación gráfica se refiere a la representación geométrica de una entidad. El propósito del modelo es permitir a la gente visualizar la estructura de la entidad modelada. Podemos identificar un espacio de tres dimensiones por que vemos nuestro mundo de esta manera. Los objetos no tienen únicamente el largo y altura de un espacio de dos dimensiones, si no también tienen una profundidad asociada a ellos. Así, un modelo 3D se refiere a una representación geométrica de una entidad en un espacio de tres dimensiones.[Sha04]

HSR Toyota Virtual

Para el modelo virtual del robot HSR de Toyota, se utilizó el modelo creado por SIGVerse project organized by Inamura group of NII, en el proyecto Common Unity Project para el Partner Robot Challenge (Virtual Space) de la competencia en el World Robot Competition 2018.[oI17]

(a) Vista frontal (b) Vista a tres cuartos

Figura 3.7: Robot HSR Virtual

3.2.4. Sistema Operativo para Dispositivos Móviles

Un sistema operativo se encarga de manejar el hardware del equipo computacional, así como de manejar los procesos que se ejecutan. Actualmente los sistemas operativos se encuentran en múltiples dispositivos, entre estos se encuentran los dispositivos móviles, como equipos de telefonía, tablets, relojes.[SGG12]

Android

Android es un sistema operativo para dispositivos móviles, desarrollado actualmente por Google. Principalmente desarrollado para equipos de telefonía con pantalla táctil, Android ha expandido mucho sus capacidades hasta llegar a televisiones, automóviles y dispositivos usables como relojes y ropa.[Wik]

3.3. Diseño del sistema

El desarrollo está dividido en cinco etapas:

- **Programación de la interfaz local que se utilizará para la teleoperación:** Se describen los elementos necesarios para poder crear la interfaz de Realidad

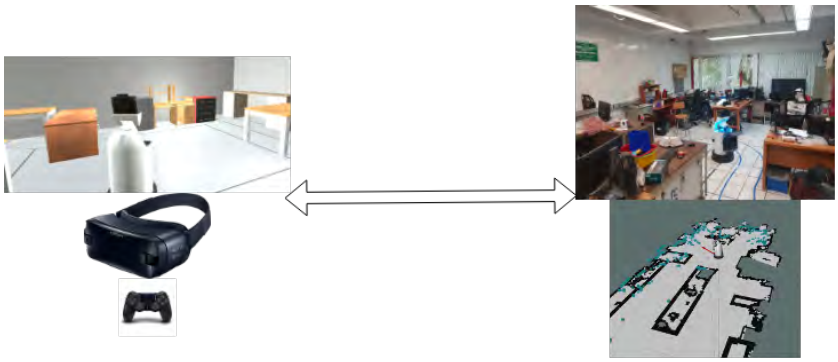


Figura 3.8: Configutación del proyecto

Virtual.

- **Creación del ambiente virtual predefinido sobre el que funcionará la interfaz:** Se describen las herramientas y elementos necesarios para la creación de un mundo virtual que represente un espacio real predefinido.
- **Programación de la interfaz de comunicación entre el robot en el ambiente real remoto y la interfaz virtual local:** Se describe la manera y los elementos necesarios para establecer una comunicación entre el robot en el ambiente real remoto y el ambiente virtual local que permita una teleoperación y retroalimentación más rápida y eficiente.
- **Creación dinámica de objetos en el ambiente virtual para objetos que pueden cambiar de posición en el ambiente real:** Se describen los elementos necesarios para crear una representación de objetos con posiciones no fijas en el mundo real y poder representarlas, así como su posición en el mundo virtual.
- **Filtros y utilización de información de imágenes reales para mejorar la experiencia:** Utilizar la información contenida en las imágenes reales transmitidas, puede ayudar a mejorar la experiencia al incluir estos datos en la representación virtual, y en el caso de visualizar las imágenes reales directamente, se pueden hacer ajustes para dar una mejor percepción al usuario.

Para el desarrollo de este proyecto se pensó en que la estructura final con sus componentes pudieran ser utilizados de manera sencilla y sin complicaciones para un público general. Debido a esto y a los elementos con los que se contaban se decidió utilizar:

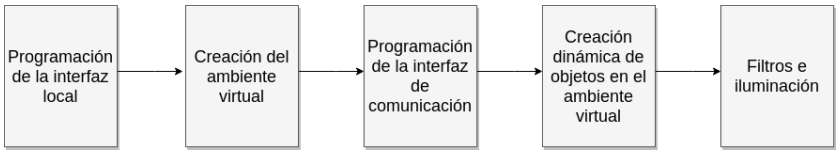


Figura 3.9: Diagrama de implementación

- **Robot HSR de Toyota:** Es un robot comercial que cuenta con una interfaz agradable a la vista humana y sin complicaciones para el usuario.
- **Oculus Rift y Samsung Gear VR:** Estos dispositivos son de un uso muy intuitivo, además de ser compactos. En el caso de Samsung Gear VR, utiliza un dispositivo móvil Android, de uso común hoy en día por lo que su uso es más familiar para el público en general.
- **Dual Shock 4:** Este control de juego permite la conexión a una computadora mediante cable microUSB-USB, y a Android mediante bluetooth de manera sencilla en ambos casos, así, se puede utilizar el mismo control para Oculus Rift y Samsung Gear VR.

En cuanto al software utilizado para el proyecto y el código generado para el mismo, se eligió tomando en cuenta el hardware que se utilizó y los elementos que permiten programarlos:

- **ROS** con lenguaje de programación C++
- **UNITY 3D** con lenguaje de programación C#

3.3.1. Creación del ambiente virtual predefinido sobre el que funcionará la interfaz

La interfaz de usuario es uno de los puntos más importantes en cualquier sistema con interacción humano-computadora. En el caso de los sistemas de telepresencia, es muy importante que el usuario tenga la sensación de encontrarse en el ambiente remoto, que la experiencia sea convincente. Los primeros programas computacionales se ejecutaban de inicio a fin, así, los usuarios y/o programadores debían esperar el término de la ejecución para ver los resultados, además de no poder interactuar con

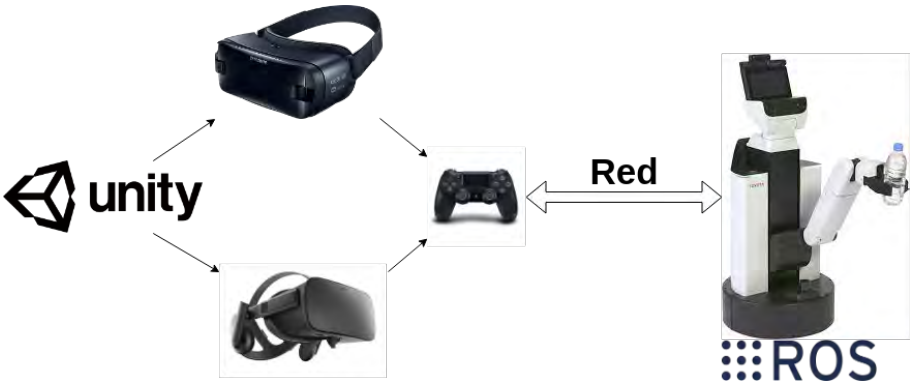


Figura 3.10: Diagrama de interacciones

los programas. Un ambiente virtual debe ser interactivo, el usuario debe ser capaz de controlar o explorar los elementos de manera interactiva y de manera fluida. Una manera de hacer esto es mediante un ciclo infinito en el que se pidan entradas al usuario para ser procesadas y regresar resultados intermedios.

Algorithm 1: Ciclo principal

```
1 Inicialización de variables;
2 while true do
3 |   Lectura de entradas del usuario;
4 |   Procesamiento de datos del ambiente;
5 |   Mostrar resultados;
6 end
```

En el ambiente virtual así como en videojuegos, no son únicamente las entradas del usuario las que se procesan, también se ejecutan otros procesos como la física del ambiente, la posición de los objetos, la inteligencia artificial de ser el caso y el despliegue de gráficos. Si se espera siempre alguna entrada del usuario, no es posible ejecutar lo demás. Para solucionar esto y hacerlo de una manera más simple, cada ciclo se revisa si el usuario ha ejecutado alguna acción, si no es así, se continúa la ejecución de otros procesos. Y para simplificar aún más, se separa el procesamiento lógico de el despliegue de gráficos generalmente en funciones llamadas Update y Draw.

Algorithm 2: Ciclo principal

```
Inicialización de variables;
while true do
    Update;
    Draw;
end
```

Así, cada ciclo se ejecuta la función Update() que se encarga de revisar qué ha cambiado en el ambiente de manera lógica: Entradas del usuario, inteligencia artifical, física, posición de objetos. De igual manera, en cada ciclo, se ejecuta la función Draw() que se encarga de desplegar los objetos en la pantalla. Comúnmente la ejecución de los ciclos se realiza 60 veces por segundo, es decir, se despliegan 60 cuadros por segundo en la pantalla, aunque puede variar y ejecutarse en 30, 120, 144 u otra cantidad de cuadros por segundo, además de que si la ejecución de Update o Draw toma demasiado tiempo, los cuadros por segundo pueden verse reducidos. La interfaz visual para este proyecto es realizada utilizando cascos para realidad virtual, con los que el usuario puede sentirse completamente inmerso en el ambiente remoto, tanto utilizando el video de retroalimentación real, como en el ambiente virtual. Es importante también, que el usuario pueda mover la cabeza para explorar el ambiente.

Otro punto importante para la telepresencia, es el poder controlar más aspectos del dispositivo o robot remoto, como el movimiento en el ambiente. La interfaz utilizada para este propósito en el proyecto es un control remoto para jugar videojuegos, esto debido a la facilidad y comodidad de uso, así como su facilidad de conexión mediante usb y bluetooth. En los ambientes reales existen objetos que no se mueven en el espacio o que es muy difícil que cambien de posición, como son las paredes, el piso, el techo, o algunos muebles como mesas grandes, refrigeradores, escritorios. Para este tipo de objetos, en el ambiente virtual podemos crear una representación con modelos 3D en un lugar fijo, que es donde se encuentran estos objetos ya que no cambiarán de posición.

Para abstraer un espacio real a un espacio virtual es necesario contar con las escalas adecuadas, así, en el momento de navegar con el robot real en el espacio, será sencillo localizarlo dentro del mundo virtual. La abstracción puede realizarse utilizando los sensores del robot para hacer un mapa del espacio en el que navegará el robot, de la misma manera como se hace para que el robot pueda planear

la navegación en el ambiente. Contando con el mapa del lugar, es posible localizar los objetos que no se mueven y colocar una representación virtual de manera fija.

3.3.2. Programación de la interfaz de comunicación entre el robot en el ambiente real remoto y la interfaz virtual local

La comunicación entre la interfaz local y el robot debe ser precisa y rápida, se debe poder comunicar información de posición, localización y control desde y hacia el robot en el ambiente real remoto.

Transmisión y proyección de video a interfaz local

Aun contando con la ayuda del espacio virtual, es importante poder contar con la opción de ver el mundo real tal cual es, para saber exactamente qué es lo que está ocurriendo. Utilizando las cámaras del robot HSR de Toyota, es posible capturar video del ambiente real para posteriormente transmitirlo al ambiente local. Además el robot cuenta con cámaras estéreo, lo cual permite enviar videos de ambas cámaras para lograr una visión estereoscópica en la interfaz local. Utilizando Unity 3D, los lentes de realidad virtual son capaces de reproducir el video del ambiente real en el que se encuentra el robot y de cambiar la vista a ver el ambiente virtual, representación de ese ambiente real.

Comunicación con el ambiente virtual

En el trabajo "A network communication protocol for distributed virtual environments"de G. Drew Kessler y Larry F. Hodges, categorizan las comunicaciones entre ambientes virtuales en tres[KH02] :

- **Mensajes de eventos:** Transmiten información que no puede ser descartada.
- **Mensajes de comandos:** Similares a los mensajes de eventos, pero requieren una respuesta.
- **Mensajes de actualización de estados:** Cualquier transmisión acerca del estado actual del ambiente compartido. Un mensaje de estado se vuelve obsoleto cuando un nuevo mensaje de estado se genera para el mismo conjunto a menos que

un evento o comando haya ocurrido desde la última actualización de estado o el receptor requiera la historia completa de cambios de estado.

Los ambientes virtuales distribuidos generalmente transmiten gran cantidad de mensajes de actualización de estados. Una tarea no puede permitirse perder tiempo recibiendo viejos mensajes[KH02]. En el caso de este trabajo, aun cuando el ambiente virtual es únicamente de un lado de la comunicación, la información del ambiente real es abstraída por el robot y esto permite manejar la información como si ambos ambientes fueran virtuales y de esta manera mejorar y simplificar la comunicación entre la interfaz virtual y el robot.

Modelo OSI

El modelo OSI (Open Systems Interconnection) provee de una base común para la coordinación de desarrollo de normas con el propósito de interconectar sistemas, que permite a estándares existentes ser colocados en perspectiva dentro del modelo de referencia. El modelo OSI se encuentra dividido en siete capas: Fisica, Enlace de datos, Red, Transporte, Sesión, Presentación y Aplicación[Int96]. Dentro del modelo OSI, la capa de transporte se encarga de la transmisión de datos independientemente de la capa física que se use, y el direccionamiento ya está dado por la capa de enlace y red. Se encarga de establecer la conexión entre dos puntos, se puede establecer más de una conexión de la capa de transporte entre los mismos puntos, la calidad de servicio es dependiente de la clase de servicio solicitada. Al no encargarse de controlar y mantener el enlace entre dos equipos como la capa de sesión, nosotros definimos cuando se crea la conexión y qué hacer si se pierde, y, al solo transmitir los datos sin dar semántica o sintaxis, podemos definir que datos serán enviados exactamente y cómo serán interpretados.

Berkeley sockets

Los Berkeley sockets son una interfaz de programación de aplicaciones basados en la capa de transporte del modelo OSI, además[Unib]

- Son una abstracción para que las aplicaciones envíen/reciban mensajes.

Figura 3.11: Capas del modelo OSI[Sol19]

- Las aplicaciones crean sockets que se ”conectan” a la red.
- Las aplicaciones leen/escriben a/de sockets.
- Implementados en kernel.
- Facilita la creación de aplicaciones de red.
- Esconde detalles de protocolos y mecanismos más bajos.
- Disponibles en Windows, Linux y otros sistemas operativos.

- Los sockets de flujo permiten a los procesos comunicarse utilizando el protocolo TCP (Transmission Control Protocol). Este provee un flujo de datos bidireccional, confiable, en secuencia, sin duplicados. Después de que se establece la conexión, la información puede ser leída y escrita de esos sockets como un stream de bytes.
- Los sockets de datagramas permiten a los procesos utilizar UDP (User Datagram Protocol) para comunicarse. Un socket de datagramas soporta un flujo

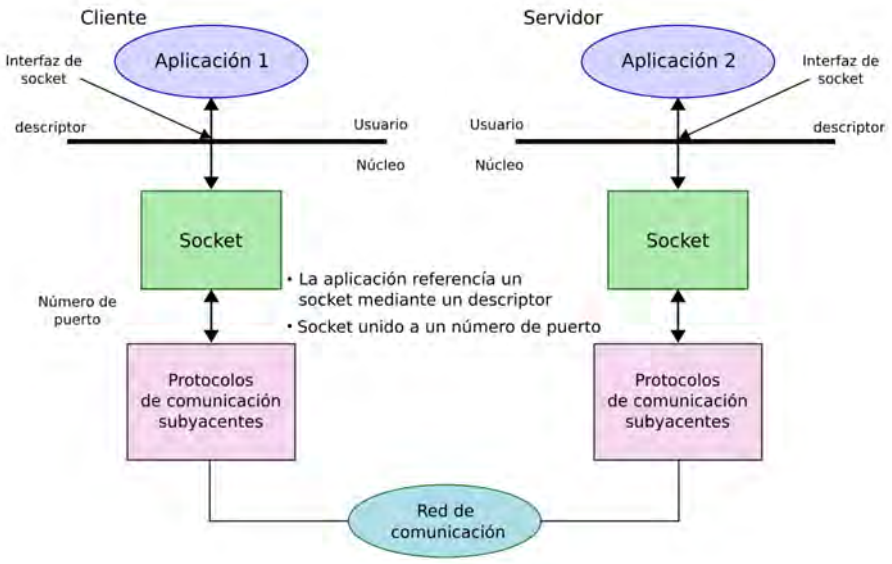


Figura 3.12: Comunicación entre sockets, adaptado de [Unib]

bidireccional de mensajes. Un proceso en un socket de datagrama puede recibir mensajes en un orden diferente de la secuencia enviada, puede recibir mensajes duplicados y puede tener pérdidas de información[Unib].

- Los sockets crudos proveen acceso a protocolos e interfaces internas de la red. Estos solo se encuentran disponibles para el "superusuario".[Fre18]

Al ser independientes de ROS y del Sistema Operativo, los sockets pueden ser utilizados de diversas maneras. En este proyecto, al estar utilizando diversos Sistemas Operativos, bibliotecas y frameworks, ésto viene a ser de gran utilidad.

Ubicación del robot en el mundo real

Para poder localizar al robot en el espacio virtual, es necesario que el robot conozca su ubicación en el ambiente real que se encuentra, una vez obtenida esta información, es necesario comunicarla al ambiente virtual. ROS cuenta con bibliotecas para localización, entre estas amcl, que es un sistema de localización probabilístico para un robot moviéndose en 2D.

3.3.3. Creación dinámica de objetos en el ambiente virtual.

Al momento de abstraer el mundo real a un mundo virtual hay un problema cuando existen objetos que pueden cambiar de posición constantemente como son sillas, botellas, cajas pequeñas, incluso pueden cambiar de posición durante la navegación del robot, y esto se debe tomar en cuenta para la representación virtual. En ROS existen diversas bibliotecas para la detección y ubicación de etiquetas QR en el ambiente del robot. Esto es de utilidad al asociar etiquetas con objetos, conocer la ubicación y enviarla al ambiente virtual para su representación y ubicación en el espacio con modelos prediseñados en el ambiente virtual.

3.3.4. Filtros y utilización de información de imágenes reales para mejorar la experiencia.

El uso de algoritmos para mejorar imágenes y el uso de la información de las mismas, puede resultar útil para mejorar la experiencia durante la navegación con las imágenes reales o en el ambiente virtual.

Procesamiento de imágenes para mejorar la calidad del video

Poder contar con una mejor calidad de imágen al presenciar el ambiente real puede resultar de gran utilidad ya que la percepción mejora y se puede realizar una teleoperación más eficiente. Las técnicas de realce de imágenes son útiles para mejorar la apariencia visual de estas, en cuanto a nitidez, distorsión, contraste, etc. Las técnicas de realce en el dominio espacial utilizan la información de los pixeles de la imágen para realizar la mejora.

"Los filtros espaciales pueden clasificarse basándose en su linealidad: filtros lineales y filtros no lineales. A su vez los filtros lineales pueden clasificarse según las frecuencias que dejen pasar: los filtros paso bajo atenúan o eliminan las componentes de alta frecuencia a la vez que dejan inalteradas las bajas frecuencias; los filtros paso alto atenúan o eliminan las componentes de baja frecuencia con lo que agudizan las componentes de alta frecuencia; los filtros paso banda eliminan regiones elegidas de frecuencias intermedias. La convolución es una operación por la cual se lleva a cabo una acción de filtrado[Dr.06]. El filtro laplaciano es un filtro paso altas

que destaca las regiones con cambios rápidos de intensidad y es utilizado así para la detección de bordes.

El laplaciano L(x,y) de una imágen con valores de intensidad de pixeles I(x,y) esta dado por[FPWW03]:

$$L(x, y) = \frac{\delta^2 I}{\delta x^2} + \frac{\delta^2 I}{\delta y^2} \tag{3.1}$$

Esto puede ser calculado utilizando un filtro de convolución. Ya que la imagen se representa de manera discreta, se utiliza un kernel discreto de convolución.

Dos kernels utilizados comúnmente son:

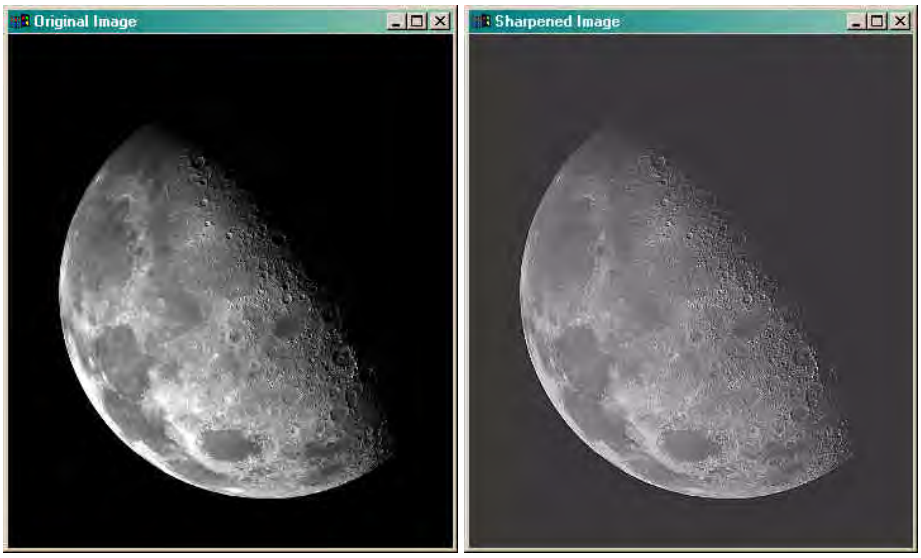
$$\begin{bmatrix} 0 & -1 & 0 \\ -1 & 4 & -1 \\ 0 & -1 & 0 \end{bmatrix} \begin{bmatrix} -1 & -1 & -1 \\ -1 & 8 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Al utilizar el filtro podemos obtener los bordes en la imágen. Para lograr mejorar la imágen debemos sumar el resultado obtenido a la imagen

(a) Original (b) Filtrada

Figura 3.13: Obtención de bordes[Idl]

original para resaltar los bordes y mejorar la nitidez.



(a) Original (b) Imagen mejorada

Figura 3.14: Mejoramiento de imagen[Idl]

Iluminación en ambiente virtual

Al utilizar la teleoperación en un ambiente virtual, es importante que el ambiente representado sea lo más cercano al ambiente real, una de las características que podemos identificar es la iluminación en el espacio. El espacio de color HSV (Hue Saturation Value / Matiz Saturación Valor) es una representación alternativa del modelo RGB utilizado comúnmente para la representación de imágenes. El Matiz de este espacio representa el color, la Saturación representa la intensidad del color y el Valor representa la luminosidad del color. Al ser una representación alternativa del espacio RGB, podemos hacer una conversión directa entre estos dos espacios. Ya que el robot utiliza cámaras para enviar información de video al ambiente virtual, con la representación RGB de estas imágenes, podemos convertir la información al espacio HSV y utilizar el Valor promedio de la imagen para controlar la iluminación en el ambiente virtual.

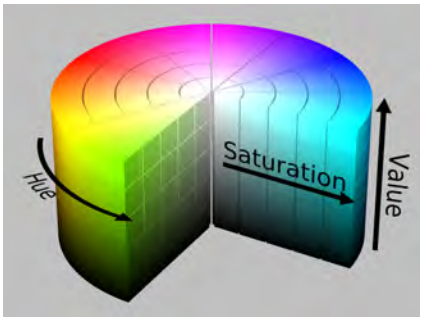


Figura 3.15: Espacio de color HSV[Wik18a]

Capítulo 4

Implementación del sistema

En este capítulo se muestra el proceso de la creación del software necesario para controlar al robot de manera local, el uso del hardware: robot, lentes de realidad virtual y los elementos utilizados para su teleoperación.

El desarrollo está dividido en cinco etapas:

- **Programación de la interfaz que se utilizará para la teleoperación:** Se describe la creación del software de la interfaz de Realidad Virtual.
- **Creación del ambiente virtual predefinido sobre el que funcionará la interfaz:** Se describe la creación de un mundo virtual que represente un espacio real predefinido.
- **Programación de la interfaz de comunicación entre el robot en el ambiente real remoto y la interfaz virtual local:** Se describe la creación del software para establecer una comunicación entre el robot en el ambiente real remoto y el ambiente virtual local que permita una teleoperación y retroalimentación más rápida y eficiente.

- **Creación dinámica de objetos en el ambiente virtual para objetos que pueden cambiar de posición en el ambiente real:** Se describe la creación de software y utilización de modelos 3D que representan los objetos del mundo real en el mundo virtual.
- **Filtros y utilización de información en de imágenes reales para mejorar la experiencia:** Se muestra la manera en la que se logró incluir información de las imágenes reales al ambiente virtual y los filtros para mejorar la percepción en la calidad de imágen.

4.1. Creación del ambiente virtual predefinido sobre el cual funcionará la interfaz

Para la construcción de la interfaz inmersiva y el control de la misma, se utilizó:

- **Oculus Rift:** Casco de realidad virtual desarrollado por Oculus.
- **PC Dell:** Precision Tower 7810
  - **Procesador:** Intel(R) Xeon(R) CPU E5-2650 v4 2.2GHz
  - **RAM:** 16 Gb
  - **Sistema operativo:** Windows 10 64 bit
  - **Tarjeta Gráfica:** Quadro P4000:
    - **Memoria GPU:** 8Gb GDDR5
    - **Nucleos CUDA:** 1792
- **Samsung Gear VR (2017):** Casco de realidad virtual desarrollado por Samsung y Oculus
- **Samsung Gear VR API:** Interfaz de programación de aplicaciones de realidad virtual utilizando dispositivos Samsung
- **Samsung Galaxy Note 8:** Modelo SM-N950F con sistema operativo Android 9
- **Gamepad:** PlayStation, Dualshock 4

Para la creación del mundo virtual se utilizó:

- **Modelos 3D:** Prediseñados, en formato obj y fbx
- **Unity 3D:** Versión 2018.2.21f1: Motor de creación de contenido. (Videojuegos)

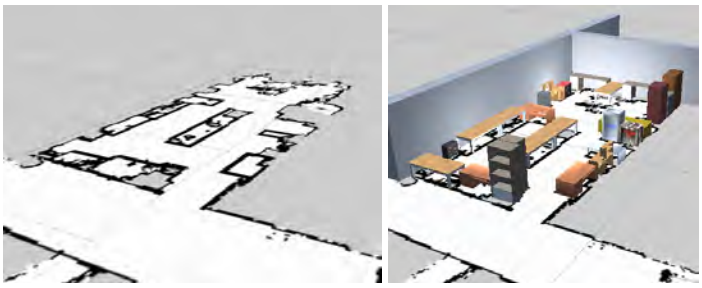
La API de Oculus Rift permite integrar de manera sencilla una interfaz que reconoce el movimiento del casco Oculus y la utilización de los acelerómetros y giroscopios en un celular dentro del Samsung Gear VR. Unity 3D permite crear una interfaz para realidad virtual y que sea funcional para Oculus Rift y Samsung Gear VR. A partir de la versión 2017.4.x de Unity, la integración de la API se encuentra disponible junto con la instalación, únicamente es necesario activarla para ser utilizada. El compilador de Unity 3D permite realizar la compilación tanto para Windows utilizando Oculus Rift como para Android utilizando Samsung Gear VR. Para Samsung Gear VR, es necesario instalar el JDK (Java Development Kit) y contar con Android SDK (Standard Development Kit) para que sea posible la compilación. En el caso de Android SDK es útil instalar Android Studio ya que desde este se pueden actualizar las bibliotecas del Android SDK de manera sencilla. A partir de aquí se agregaron los elementos necesarios para que el mundo virtual represente al mundo real.

El robot HSR de Toyota cuenta con un sistema de láser que permite crear mapas del lugar donde se encuentra, gracias al trabajo realizado por el Laboratorio de Bio-robótica de la UNAM, se cuenta con un algoritmo que permite crear imágenes con un mapa del lugar donde se encuentra el robot, únicamente navegando en el ambiente como se muestra en la figura 4.1. Una vez teniendo esta imágen del

Figura 4.1: Mapa en 2D del Laboratorio de Bio-robótica

ambiente real, se traslada la imagen a Unity 3D en la posición y escala adecuadas, utilizando el modelo virtual del robot como referencia. A partir de esta imagen, se

pueden colocar modelos 3D, los modelos prefabricados, a excepción del robot HSR de toyota, fueron extraidos de https://www.turbosquid.com/ y después colocados en su posición y escala correspondientes sobre el mapa 2D. Los gráficos son básicos para evitar sobrecarga como se muestra en la figura 4.2. Al agregar estructuras complejas, dada la cantidad de elementos, la interfaz puede volverse lenta por lo que se prefiere utilizar modelos 3D sencillos.



(a) Mapa 2D en el ambiente virtual (b) Mapa con modelos 3D

Figura 4.2: Creación del ambiente virtual

4.2. Programación de la interfaz de comunicación entre el robot en el ambiente real remoto y la interfaz virtual local

La decisión de utilizar sockets fue tomada debido a que proveen una manera sencilla y confiable de comunicación entre procesos completamente diferentes y en este proyecto es necesario comunicar ROS con Unity 3D, siendo estos sistemas diferentes, con lenguajes de programación diferentes. En cada terminal, Unity 3D y ROS, se crearon dos archivos de código que crean los sockets de comunicación confiable y no confiable para manejar esta comunicación. La comunicación confiable se refiere a que los mensajes siempre llegan al destino, implementado por medio de sockets TCP, la comunicación podría ser lenta debido a que para asegurar que lleguen, en caso de perdida se pide reenvío de mensajes hasta ser completados, esto lo hace apto para envío de mensajes de comandos. La comunicación no confiable se refiere a que los mensajes podrian no llegar al destinatario, implementado por medio de sockets UDP,

la aun con posible pérdida de información, la comunicación es más rápida, esto lo hace apto para envío de mensajes de actualización de estados. Una vez creados los sockets, se pueden hacer las conexiones para enviar y recibir los mensajes necesarios. La configuración establecida se muestra en la figura 4.3.

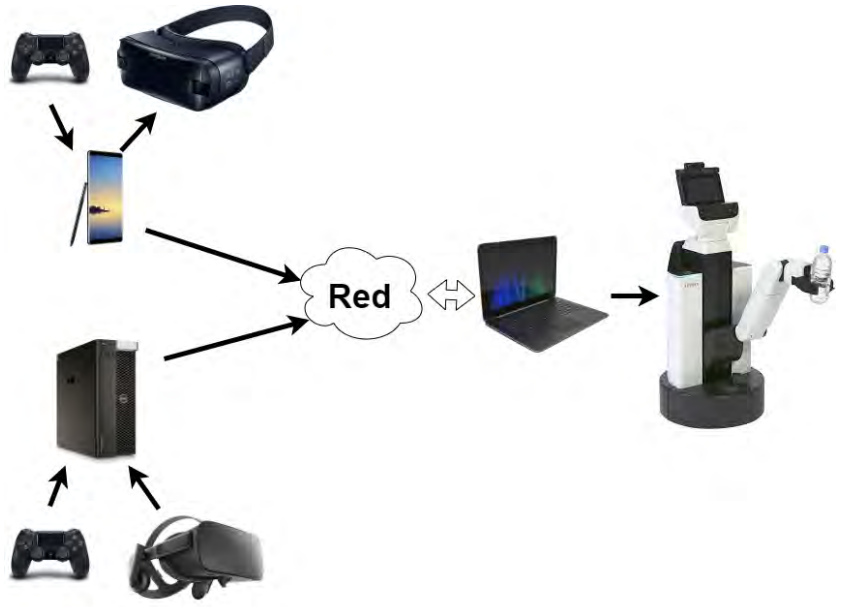


Figura 4.3: Diagrama de conexiones

El equipo con ROS se conecta directamente al robot mediante cable ethernet para tener la mejor comunicación posible entre ellos, a la vez, mediante la tarjeta de red inalámbrica, se conecta a la red wifi y a internet. Esto permite la movilidad del robot sin necesidad de cables. Las comunicaciones entrantes en los puertos correspondientes deben ser redireccionadas a este equipo para asegurar la comunicación adecuada. El equipo con el programa creado en Unity 3D puede ser conectado a la red de manera inalámbrica a la red wifi, con cable ethernet en el caso de la PC o red móvil en el caso de Samsung Gear VR.

La comunicación se establece desde el equipo con el sistema creado en Unity 3D hacia el equipo que está ejecutando ROS. Pueden conectarse diferentes equipos con el sistema de Unity 3D hacia el equipo con ROS y enviar comandos, entre ellos no hay comunicación. El equipo con ROS recibe los datos y los convierte a información que luego se publica en nodos de ROS para la ejecución

por parte del robot, y cuando recibe información de los nodos, la envía mediante los sockets hacia los equipos conectados.

4.2.1. Transmisión y proyección de video a la interfaz local

Una vez creada la interfaz en Unity 3D se procedió a integrar las imágenes recibidas del robot a las cámaras de juego de Unity 3D que despliegan en cada una de las pantallas de los ojos del usuario, cada imágen correspondiente, cámara izquierda del robot con ojo izquierdo del usuario y cámara derecha del robot con ojo derecho del usuario. Para iniciar se creó una conexión con el robot de servicio HSR para que

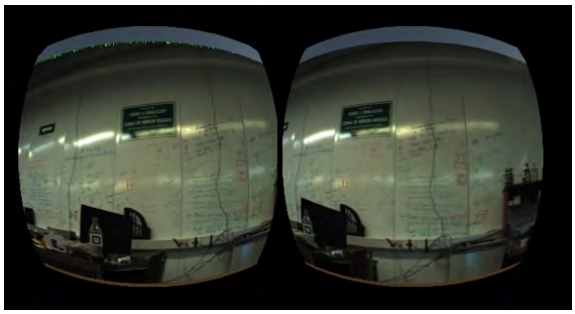


Figura 4.4: Visión estéreo

Unity pudiera recibir el las imágenes enviadas por el robot. Mediante las cámaras frontales y estéreo con las que cuenta el robot, se toma captura de video en formato de imágenes. El robot toma imágenes estáticas de gran tamaño, no video, por lo que esto suponía una gran carga de datos para enviar a través de un flujo por la red, por esta razón se tomó la decisión de utilizar la biblioteca "web video server"[ROSd] de ros, esta permite convertir los nodos de ROS que toman imágenes en un stream de imágenes comprimidas, mjpeg, el cual es un flujo continuo de imágenes comprimidas en formato jpeg una tras otra, sin otro tipo de encabezados o pies del mensaje. En Unity se recibe este stream de imágenes de manera asíncrona para no causar un lento procesamiento de las imágenes (virtuales o reales) que se muestran en pantalla.

Algorithm 3: Recepción de flujo de imágenes

```
while Se recibe un flujo de datos do
    Espera por encabezado de imágen jpeg, representado por el byte 0xd8;
    Guardar flujo en arreglo hasta encontrar pie de imágen;
    Encuentra pie de imágen jpeg, representado por byte 0xd9;
    Cargar imagen a una textura de Unity para poder utilizar con objetos en el
     ambiente.;
    Asigna la textura creada en un objeto de unity dentro del espacio de juego
     para mostrar en pantalla.;
end
```

Estas imágenes se asignan cada que son recibidas a las texturas que se muestran a cada ojo del usuario, esto crea la animación (video) del ambiente real. Recibimos el la información de video del robot para ambas cámaras estéreo con las que cuenta el robot, y de esta manera se presenta al usuario con un efecto de tres dimensiones, una imágen para cada ojo utilizando la interfaz de realidad virtual Oculus Rift o Samsung Gear VR.

Nota: En la versión de la API para Android 9, las comunicaciones requieren una conexión segura, https, las imágenes del proyecto son enviadas mediante http por lo que no funciona en esta versión. Dado que Samsung Galaxy Note 8 cuenta con esta versión, la solución, debido a limitantes de tiempo, fue compilar el proyecto con la versión máxima compatible Android 8.1, esto permite que las comunicaciones se realicen mediante http incluso al ejecutar en Android 9.

4.2.2. Comunicación con el ambiente virtual

En este trabajo, la posición del robot en el ambiente real es abstraída para comunicar al ambiente virtual. Las acciones tomadas en el ambiente virtual se envían al robot en el ambiente real. Debido a esta abstracción, podemos tomar ambos ambientes como si fueran virtuales. Como se mencionó en el capítulo 3, existen tres tipos de comunicación entre ambientes virtuales: Mensajes de eventos, mensajes de comandos y mensajes de actualización de estados.

La comunicación se realizó con sockets de la siguiente manera:

- **Mensajes de eventos - Sockets de flujo (TCP)**

Para acciones en las que se requiere que el mensaje se transmita con confianza, son poco frecuentes y no requieren respuesta.

- **Mensajes de comandos - Sockets de flujo (TCP)**

  Para acciones en las que se requiere que el mensaje se transmita con confianza y son poco frecuentes, ejemplos: Activación de apertura/cierre de mano del robot, El robot toma un objeto.

- **Mensajes de actualización de estados - Socket de Datagrama (UDP)**

  Para conocer el estado actual del ambiente compartido y poder sincronizarlo. En este caso, el ambiente real es el único que se puede modificar directamente, el ambiente virtual debe sincronizar su estado con éste. También se utiliza este tipo de sockets para transmitir algunos comandos de movimiento al robot que pueden variar en gran medida y que su transmisión confiable puede provocar retrasos, ejemplos: Cambio de velocidad del robot, movimiento de la cabeza.

Mensajes de comandos.

En ROS se creó como parte de un nodo para que fuera fácil de integrar con todas las bibliotecas propias de ROS y poder utilizar todas las funciones del robot desde ese código.

Algorithm 4: Utilización de sockets TCP

```
Se crea socket Servidor TCP para recepción y envío de información;
De manera asíncrona se espera para recepción de datos;
while true do
    Envío de datos;
end
```

Mensajes de actualización de estados.

En ROS se creó como parte del mismo nodo que la comunicación Confiable para que fuera fácil de integrar con todas las bibliotecas propias de ROS y poder utilizar todas las funciones del robot desde ese código.

Algorithm 5: Utilización de sockets UDP

```
Se crea socket Servidor UDP para recepción y envío de información;
De manera asíncrona se espera para recepción de datos;
while true do
    Envío de datos;
end
```

Movimiento de la cabeza

Al tener la representación del mundo real a través de la retroalimentación de video, y al utilizar una interfaz inmersiva, el usuario esperaría a que al mover su cabeza pudiera ver otra parte del entorno correspondiente con su movimiento, entonces es necesario enviar la información de este movimiento hacia ROS para que el robot realice el movimiento correspondiente. Con esta información, se utilizaron mensajes de actualización de estados para enviar la información. En ROS se recibe esta información, se convierte a movimiento del robot y se manda a ejecutar el movimiento. Esta situación causaba inconvenientes ya que al ser muy precisa la detección de movimiento, cualquier cambio por mínimo que fuera por parte del usuario generaba un movimiento por parte del robot, esto se soluciona haciendo un umbral en el que si el movimiento era mínimo, no se enviaba la información de movimiento.

En ROS, previo a recibir esta información, se requiere dar de alta un *publisher* que reenviará la información al robot para que el movimiento de la cabeza sea ejecutado. Al recibir información del movimiento de la cabeza se publica como mensaje de ROS, así, el robot ejecutará el movimiento de la cabeza correspondiente al movimiento realizado con el casco de realidad virtual.

Movimiento del robot

Para controlar el robot se utilizó un control Dual Shock 4. La manera de conectarlo a una PC es directamente con un cable USB-microUSB. Para conectarlo al Samsung Galaxy Note 8, se puede realizar mediante bluetooth o utilizando USB OTG y un cable USB-microUSB. En Unity es necesario hacer un mapeo de botones del control con las diferentes entradas al mundo virtual, este mapeo se realiza en el editor de Unity y varia un poco entre la versión de PC y la versión de Android. En unity podemos leer

(a) Conexión bluetooth con Dispositivo Móvil

(b) Conexión USB con PC

Figura 4.5: Conexiones del Dualshock 4

las entradas del control mediante código, ésto se realiza cada ciclo Update de Unity, en el caso de las palancas del DualShock 4, el valor obtenido es un float entre -1 y 1 en el eje X y en el eje Y, esto permite saber que tanto se mueve en cada dirección, así, es posible controlar la velocidad del robot utilizando la presión sobre las palancas. Una vez que contamos con la entrada del control, podemos enviarla a través de mensajes de actualización de estados en el caso del movimiento del *stick* en el control, esto debido a que cada que se ejecuta *Update* puede cambiar el valor ingresado y se desea la información más actual; y mensajes de eventos o comandos en el caso de botones, debido a que al presionar un botón, deseamos que se ejecute una sola acción y que esta se ejecute de manera confiable.

Nota: Después de actualizar Android, se presentaron problemas con la lectura de comandos en las palancas DualShock 4, no se realizaba la lectura adecuada de las entradas de comandos y en ocasiones absolutamente ninguna entrada, en el caso de los botones, no se presento ningún problema. Debido a esto, se optó por utilizar un adaptador USB OTG (On The Go), para conectar el control a Gear VR utilizando un cable, esto soluciona el problema. Nuevamente con una actualización de Android, el *stick* análogo del Dualshock 4 presenta fallas de lectura vía bluetooth o USB OTG. Estas fallas se presentaron después de realizarse las pruebas de movimiento, al revisar a detalle, se concluye que el mapeo de entradas para las palancas en Samsung Gear VR ya no difiere del mapeo utilizado para PC, por lo que utilizar el mismo mapeo soluciona el problema.

4.2.3. Ubicación del robot en mundo real

En el mundo virtual es necesario que el espacio en el que se encuentra el robot corresponda con el espacio en el que se encuentra el robot real. Esto se realizó utilizando los nodos de localización del robot, estos permiten conocer la localización y posición del robot respecto a su entorno real, de esta manera podemos ubicarlo en el entorno virtual. ROS utiliza marcos de coordenadas estandarizados para que la información sea utilizada de la misma manera por diferentes desarrolladores, en nuestro caso se utiliza la medida de distancia, la cual está dada en metros. Tf2 es una biblioteca de ROS que permite utilizar múltiples marcos de coordenadas a través del tiempo, y mantiene la relación entre ellos en una estructura de árbol que mantiene la relación de posición y orientación entre ellos durante el tiempo, para las diferentes partes del robot y en relación también con el mapa. El robot HSR de Toyota cuenta con el siguiente árbol de marcos de coordenadas en la parte superior relacionada con el mapa y la base del robot. La biblioteca amcl de ROS implementa

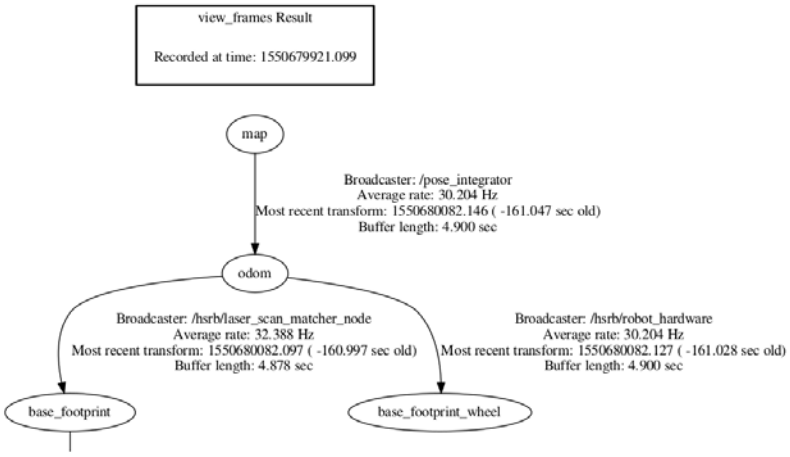


Figura 4.6: Árbol de marcos de coordenadas de HSR Toyota

una aproximación adaptativa para la localización del robot. Los algoritmos utilizados por amcl y sus parámetros son descritos de manera adecuada en el libro *Probabilistic Robotics* de Thurn, Burgard y Fox, en particular se utilizan los siguientes algoritmos de ese libro: sample_motion_model_odometry, beam_range_finder_model, likelihood_field_range_finder_model, Augmented_MCL, and KLD_Sampling_MCL. Al ejecutar este nodo, se publica la ubicación del robot en el mapa y puede ser utilizada

para nuestros propósitos. Ya que el robot conoce su posición en el ambiente, se utiliza la función *lookupTransform* la cual recibe como parámetros los nombres de dos marcos de coordenadas para conocer la posición de uno con respecto al otro, una variable de tiempo para pedir las coordenadas en algún momento específico y una variable *StampedTransform* donde se colocarán los datos.

En este proyecto se solicita la posición de la base del robot con respecto al mapa, y en la última actualización disponible. Una vez obtenida la posición, esta es enviada al ambiente virtual por medio de los sockets, y ya que esta información cambia constantemente y solo nos interesa la más actual disponible, se utilizan mensajes de actualización de estados. En el ambiente virtual se recibe la información y se asigna al modelo del robot para posicionarlo en el lugar adecuado.

4.3. Creación dinámica de objetos en el ambiente virtual.

Para la creación dinámica de objetos, se utilizó:

- **Hardware:**
  - Cámara estéreo del HSR de Toyota
- **Software:**
  - Biblioteca ”ar track alvar” de ROS

La biblioteca ”ar track alvar” nos permite crear etiquetas ”QR code” o Quick Response code que son códigos de barra en forma de matriz, además de poder identificar estas mismas etiquetas y ubicarlas en el espacio utilizando las cámaras del robot. Utilizando estas etiquetas creadas, se asignaron a distintos obje-

Figura 4.7: Ejemplo de códigos QR de biblioteca ”ar track alvar”[Ros18]

tos para poder ubicarlos en el espacio del robot y saber qué objeto es. Después se

envían mensajes de actualización de estados con la información de el tipo de objeto y la posición en el espacio de este hacia la representación virtual remota. Al recibir la información en el ambiente virtual, se coloca un modelo prediseñado con las características del objeto que se le asignó la etiqueta específica y se coloca en la misma posición que en el ambiente real. De esta manera, incluso navegando únicamente en

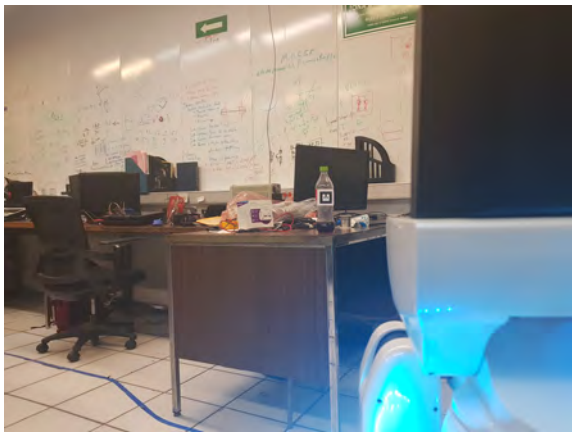
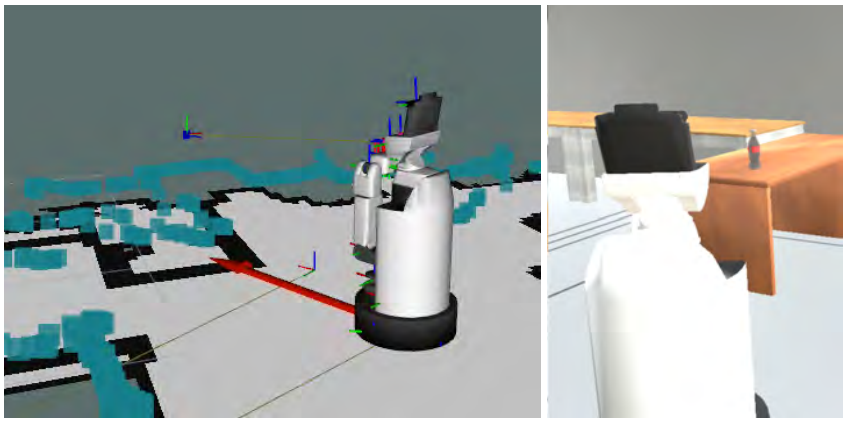


Figura 4.8: Botella con código QR

el ambiente virtual, podemos saber qué objeto ve el robot de acuerdo a la etiqueta y la posición de los objetos que podrían cambiar de posición durante la navegación y teleoperación. Para lograr obtener esta información en el ambiente virtual se utilizaron mensajes de actualización de estados, al conocer la posición de los objetos, se envía de esta manera para no saturar la red con actualizaciones constantes y que el ambiente virtual siempre obtenga la última posición conocida de los objetos que encontró el robot.

Al recibir la información en Unity, se actualiza la posición de los objetos a la percibida por el robot. Trabajo futuro es requerido para identificar objetos, ubicarlos en el espacio y crear un modelo virtual que lo represente sin necesidad de conocerlos previamente ni de utilizar etiquetas QR.



(a) Representación del código QR para el robot

(b) Representación de la botella en mundo virtual

Figura 4.9: Envío de datos abstractos al ambiente virtual

4.4. Filtros y utilización de información de imágenes reales para mejorar experiencia.

Al contar con imágenes del ambiente real, podemos obtener información adicional y mejorar la experiencia. Unity 3D es un motor de videojuegos que utiliza el lenguaje de programación C# principalmente y nos permite utilizar bibliotecas de éste, además de tener la posibilidad de implementar nuevos algoritmos.

4.4.1. Procesamiento en paralelo, Unity 3D

Hacer procesamiento de imágenes puede convertirse en una tarea que requiere demasiado tiempo para ser completada, en especial cuando las imágenes a ser procesadas son muy grandes. Hacer un procesamiento con estas características puede crear una reducción en la fluidez con la que se genera la experiencia, visualización de video real o espacio virtual, por lo que es útil realizar el procesamiento de las imágenes de manera paralela para evitar detener la interfaz y afectar la experiencia. Para realizar esto en Unity 3D se debe crear una tarea asíncrona y preguntar si ya ha terminado constantemente ya que no es posible avisar desde la tarea al hilo principal. Una vez creada esta tarea asíncrona, es posible que espere a que termine algún otro proceso. En

ciclo del hilo principal preguntamos si existe alguna tarea asíncrona y si ha terminado.

4.4.2. Procesamiento de imágenes para mejorar calidad de video

Las imágenes del ambiente remoto real se reciben como un flujo de imágenes jpeg, y son colocadas en un objeto de tipo Texture2D y de éste se puede obtener una matriz de colores a la que se le puede aplicar una convolución para realizar el filtrado y mejora de la imágen.

Se utilizó un filtro de convolución paso-altas laplaciano de 3x3, la matriz utilizada fue:

$$\begin{bmatrix} -1 & -1 & -1 \\ -1 & 9 & -1 \\ -1 & -1 & -1 \end{bmatrix}$$

Todo este proceso se realiza de manera paralela en una tarea asíncrona. Al momento de terminar con el filtro, el ciclo del hilo principal lo detecta y la nueva textura con la imagen mejorada es aplicada a los objetos que muestran el video a través de Oculus Rift y comienza el filtrado de la última imágen recibida, esto causa pérdida de algunos frames, pero se mantiene la imágen más actual posible.

4.4.3. Iluminación en el ambiente virtual

Unity 3D nos permite crear luces de ambiente con niveles de luminosidad dentro del ambiente virtual, estos niveles pueden ser ajustados en tiempo real, por lo que si se cuenta con la información de luminosidad del ambiente real, estos valores pueden ser ajustados para acercarse lo más posible a la iluminación del ambiente real y crear una mejor experiencia. Al contar con las imágenes del mundo real como matrices de colores, podemos recorrerla y obtener su información. La representación RGB de cada píxel en una imagen es convertida al espacio de color HSV. De aquí podemos obtener la información de Valor o luminosidad de cada pixel, posteriormente se saca un promedio con la luminosidad de todos los pixeles en la imágen, de esta manera se obtiene la luminosidad aproximada en el ambiente real y se ajustan los valores en las luces del ambiente virtual. El nivel de luminosidad en el ambiente virtual se ajusta con un valor base, más el promedio de luminosidad en la imágen real, esto con la

finalidad de que aunque las cámaras no sean capaces de ver o el ambiente real tenga muy poca luminosidad, el ambiente virtual siga siendo visible.

Capítulo 5

Pruebas y resultados

En éste capítulo se describen las pruebas realizadas con el robot y el ambiente virtual utilizando Oculus Rift y Samsung Gear VR.

Se realizaron pruebas de:

- Latencia para el movimiento de la base del robot y para el movimiento de la cabeza del robot, de manera local, a distancia real y mediante la utilización de Red Privada Virtual.
- Latencia aproximada para la transmisión de video implementada.
- Cuadros por segundo que mantiene la implementación usada para transmitir video.
- Pruebas de localización del robot y objetos en el ambiente.

5.1. Latencia y cuadros por segundo en transmisión de video y ambiente virtual

Para tener un punto de comparación en el propio sistema se realizaron pruebas de latencia y de cuadros por segundo que se pueden desplegar mediante el envío de un flujo de imágenes (mjpeg). Las pruebas se realizaron en una red local y mediante la utilización de red privada virtual. Una red privada virtual extiende una red privada a través de una red pública como internet y permite enviar y recibir datos como si los dispositivos se encontraran en esa red. Las pruebas que se realizaron con red privada virtual fueron desde una red privada en Tampa/Estados Unidos, Singapur y Sudáfrica y utilizando el servicio de Express VPN. Los datos viajaron desde el laboratorio de Bio-Robótica en Ciudad Universitaria, UNAM hacia la red privada y de regreso hacia la computadora que controla el robot, y los datos de respuesta de regreso por la misma ruta. En estas pruebas se utilizó Oculus Rift y el equipo computacional utilizado para desarrollar el proyecto. Para medir la latencia en el video se utilizó un cronómetro iniciando el contador en la cámara del robot y deteniendo el mismo cuando se registra el inicio en la recepción del video. Para medir los cuadros por segundo, dentro del código de Unity 3D se tiene un contador cada que se recibe una imágen nueva completa, se suman y se divide entre el tiempo.

Pruebas de latencia para el video en milisegundos:

Lugar	Red Local	RPV Tampa	RPV Singapur	RPV Sudáfrica
Prueba 1	520 ms	1450 ms	1710 ms	4780 ms
Prueba 2	520 ms	1380 ms	1580 ms	3600 ms
Prueba 3	850 ms	1380 ms	1640 ms	3670 ms
Prueba 4	390 ms	1180 ms	1450 ms	3600 ms
Prueba 5	1180 ms	1710 ms	1580 ms	6690 ms
Prueba 6	850 ms	2950 ms	1770 ms	2620 ms
Prueba 7	720 ms	1450 ms	1450 ms	6090 ms
Prueba 8	850 ms	1180 ms	1710 ms	8500 ms
Prueba 9	720 ms	1840 ms	1580 ms	5230 ms
Prueba 10	1120 ms	2630 ms	1510 ms	4710 ms
Prueba 11	520 ms	3090 ms	1640 ms	1780 ms
Prueba 12	650 ms	2370 ms	1580 ms	1200 ms
Prueba 13	720 ms	1500 ms	1640 ms	7070 ms
Prueba 14	1180 ms	2290 ms	1780 ms	8840 ms
Prueba 15	980 ms	1720 ms	1580 ms	3020 ms
Prueba 16	650 ms	1900 ms	1700 ms	3540 ms
Prueba 17	720 ms	1640 ms	1570 ms	3730 ms
Prueba 18	1320 ms	2100 ms	1590 ms	5960 ms
Prueba 19	1310 ms	1580 ms	1900 ms	7850 ms
Prueba 20	1110 ms	980 ms	1520 ms	11260 ms
Promedio	844 ms	1816 ms	1624 ms	5187 ms

En el caso de los cuadros por segundo logrados en la transmisión de video se tomo una gran cantidad de muestras y se realizó un promedio:

Lugar	Red Local	RPV Tampa	RPV Singapur	RPV Sudáfrica
Promedio de cuadros por segundo	35.08	35.79	34.56	22.92

De las pruebas realizadas podemos ver que dada la gran cantidad de información que se requiere enviar para la transmisión de video y de acuerdo a la distancia de transmisión, aunque se mantiene aproximadamente la misma cantidad de cuadros por segundo, la latencia se eleva en gran medida. Durante las pruebas realizadas se midieron también los cuadros por segundo del ambiente virtual, el cual fue en promedio de 316 cuadros por segundo.

5.2. Latencia: Base y cabeza del robot

Para poder medir la eficacia del sistema de manera objetiva se realizaron pruebas de latencia en el tiempo de respuesta del movimiento de la base del robot y el movimiento de la cabeza del robot. Dentro del código hecho en Unity 3D se creó un cronómetro de la clase Stopwatch de C#. Al envío de datos se inicia el cronómetro, en ROS, al momento de recibir datos se regresa una señal de regreso para indicar la recepción de datos correcta en el caso de la cabeza y cuando se registra el movimiento del robot en el caso de la base. Para la base se está midiendo el tiempo total de respuesta de la red, del robot y la actualización respectiva de la nueva posición del robot, para la cabeza se está midiendo únicamente el tiempo de la red con la cantidad de datos necesaria para poder enviar los comandos necesarios al robot. Las pruebas se realizaron de igual manera que para medir latencia y cuadros por segundo en el video: en una red local y red privada virtual en Tampa /Estados Unidos, Singapur y Sudáfrica. Adicional a esto, se realizaron pruebas a una distancia real de 3.5 Km aproximadamente desde una red doméstica fuera del campus. En estas pruebas se utilizó Oculus Rift y el equipo computacional utilizado para desarrollar el proyecto.

Pruebas de latencia para el movimiento de la base del robot en milisegundos:

Lugar	Red Local	Red Doméstica	RPV Tampa	RPV Singapur	RPV Sudáfrica
Prueba 1	138 ms	184 ms	271 ms	731 ms	705 ms
Prueba 2	145 ms	208 ms	270 ms	726 ms	771 ms
Prueba 3	693 ms	2066 ms	98 ms	690 ms	737 ms
Prueba 4	171 ms	187 ms	368 ms	678 ms	734 ms
Prueba 5	178 ms	204 ms	320 ms	299 ms	704 ms
Prueba 6	134 ms	194 ms	274 ms	717 ms	414 ms
Prueba 7	163 ms	221 ms	146 ms	741 ms	742 ms
Prueba 8	123 ms	58 ms	193 ms	707 ms	728 ms
Prueba 9	184 ms	154 ms	408 ms	684 ms	105 ms
Prueba 10	968 ms	214 ms	255 ms	651 ms	723 ms
Prueba 11	138 ms	27 ms	312 ms	697 ms	726 ms
Prueba 12	151 ms	160 ms	1314 ms	1034 ms	744 ms
Prueba 13	99 ms	174 ms	333 ms	797 ms	661 ms
Prueba 14	150 ms	144 ms	359 ms	927 ms	705 ms
Prueba 15	204 ms	136 ms	287 ms	687 ms	761 ms
Prueba 16	200 ms	177 ms	284 ms	677 ms	784 ms
Prueba 17	126 ms	184 ms	301 ms	641 ms	896 ms
Prueba 18	248 ms	161 ms	315 ms	698 ms	765 ms
Prueba 19	134 ms	168 ms	237 ms	717 ms	724 ms
Prueba 20	686 ms	2019 ms	308 ms	667 ms	761 ms
Promedio	251.65 ms	352 ms	332.65 ms	708.3 ms	694.5 ms

Gear VR

La información es enviada y recibida utilizando Samsung Gear VR.

Pruebas de latencia para el movimiento de la cabeza del robot en milisegundos:

Lugar	Red Local	Red Doméstica	RPV Tampa	RPV Singapur	RPV Sudáfrica
Prueba 1	405 ms	34 ms	130 ms	532 ms	592 ms
Prueba 2	10 ms	14 ms	134 ms	533 ms	588 ms
Prueba 3	4 ms	5 ms	135 ms	530 ms	593 ms
Prueba 4	50 ms	124 ms	133 ms	538 ms	604 ms
Prueba 5	4 ms	3 ms	134 ms	531 ms	590 ms
Prueba 6	237 ms	39 ms	136 ms	532 ms	589 ms
Prueba 7	3 ms	11 ms	134 ms	529 ms	594 ms
Prueba 8	3 ms	14 ms	30 ms	531 ms	591 ms
Prueba 9	175 ms	37 ms	240 ms	531 ms	593 ms
Prueba 10	3 ms	184 ms	151 ms	582 ms	590 ms
Prueba 11	5 ms	5 ms	134 ms	532 ms	588 ms
Prueba 12	4 ms	6 ms	134 ms	666 ms	597 ms
Prueba 13	274 ms	7 ms	134 ms	534 ms	589 ms
Prueba 14	3 ms	46 ms	133 ms	521 ms	636 ms
Prueba 15	8 ms	98 ms	30 ms	530 ms	588 ms
Prueba 16	19 ms	25 ms	185 ms	532 ms	594 ms
Prueba 17	3 ms	44 ms	133 ms	532 ms	614 ms
Prueba 18	5 ms	7 ms	134 ms	531 ms	590 ms
Prueba 19	503 ms	21 ms	134 ms	531 ms	622 ms
Prueba 20	5 ms	7 ms	133 ms	616 ms	591 ms
Promedio	86.15 ms	36.55 ms	132.05 ms	544.7 ms	596.65 ms

En el trabajo "Response time in man-computer conversational transactions" [Mil68] de Robert M. Miller se muestra que un tiempo de respuesta de dos segundos es un requerimiento universal para que una interacción humano-computadora sea tolerada y percibida como interactiva, y en algunos casos, dependiendo de la acción realizada, de hasta medio segundo. De las pruebas realizadas podemos ver que los tiempos de respuesta son en todos los casos menores a 1 segundo, mucho menores a los tiempos de transmisión de video en las pruebas realizadas y menores o similares a los encontrados en la literatura, además de ser lo suficientemente rápidos para ser percibida como interactiva, además de contar con una definición adecuada para la interacción. El envío de datos se hace de la manera correcta, se recibe la rotación del casco de acuerdo a las configuraciones(posición, centro) iniciales del oculus con rotación 0 de acuerdo a las configuraciones iniciales durante la instalación

de Oculus Rift.

La posición inicial para la rotación de la cabeza del robot está dada por la configuración inicial hecha en la instalación de Oculus Rift. La traslación en el mundo utiliza en parte la posición real tomada por los sensores de Oculus Rift, por lo que, la posición sufre un desfase con respecto al robot. Para solucionar esto, si es que se desea mantener la posición con respecto al robot, basta con cubrir los sensores de Oculus previo a la ejecución del programa.

(a) Orientación del Oculus Rift (b) Orientación de la cabeza del robot

Figura 5.1: Movimiento de la cabeza utilizando Oculus Rift

Cabe resaltar que las pruebas realizadas fueron en ambiente real no controlado (Internet) en el caso de la red doméstica y la red privada virtual, es decir, en un ambiente real, este método muestra resultados satisfactorios. La velocidad de reacción del robot y de la transmisión de comandos se ve claramente disminuida por la distancia a la que se realizan las pruebas, sin exceder 1 segundo.

Gear VR

La rotación e inclinación del Samsung Gear VR corresponden con las realizadas por el robot.

Oculus / Gear VR

Durante el desarrollo se encontraron problemas en la conexión debido a que en ocasiones la red realizaba un cambio de dirección IP en el equipo receptor lo que causaba que los mensajes no fueran entregados de manera adecuada. Otro problema era que

(a) Orientación neutral (b) Giro a la derecha

Figura 5.2: Movimiento de la cabeza utilizando Samsung Gear VR

el cable de la conexión entre el robot y PC (que controla al robot) al ser desconectado, cambiaba la dirección IP lo que ocasiona que el robot no responda a los comandos enviados por el equipo (Que controla al robot), el problema se soluciona cambiando dirección ip en codigo de Unity en el caso de cambio de ip por la red y reiniciando el robot HSR en el caso de cambio de ip por desconexión de cable.

5.3. Interfaz local

5.3.1. Visualización de mundo real y virtual

Oculus

La visualización del mundo real en el equipo donde se creo el sistema (**PC Dell:** Precision Tower 7810) se realiza a una velocidad de 350 FPS de ejecución, en cuanto a la frecuencia de cambio de la reproducción del video se dió de la siguiente manera.

Lugar	Red Local	Red Doméstica	RPV Tampa	RPV Singapur	RPV Sudáfrica
FPS promedio	35.08	35.33	35.79	34.56	22.92

Los FPS se ven limitados en este caso, además de la velocidad de la red, por la conversión del video de imágenes ”crudas” a flujo de imágenes jpeg (mjpeg). Este conteo de FPS es para el video que se ve en cada pantalla del Oculus, es decir, para cada ojo. Al ser una imágen estéreo, al ser observado por un humano, se

tiene una sensación de profundidad. En cuanto a la reproducción del mundo virtual, con el equipo (**PC Dell:** Precision Tower 7810) se ejecuta a una velocidad de 300 FPS.

La prueba se realizó también en el equipo:

HP OMEN (laptop): 15-dh0005la

- **Procesador:** Intel® Core i7-9750H 2.6GHz
- **RAM:** 16 Gb SDRAM DDR4-2666 (2 x 8 GB)
- **Sistema operativo:** Windows 10 Home 64 bit
- **Tarjeta Gráfica:** NVIDIA® GeForce RTX 2070 Max-Q:
  - **Memoria GPU:** GDDR6 de 8 GB dedicada
  - **Nucleos CUDA:** 2304

La velocidad de ejecución para este equipo fue de 300 FPS de ejecución del sistema, para la reproducción del video, se mantiene con los datos del equipo anterior. La velocidad de ejecución para la reproducción del mundo virtual para este equipo fue de 250 FPS.

Gear VR

El video es recibido de igual manera que con Oculus Rift.

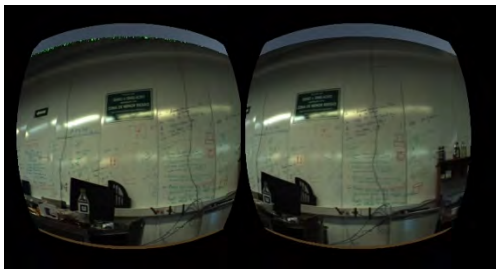


Figura 5.3: Vista del mundo real en Samsung Gear VR

En cuanto al mundo virtual, los gráficos y la definición se ven disminuidos para Samsung Gear VR, existe una notoria disminución de poligonos desplegados, sin embargo, la velocidad de reproducción del video y los FPS no se ven disminuidos en gran medida.

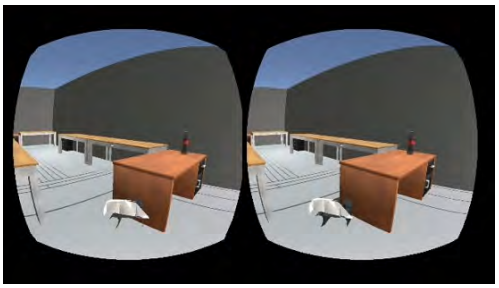


Figura 5.4: Vista del mundo virtual en Samsung Gear VR

5.3.2. Localización y posicionamiento adecuado del robot en el ambiente virtual

El robot se ubica correctamente en el laboratorio utilizando amcl, el envío de datos de posición y rotación del robot son enviados de manera correcta y el robot virtual se ubica en la posición correspondiente del ambiente virtual.

(a) Robot en el ambiente real (b) Localización en RVIZ utilizando amcl

Figura 5.5: Localización del robot

Se hicieron pruebas además en un ambiente real sin iluminación. El robot se ubica de correctamente gracias a que cuenta con ayuda de láser para su localización. Esto permite que aun siendo imposible la utilización de cámaras de video para la navegación, ésta aún se pueda lograr en el ambiente virtual.

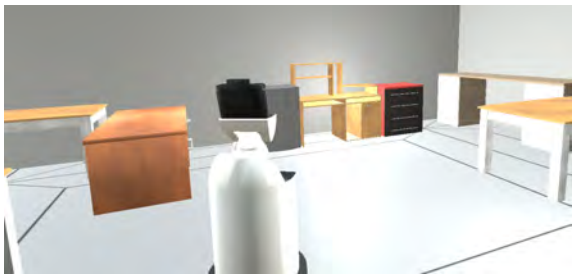
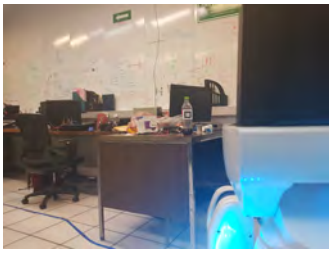


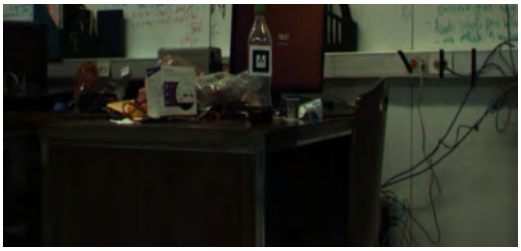
Figura 5.6: Localización del robot en el ambiente virtual

5.3.3. Localización y visualización de objetos no fijos del ambiente real al ambiente virtual

Los tags agregados a objetos son detectados por el robot y situados correctamente en el espacio con relación a la posición del robot, la información enviada, se recibe en el ambiente virtual y los objetos virtuales se colocan en la posición correspondiente a su contraparte real.



(a) Botella en el ambiente real



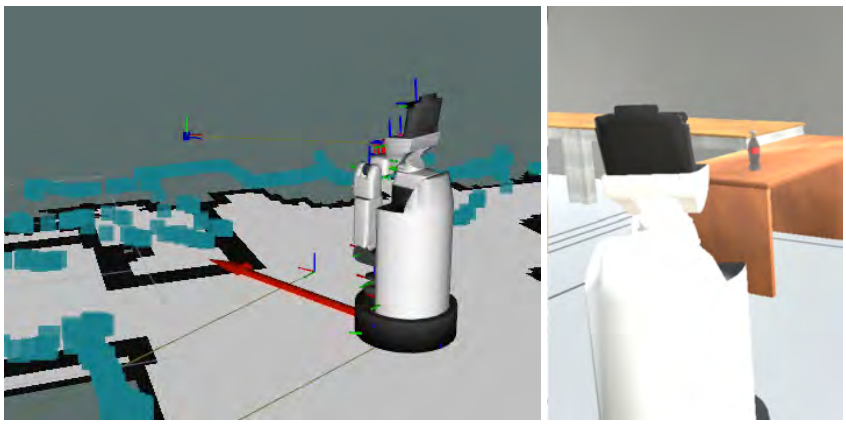
(b) Botealla vista desde la cámara del robot

Figura 5.7: Objeto en el mundo real

5.3.4. Imágenes reales para mejorar la experiencia

Oculus

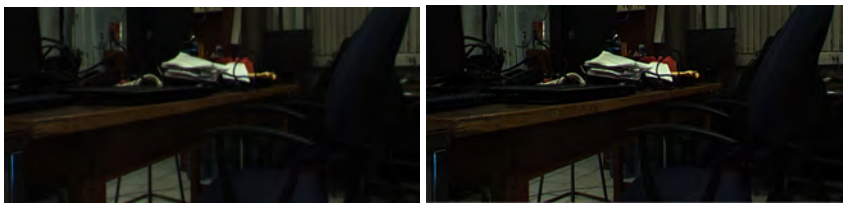
La prueba se realizó utilizando filtro Laplaciano de convolución, al cambiar la imagen recibida por la imágen filtrada se nota una mejora en la calidad del video. Al hacer convolución para filtrar la imagen, el proceso es lento, por lo que el video disminuye los FPS desplegados.



(a) Representación en RVIZ

(b) Representación de la botella en mundo virtual

Figura 5.8: Envío de datos abstractos al ambiente virtual



(a) Imagen sin filtro

(b) Imagen filtrada

Figura 5.9: Filtrado de video

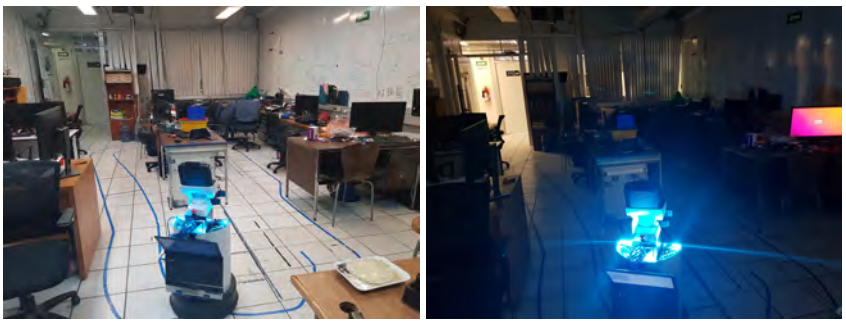
Gear VR

El filtrado no fue posible realizarlo con Android.

5.3.5. Iluminación en el mundo virtual

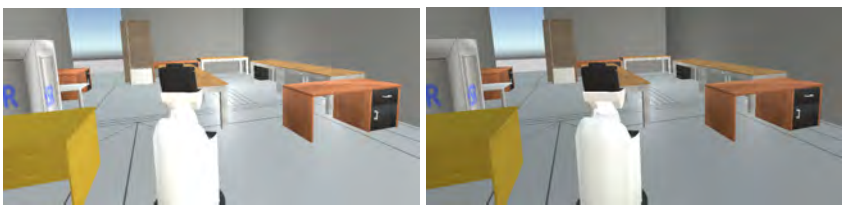
Oculus

La iluminación del mundo virtual cambia de acuerdo a la iluminación en el mundo real, al disminuir o apagar completamente la luz en el ambiente real, el ambiente virtual disminuye la iluminacion y visceversa. Utilizando el threshold, al estar en completa oscuridad el mundo real, en el mundo real aún se ve con claridad aunque con poca luz.



(a) Robot en el ambiente iluminado (b) Robot en el ambiente sin iluminación

Figura 5.10: Iluminación en el ambiente real



(a) Robot virtual con ambiente iluminado (b) Robot virtual con ambiente sin iluminar

Figura 5.11: Iluminación en el ambiente virtual

Gear VR

El cambio de iluminación no fue posible realizarlo con Android.

Capítulo 6

Conclusiones y Trabajo futuro

6.1. Conclusiones

El objetivo fundamental de esta tesis era crear un robot virtual que represente a un robot real remoto para realizar teleoperación sobre este utilizando una interfaz inmersiva fácil de utilizar y exportar a otras plataformas, que requiera pocos recursos computacionales para reproducir el ambiente virtual, utilizando abstracción de datos para una comunicación más eficiente y rápida.

Así, la aportación principal de este trabajo consiste en el diseño y la implementación de la interfaz de comunicación entre el robot real y el ambiente virtual, la interfaz humano-computadora para introducción de comandos y el ambiente virtual donde navega el robot virtual, que, al ser realizada con ROS y UNITY 3D, además de sockets, exportar a otras plataformas se realiza de manera simple. A partir del trabajo realizado, las pruebas y resultados de este trabajo podemos concluir que la utilización de un ambiente presenta una interacción más rápida que utilizan-

do únicamente video debido a que la exploración en el ambiente virtual depende mayormente del despliegue de gráficos por parte del equipo computacional local.

En cuanto a la navegación y teleoperación del robot, la latencia se ve reducida de manera importante debido a la transmisión de menor cantidad de datos que la utilizada para la transmisión de video, en el ambiente virtual además es posible explorar otras partes del ambiente y no estar limitado a las cámaras existentes en el ambiente real o en el caso del presente trabajo, a las cámaras con las que cuenta el robot. En un ambiente real con ausencia de luz es imposible utilizar las cámaras del robot para ver el ambiente. El robot cuenta con sensores láser para su localización, estos no requieren luz visible para funcionar por lo que en un ambiente en completa obscuridad la ubicación del robot se puede obtener y ser utilizada en el ambiente virtual.

6.2. Trabajo futuro

Con el fin de mejorar el trabajo de la presente tesis, se propone realizar el siguiente trabajo:

- **Creación dinámica de ambiente virtual:** En el presente trabajo, el ambiente virtual se creó manualmente utilizando el mapa en 2D creado utilizando el robot y después poder navegarlo. Es deseable que este proceso sea realizado de manera automática durante la navegación del robot. Esto presenta un reto debido a la propia reconstrucción del ambiente así como la transmisión de la información de reconstrucción al ambiente virtual ya que podría exceder la información enviada mediante video que es el principal problema en la teleoperación. Además del reconocimiento y separación de objetos independientes en el ambiente.

- **Reconocimiento automático de objetos no fijos:** Aunque la solución utilizando códigos QR para ubicar este tipo de objetos es funcional, se encuentra limitada ya que hay que asignar un tipo de objeto a cada código, además de que objetos sin código QR no son visibles en el ambiente virtual.

- **Reconocimiento automático de personas y su representación en el ambiente virtual:** El presente proyecto se enfocó únicamente en la navegación del robot y el reconocimiento de algunos objetos en el ambiente. Es de mucha utilidad

poder también observar e interactuar con personas reales utilizando el ambiente virtual.

- **Creación de bibliotecas para comunicación:** La comunicación se realiza de manera adecuada, aunque si se desea agregar alguna funcionalidad extra para el control del robot, es necesario hacer paso a paso el formato de envío de datos, la recepción y procesamiento de estos. Sería muy útil contar con una modularización para hacer más sencillo el proceso.
- **Compresión de datos transmitidos:** La latencia puede verse reducida aun más al comprimir de una manera eficiente los datos transmitidos de comandos y posiciones de objetos y del robot.
- **Mejoramiento en transmisión de video:** Dadas las herramientas con las que se contaban y los límites de tiempo se decidió hacer la transmisión de video mediante un flujo mjpeg, esta transmisión puede mejorarse con un formato de video con más compresión.

Código, Sockets

Éste código muestra la manera en la que se crean sockets en Unity 3D y en ROS para poder iniciar una comunciación entre las dos terminales. El código muestra la creación de sockets TCP y UDP.

Código ROS, C++, inicio de comunicación confiable:

```
socket_desc = socket(AF_INET , SOCK_STREAM , 0);

if (socket_desc == -1){
        printf("Could not create TCP socket");
}

server.sin_family = AF_INET;
server.sin_addr.s_addr = INADDR_ANY;
server.sin_port = htons( TCPPORT );

if( bind(socket_desc,(struct sockaddr *)&server , sizeof(server)) < 0){
        perror("bind TCP failed. Error");
```

```
    return;
}

listen(socket_desc , 3);
```

Código en Unity 3D, C#, inicio de comunicación confiable:

```
IPAddress ipAddress = IPAddress.Parse(hostAddress);
IPEndPoint remoteEP = new IPEndPoint(ipAddress, hostPort);
sender = new Socket(ipAddress.AddressFamily, SocketType.Stream,
 ProtocolType.Tcp);
sender.Connect(remoteEP);
```

Código ROS, C++, inicio de comunicación no confiable:

```
if ((s=socket(AF_INET, SOCK_DGRAM, IPPROTO_UDP)) == -1){
        perror("socket UDP");
}

memset((char *) &si_me, 0, sizeof(si_me));

si_me.sin_family = AF_INET;
si_me.sin_port = htons(UDPPORT);
si_me.sin_addr.s_addr = htonl(INADDR_ANY);

if( bind(s , (struct sockaddr*)&si_me, sizeof(si_me) ) == -1) {
        perror("bind UDP");
}
```

Código Unity3D, C#, inicio de comunicación no confiable:

```
IPAddress ipAddress =  IPAddress.Parse(hostAddress);
sender = new Socket(ipAddress.AddressFamily, SocketType.Dgram,
 ProtocolType.Udp);
EndPoint remoteEP = new IPEndPoint(ipAddress, hostPort);
sender.Connect(ipAddress, hostPort);
```

Código, Control del robot

Éste código muestra:

- La manera en la que se leen las entradas del usuario para controlar el robot desde el ambiente local virtual, el envío de tales datos a la computadora que controla al robot real.
- La recepción de datos por la computadora que controla al robot real y publicación de la información a un "Topic" de ROS para que el robot realice la tarea solicitada.

Lectura del movimiento de los lentes de realidad virtual:

```
OVRPose headPose;
headPose.orientation=UnityEngine.XR.InputTracking.
GetLocalRotation(UnityEngine.XR.XRNode.Head);
senderUDP.sendMSG(headPose.orientation);
```

Limites de lectura de movimiento:

```
if ( (horizontal_position - previous_horizontal_possition) > threshold
|| (vertical_position - previous_vertical_possition) > threshold )
        senderUDP.sendMSG(headPose.orientation);
```

Creación del "Publisher" que envía comandos al robot:

```
std::string pub_head_move_topic_name;
node_handle.param<std::string>("pub_head_move_topic_name",
pub_head_move_topic_name,"/hsrb/head_trajectory_controller/command");
pub_head_move=node_handle.advertise<trajectory_msgs::JointTrajectory>
(pub_head_move_topic_name, 10);
```

Recepción de información del movimiento de la cabeza en ROS:

```
recvfrom(data, buf, BUFLEN, 0, (struct sockaddr *) &si_other, &slen)
```

Código que publica la información recibida para que el robot ejecute el movimiento de la cabeza:

```
trajectory_msgs::JointTrajectory trajectory;
trajectory <- data
pub_head_move.publish(traj);
```

Lectura de entradas del Dualshock 4 en Unity 3D:

```
float leftX = Input.GetAxis("LeftX");
float leftY = Input.GetAxis("LeftY");
```

```
bool pressX = Input.GetButtonDown("X");
```

Envío de mensajes de actualización de estados en Unity 3D:

```
msg = data(leftX);
senderUDP.sendMSG(msg);
```

Envío de mensajes de actualización de eventos/comandos en Unity 3D:

```
msg = data(pressX);
senderTCP.sendMSG(msg);
```

Recepción de mensajes de actualización de estados en ROS:

```
recvfrom(data, buf, BUFLEN, 0, (struct sockaddr *) &si_other, &slen)
```

Recepción de mensajes de actualización de eventos/comandos en ROS:

```
read(client_sock , client_message , sizeof(data))
```

Código que publica la información recibida para que el robot ejecute el movimiento de la base:

```
geometry_msgs::Twist twist <- data;
pub_base_twist.publish(twist);
```

Código, Posición del robot y objetos

Éste código muestra la lectura de posición del robot, envío y recepción para su localización en el ambiente virtual.

Lectura de la posición del robot respecto al mapa:

```
listener.lookupTransform("/map", "/base_footprint", ros::Time(0), transform);
```

Envío de la posición del robot en ROS mediante mensajes de actualización de estados:

```
char messageUDP[BUFLEN];

memset( &messageUDP, 0, sizeof(messageUDP));
```

```
strncpy(messageUDP, transform_data, transform_data.size());

sendto(s, messageUDP, strlen(messageUDP), 0,
 (struct sockaddr*) &si_other, slen);
```

Recepción de mensajes de actualización de estados en Unity 3D de la posición del robot:

```
robot.position = new Vector3(transform_data.position);
robot.rotation = Quaternion.Euler(transform_data.rotation);
```

Lectura de la posición de objetos respecto al mapa:

```
void alvarTagsCallback(const
ar_track_alvar_msgs::AlvarMarkers::ConstPtr& message){

UDPmessage <- message.data;
sendto(s, messageUDP, strlen(messageUDP), 0,
 (struct sockaddr*) &si_other, slen)

}
```

Recepción de mensajes de actualización de estados en Unity 3D de la posición de los objetos:

```
sender.BeginReceiveFrom(buffer, 0, bufSize,
 SocketFlags.None, ref remoteEP, recv = (ar) => {
        [id,transform_data] <- buffer.data;
}
```

```
object = findObject(id);

object.transform = transform_data;
```

Código, Tareas asíncronas y procesamiento de imágenes

Éste código muestra la creación y manejo de tareas asíncronas en Unity 3D para realizar procesamiento de imágenes sin detener la interfaz virtual y la obtención de datos RGB de una imágen para convertirlos al espacio HSV y obtener la iluminación promedio de una escena.

Creación de una tarea asíncrona en Unity 3D:

```
taskreturn = Task.Run(() => TaskAsyncCountDown(imageData,w, h, im));
```

Espera activa de una tarea asíncrona en Unity 3D:

```
return await Task.Run(() =>  LogToTUnityConsole(imageData, w, h));
```

Consulta de término de una tarea asíncrona en Unity 3D:

```
if(taskreturn != null && taskreturn.IsCompleted)
```

Obtención de datos de una textura en Unity 3D:

```
Texture2D tex;
Color32[] imageData = tex.GetPixels32();
```

Conversión de datos RGB a HSV en Unity 3D:

```
Color.RGBToHSV(imageData[i], out H, out S, out V);
```

Obtención de iluminación promedio de una imagen en Unity 3D:

```
for(int i = 0; i < lights.Length; i++){
        lights[i].color = Color.HSVToRGB(0.0f,0.0f,
        promedioV+minimumLight);
}
```

Bibliografía

[Ama18] Amazon. Samsung Note 8, 2018.

[BCL14] Nassima Bouzakaria, Cyril Concolato, and Jean Le Feuvre. Overhead and performance of low latency live streaming using MPEG-DASH. *IISA 2014 - 5th Int. Conf. Information, Intell. Syst. Appl.*, pages 92–97, 2014.

[Boo11] Booredatwork.com. Nintendo 3DS AR Game Hands-on (Augmented Reality), 2011.

[Cat16] Patrick Catanzariti. There Are More Virtual Reality Headsets Than You Realize! 2016.

[CE15] Lauren Collins and Scott R. Ellis. *Mobile devices: Tools and technologies*. 2015.

[Deb18] Patrick Debois. Ultra Low Latency Video Streaming: The Current State, 2018.

[Dr.06] Dr. Boris Escalante R. Procesamiento Digital de Imágenes, 2006.

[Equ] Equipo médico de Ginecología y obstetricia. Robot Da Vinci.

[Esp19] Real Academia Española. Telecontrol, 2019.

[FPWW03] R Fisher, S Perkins, A Walker, and E Wolfart. Laplacian/Laplacian of Gaussian, 2003.

[Fre18] Free Software Foundation. Socket, 2018.

[Gat08] Bill Gates. A robot in every home: overview/The robotic future. *Sci. Am.*, 2008.

[GRA15] RICHARD GRAY. Dawn of the ROBO-NURSE: Toyota droid can fetch and carry medication, water and even the TV remote for patients, 2015.

[HBG+13] Tamás Haidegger, Marcos Barreto, Paulo Gonçalves, Maki K. Habib, Sampath Kumar Veera Ragavan, Howard Li, Alberto Vaccarella, Roberta Perrone, and Edson Prestes. Applied ontologies and standards for service robots. *Rob. Auton. Syst.*, 61(11):1215–1223, 2013.

[Hou19] Houston Mechatronics. Aquanaut, 2019.

[Idl] Idlcoyote. Sharpen.

[Int96] International Organization for Standardization. ISO/IEC 7498-1: Information technology open systems interconnection basic reference model, 1996.

[ISO12] ISO. ISO 8372:2012 - Robots and robotic devices - Vocabulary, 2012.

[Kar17] Jen Karner. How to fix a stuck keyboard in your Gear VR, 2017.

[KH02] G.D. Kessler and L.F. Hodges. A network communication protocol for distributed virtual environment systems. 2002.

[LOP15] NAPIER LOPEZ. Sign up to try the Hololens in New York from today, 2015.

[MAR80] MARVIN MINSKY. TELEPRESENCE. *OMNI Mag.*, page 9, 1980.

[MCR+17] F. Muhla, F. Clanché, C. Rose, A. Cosson, and G. Gauchard. Biomechanical and human behavior assessment using virtual reality to challenge balance and posture for the elderly and patients with Parkinson's disease. *Comput. Methods Biomech. Biomed. Engin.*, 2017.

[MGMADGM17] Jorge Martín-Gutiérrez, Carlos Efrén Mora, Beatriz Añorbe-Díaz, and Antonio González-Marrero. Virtual technologies trends in education. *Eurasia J. Math. Sci. Technol. Educ.*, 2017.

[Mil68] Robert B Miller. Response time in man-computer conversational transactions INTRODUCTION AND MAJOR CONCEPTS. 1968.

[MK94] Paul Milgram and Fumio Kishino. Taxonomy of mixed reality visual displays. *IEICE Trans. Inf. Syst.*, 1994.

[oI17] Tetsunari Inamura of National Institute of Informatics(NII). PartnerRobotChallengeVirtual/common-unity, 2017.

[Pri18] Printsome.com. 7 brilliant augmented reality projects that don't involve 'Pokémon GO', 2018.

[ROSa] ROS. Navigating The Filesystem.

[ROSb] ROS. ROS Topics.

[ROSc] ROS. Understanding ROS Nodes.

[ROSd] ROS. web_video_server.

[Ros18] Ros.org. ar track alvar, 2018.

[Sam18a] Samsung. Galaxy Note 8 Spec+, 2018.

[Sam18b] Samsung. Gear VR, 2018.

[Sav18] Jesús Savage Carmona. Justina, un robot de servicio para una casa inteligente. 2018.

[SGG12] Abraham Silberschatz, Peter Baer Galvin, and Greg Gagne. *Operating systems concepts*. 2012.

[Sha04] Govil-Pai Shalini. *Principles of Computer Graphics*. 2004.

[Shi02] Takashi Shibata. Head mounted display. 23:1–3, 2002.

[Sol19] Jose Solano. El modelo OSI, 2019.

[Son] Sony. DualShock 4.

[Syn11] Synthiam. Joystick Control Robot Tutorial with DJ Sures, 2011.

[Tok19] Tokyo Opensource Robotics Kyokai. ROS Consulting Support, 2019.

[Toy15] Toyota. Toyota Shifts Home Helper Robot R&D into High Gear with New Developer Community and Upgraded Prototype, 2015.

[Toy18] Toyota. Human Support Robot, 2018.

[Toy19] Toyota. HSR Basic Specifications, 2019.

[Unia] Unity. Game engines-how do they work?

[Unib] University of Colorado System. Berkeley Sockets API - I.

[Uni15] Northwestern Switzerland University of Applied Sciences and Arts. BIT magazine, 2015.

[Uni19] University of Waterloo. SOFTWARE ENGINEERING, 2019.

[Unr18] Unreal Engine. Oculus Rift, 2018.

[WBM+18] Brandon Wilson, Matthew Bounds, David McFadden, Jace Regenbrecht, Loveth Ohenhen, Alireza Tavakkoli, and Donald Loffredo. VETO: An immersive Virtual Environment for Tele-Operation. *Robotics*, 2018.

[Wik] Wikipedia. Android (operating system).

[Wik18a] Wikipedia. HSL and HSV, 2018.

[Wik18b] Wikipedia. Oculus Rift, 2018.

[Wik18c] Wikipedia. Samsung Gear VR, 2018.

[Wik19] Wikipedia. Telerobotics, 2019.

[WYTK18] Ming Chang Wen, Cheng Hsuan Yang, Meng Han Tsai, and Shih Chung Kang. Teleyes: A telepresence system based on stereoscopic vision and head motion tracking. *Autom. Constr.*, 89:199–213, 2018.

[YNK+] Takashi Yamamoto, Tamaki Nishino, Hideki Kajima, Mitsunori Ohta, and Koichi Ikeda. Human Support Robot (HSR). Technical report, Vancouver.