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

SELECTOR DE ESTRATEGIAS DE POSICIONAMIENTO DE OBJETOS CON ROBOTS DE SERVICIO UTILIZANDO REDES NEURONALES PROFUNDAS

TESIS

QUE PARA OPTAR POR EL GRADO DE:

DOCTORA EN CIENCIA E INGENIERÍA DE LA COMPUTACIÓN

PRESENTA:
MARTHA ANGÉLICA NAKAYAMA CERVANTES

TUTORES
DR. JESÚS SAVAGE CARMONA
POSGRADO DE INGENIERÍA, UNAM

DR. ERNESTO BRIBIESCA CORREA
INSTITUTO DE INVESTIGACIONES EN MATEMÁTICAS APLICADAS Y EN SISTEMAS, UNAM

CIUDAD UNIVERSITARIA, CD.MX., MARZO, 2022

Agradecimientos

Agradezco infinitamente...

A mi mamá, quien me dió la vida y ha estado conmigo siempre, apoyándome en cada momento de mi vida, todo lo que soy te lo debo a tí.

A Alexis, el amor de mi vida y mi compañero de aventuras, quien siempre confía en mí y apoya mis locuras aún hasta el otro lado del mundo. Te amo.

A mi familia, quienes siempre me han apoyado y creído en mí, son el pilar de mi vida, sin ustedes no habría llegado hasta aquí.

A mi familia política, que ya son parte importante de mi vida y que me han adoptado como parte de la suya, gracias por el apoyo y el aliento para seguir siempre adelante.

A mi gran amiga Tanya, quien siempre estuvo acompañándome en esta aventura, compartiendo alegrías y tristezas, eres la mejor.

A mi gran amigo Eduardo, quien ha estado conmigo desde el inicio de la licenciatura, compartió conmigo la maestría y afortunadamente estuvo apoyandome en el doctorado, gracias por todos los grandes momentos y sabios consejos.

A todos mis amigos, que siempre están a mi lado (o incluso a la distancia) dándome ánimos para seguir adelante, es una fortuna tener amigos como ustedes, los quiero.

A Saki, Maki, Tux y Ari, sin quienes no habría logrado mantenerme cuerda durante el encierro de la pandemia, le dan alegría a mi vida.

A mi tutor, el Dr. Jesús Savage, por todo su apoyo y consejos, gracias por compartir su sabiduría conmigo.

A mi cotutor, el Dr. Ernesto Bribiesca, por su apoyo y comprensión, gracias por siempre tener palabras reconfortantes cuando las necesitaba.

A mi comité tutoral y sinodales, por su apoyo y recomendaciones que hicieron un gran aporte a mi trabajo, les agradezco sus ideas y consejos.

A mis compañeros y amigos del Laboratorio de BioRobótica, por compartir sus conocimientos y apoyarme cuando fue necesario, por compartir muchas experiencias en el laboratorio, en competencias y presentaciones, gracias chicos y chicas.

Al personal del Posgrado de Ciencia e Ingeniería de la Computación, por todas sus atenciones, su apoyo y comprensión, es un placer trabajar con personas como ustedes.

A mis amigos de México, Japón, Canadá, Alemania, Venezuela, China, India, Vietnam, Francia y de algunos otros países, a quienes tuve el gusto de conocer durante este proyecto llamado Doctorado y que me dejaron un grato recuerdo y una amistad que conservaré para siempre.

A CONACYT, por la beca recibida sin la cual no habría sido posible completar este proyecto.

A DGAPA-PAPIIT por el apoyo recibido a esta tesis con el proyecto AG101721.

A la UNAM, mi alma mater, por ser siempre la mejor.

Resumen

Colocar objetos puede parecer una tarea trivial para un ser humano, pero para un robot de servicio es una tarea demandante dado que debe superar una serie de situaciones como descifrar dónde y cómo poner los objetos, lidiar con las cambiantes condiciones del ambiente y los obstáculos a esquivar, determinar los movimientos que se deben ejecutar para colocar exitosamente, además de considerar las restricciones de movimientos (propias de la fisiología del robot), entre otras cosas.

Para ayudar a los robots de servicio a realizar esta tarea, se creó el Selector de Estrategias de Posicionamiento de Objetos, el cual propone utilizar un conjunto de movimientos de colocación simples y probados o estrategias de posicionamiento (colocación) y con esto evitar realizar complicados cálculos de trayectorias y/o la planeación y control de movimientos, los cuales generalmente consumen muchos recursos y podrían llegar a tornarse complejos o ineficientes.

El Selector utiliza las imágenes RGB-D del robot y un conjunto reducido de estrategias para hacer una predicción y seleccionar la estrategia óptima en cada situación. El Selector consta de tres componentes principales: el módulo de preprocesamiento, la red neuronal y el discriminador. El primer módulo procesa las imágenes RGB-D de acuerdo a las estrategias existentes para que sean aptas como entradas para la red neuronal. La red neuronal recibe las imágenes procesadas y las utiliza para predecir la probabilidad de éxito de cada estrategia. Estas predicciones se pasan al discriminador el cual selecciona la estrategia que tiene la mayor probabilidad de éxito y la devuelve al robot para ejecutar los movimientos de la estrategia seleccionada.

Se utilizó un simulador con ambiente virtual para recopilar un conjunto de datos lo suficientemente grande para entrenar el modelo y realizar los experimentos. También se probó el funcionamiento del Selector en colaboración con otros sistemas existentes (uno de teleoperación y otro de detección de objetos) adicionando funcionalidades a los mismos y probando que el Selector puede formar parte de tareas o sistemas más complejos. En general el Selector de Estrategias de Posicionamiento de Objetos se desempeña adecuadamente y cumple con el objetivo para el que fue diseñado.

Abstract

Placing objects may seem like a trivial task for a human being, but for a service robot, it is a demanding task. The robot must overcome a series of situations such as deciphering where and how to place objects, dealing with changing environment conditions and avoiding obstacles, determining the movements that must be executed to place successfully. Additionally it must consider movement restrictions (typical of the robot's physiology) among other things.

To help service robots with this task, the Object Positioning Strategy Selector was created. It proposes to use a set of simple and proven placing movements or placing strategies to avoid complicated trajectory calculations and/or movement planning and control, which generally consume many resources and could become complex or inefficient.

The Selector uses the RGB-D images of the robot and a reduced set of strategies to make a prediction and select the optimal strategy in each situation. The Selector consists of three main components: the preprocessing module, the neural network, and the discriminator. The first module processes the RGB-D images according to the existing strategies in order to make them suitable as inputs for the neural network. The neural network receives the processed images and uses them to predict the probability of success for each strategy. These predictions are passed to the discriminator, which selects the strategy that has the highest success probability and returns it to the robot to execute the movements of the selected strategy.

A simulator with a virtual environment was used to collect a data set large enough to train the model and perform the experiments. The Selector's performance was also tested in collaboration with other existing systems (a teleoperation and an object detection system), adding them functionalities and proving that the Selector can be part of more complex tasks or systems. In general, the Object Positioning Strategy Selector performs adequately and meets the objective for which it was designed.

Índice general

Índice de figuras

Capítulo 1

Introducción

Un robot es una máquina autónoma programable capaz de realizar diversas tareas para ayudar al ser humano. Según la Federación Internacional de Robótica [1], los robots de servicio son robots que operan semiautónoma o totalmente autónomamente para realizar servicios útiles para el bienestar de los seres humanos. Es importante enfatizar que estos robots son diferentes a los industriales, ya que su objetivo es simplificar el trabajo humano en casas, oficinas, hospitales, etc., para lo cual tienen que moverse de manera intencional, evitando obstáculos, reconocer e identificar a los humanos, comunicarse a través del lenguaje hablado, ver, tomar, llevar y entregar objetos, entre otros tipos de acciones o conductas intencionales [2]. Por lo tanto, los robots de servicio deben poder trabajar en diversos tipos de entornos no estructurados y condiciones ambientales cambiantes además de tener una estrecha interacción con los humanos [3]. Los robots de servicio deben ser capaces de hacer diagnósticos, tomar decisiones y hacer planes para llevar a cabo sus funciones de manera exitosa. Todas estas situaciones hacen que el desarrollo de los robots de servicio implique grandes retos científicos y tecnológicos, muchos de los cuales aún no han sido completamente resueltos y son aún temas de investigación que se desarrollan en universidades y centros de investigación.

Hoy en día los robots están siendo usados en diferentes campos y actividades, por ejemplo, como asistentes, mascotas, juguetes, guías turísticas, etc. [4, 5, 6, 7] por lo que se puede decir que los robots juegan un papel esencial en la sociedad humana (figura 1.1). Se espera que en un futuro cercano dentro de la casa habrá robots pequeños que limpien, aspiren y trapeen el piso; robots fijos encargados del lavado y planchado de la ropa; y robots de tipo humanoide que fungirán como asistentes generales [8]. En el exterior habrá robots que cortarán el pasto y otros que harán rondines de vigilancia. Así como se incorporaron a la vida cotidiana los televisores, las computadoras y los teléfonos celulares, los robots de servicio también lo harán y llegarán a ser muy familiares si se cumplen las expectativas que tenemos actualmente [9].

Figura 1.1: Algunas aplicaciones de los robots

A pesar de que los robots ya están siendo usados con éxito, aún queda mucho por hacer para incrementar su eficiencia [10]. Por ejemplo, para que los robots sean aceptados como entes "naturales" con quienes se puede interaccionar, necesitan habilidades sociales más sofisticadas como el reconocer los contextos y las convenciones sociales [11]. En términos generales los seres humanos prefieren interactuar con las máquinas de manera similar a la que interactúan con otras personas, debido a esto, los robots interactivos están siendo más populares. Para lograr este tipo de interacción los robots necesitan mostrar cierto grado de adaptabilidad y flexibilidad y en algunas situaciones, es deseable que un robot desarrolle sus habilidades de interacción conforme pasa el tiempo [12]. Entonces, el diseño del comportamiento, la apariencia, las habilidades cognitivas y sociales de un robot se vuelve ahora un gran reto que requiere una serie de conocimientos y habilidades interdisciplinarias. Los especialistas en robótica ya no solo le apuestan a construir robots que cumplan tareas específicas sino también a construir robots que además sean capaces de participar en mayor medida en la sociedad humana, que sean más agradables y cumplan la voluntad de los seres humanos que los rodean [13].

1.1. Descripción del problema

Colocar objetos de uso cotidiano en áreas designadas es una habilidad necesaria en los robots de servicio. Para poder realizar autónomamente tareas cotidianas simples como poner la mesa, un robot de servicio doméstico debe ser capaz de descifrar la mejor manera y lugar para colocar los objetos involucrados [14]. Determinar los movimientos que se deben ejecutar para colocar exitosamente un objeto puede parecer natural o trivial para la mayoría de los humanos, sin embargo, esto puede ser particularmente desafiante para un robot ya que su entorno puede tener una gran variedad factores a considerar [15].

La habilidad de colocar objetos autónomamente es de gran utilidad en un robot de servicio ya que esta tarea generalmente forma parte de otras más complejas. Colocar objetos es parte fundamental de tareas como traer y llevar objetos, las cuales son básicas para un robot de servicio. Por ejemplo, en muchas de las competencias de robots, pruebas más elaboradas fallan por no tener éxito en la colocación de objetos y esto afecta a toda la tarea y el desempeño general del robot [16]. Por otro lado, si esta habilidad se adapta a otros sistemas, por ejemplo a los de teleoperación, se puede dotar de mayor autonomía a los mismos, además de facilitar mucho esta tarea al operador, dado que es una de las más difíciles de controlar por el nivel de precisión que requiere y los problemas que puede llegar a causar la latencia en la comunicación.

Al colocar objetos, los robots de servicio deben superar un gran número de problemas, por ejemplo: otros objetos obstruyendo su camino, las características del ambiente, seleccionar áreas viables para la colocación, la planeación de movimientos complejos, etc. Para una colocación de objetos exitosa, el robot debe tomar todos esos factores en consideración además de manejar una gran cantidad de información y hacer una serie de cálculos y planeación para encontrar una estrategia (serie de movimientos) de colocación apropiados [17]. Si no se planea adecuadamente, esta estrategia puede volverse muy compleja o ineficiente y el resultado puede ser desde una falla al colocar los objetos hasta dañar el robot o algún elemento del ambiente. Adicionalmente, la planeación de dichos movimientos suele ser muy tardado y consumir recursos computacionales valiosos.

Para llevar a cabo la colocación, es deseable utilizar el menor número de recursos posibles y usar solamente movimientos del robot simples y confiables dado que los movimientos muy complejos incrementan el riesgo de colisiones, toman más tiempo para ser ejecutados o son más difíciles de programar. Sería entonces conveniente contar con un pequeño conjunto de movimientos simples que hayan sido previamente probados (es decir un conjunto de estrategias de colocación), confiables y seguros para ser ejecutados con la certeza de que funcionan adecuadamente.

Este problema es particularmente desafiante dado que es necesario predecir lo que sucede cuando un objeto es colocado en un área designada y además considerar que en dicha área pueden encontrarse otros objetos que obstaculicen la tarea. Adicionalmente, en el espacio físico algunas colisiones se pueden considerar como dañinas y otras no, por ejemplo, un ligero empujón a un objeto o colocar un objeto sobre un obstáculo plano no es dañino. Por otro lado, las colisiones dañinas dependen en gran parte de la física de los objetos con los que interactúan en el entorno, lo cual es muy complejo de predecir.

Las redes neuronales son una buena opción para resolver este tipo de problemas, sin embargo, en ocasiones resulta complicado obtener un conjunto de datos lo suficientemente grande para poder entrenar los modelos requeridos. Para so-

lucionar este problema, recientemente se ha optado por utilizar simuladores y/o ambientes virtuales para obtener el conjunto de datos y entrenar los modelos. Usando esta técnica el modelo entrenado se puede probar posteriormente en un ambiente real con pocos o mínimos ajustes (idealmente).

1.2. Hipótesis y objetivo

Consideremos la situación en la que un robot de servicio doméstico necesita colocar un objeto en una superficie determinada. Esto puede ser en un ámbito de competencias, como parte de una tarea mayor de buscar y llevar objetos de un lugar a otro, o bien como parte de un sistema de teleoperación en el cual los movimientos finos pueden ser complicados debido a problemas de latencia en las comunicaciones o la complejidad de controlar todas las articulaciones del robot remotamente.

Asumamos que de alguna manera el robot ha logrado navegar exitosamente hasta posicionarse frente a un mueble con la superficie destino en donde se va a colocar un objeto que el robot lleva en su mano. Encima de dicha superficie destino se encuentran algunos otros objetos cotidianos colocados aleatoriamente, los cuales se desea que no sean dañados al colocar el objeto en cuestión. Entonces surgen algunas interrogantes:

- ¿Es posible colocar el objeto exitosamente usando un conjunto reducido de estrategias de colocación simples?
- ¿Es posible utilizar recursos simples como las imágenes del entorno obtenidas de las cámaras del robot para predecir el éxito de los movimientos del robot?
- ¿Se pueden evitar movimientos del robot que resulten dañinos para el mismo robot o los elementos del ambiente usando dichas predicciones?
- ¿Se puede seleccionar una estrategia de colocación simple y probada que prevea resultados óptimos sin tener que realizar cálculos y planeación complejos?
- ¿Se puede utilizar un ambiente virtual de simulación para resolver el problema y posteriormente utilizar el resultado en un robot y situación reales?

Entonces, el objetivo es lograr que un robot de servicio doméstico coloque autónoma y exitosamente un objeto en una superficie dada donde ya se encuentran otros objetos cotidianos sin tener ningún tipo de colisiones dañinas tanto para el robot como para el entorno (como derribar otros objetos, dañar el robot o los muebles, etc.). Además, se desea utilizar un número limitado de recursos (las imágenes obtenidas por las cámaras del robot) y un conjunto reducido de estrategias de colocación simples previamente probadas (sin la necesidad de

calcular complicadas trayectorias y/o control de movimientos complejos). Por lo tanto, el robot debe contar con la habilidad de seleccionar la estrategia que tenga mayor viabilidad y colocar el objeto usando dicha estrategia.

Para alcanzar este objetivo se necesita predecir la probabilidad de éxito o fracaso del robot al colocar objetos para cada una de las estrategias del conjunto reducido para poder seleccionar aquella que tenga la mayor viabilidad apoyándose en las predicciones (tanto del éxito como de las consecuencias obtenidas) y después ejecutar los movimientos de la estrategia seleccionada en el robot. Dado que la solución de este tipo de problemas suele requerir de una gran cantidad de datos para pruebas y/o entrenamiento de redes neuronales, se desea que los datos requeridos puedan ser obtenidos usando ambientes virtuales de simulación para ahorrar recursos materiales y tiempo.

1.3. Organización de la tesis

Esta tesis está organizada de la siguiente manera: en el Capítulo 1 se presenta una breve introducción al tema, la descripción del problema a resolver, la hipótesis y objetivo planteados. En el Capítulo 2 se presenta el marco teórico en donde se explican algunos temas y conceptos relacionados importantes y útiles para comprender el desarrollo del presente proyecto. En el Capítulo 3 se detalla teoría y conceptos de aprendizaje profundo, los cuales son de vital importancia para este trabajo. En el Capítulo 4 se describe el conjunto de datos utilizado, su método de obtención y la arquitectura del Selector de Estrategias. En el Capítulo 5 se detalla la integración del Selector con los sistemas de Teleoperación y Detección de Objetos desarrollados en el laboratorio de biorobótica, así como las nuevas funcionalidades desarrolladas para mejora de los mismos y su aplicación usando el Selector. En el Capítulo 6 se describen los experimentos y resultados obtenidos. Finalmente, en el Capítulo 7 se discuten los resultados y se presentan las conclusiones, las contribuciones del proyecto al campo, las publicaciones derivadas del mismo y posible trabajo futuro.

Capítulo 2

Marco teórico

Resulta fácil notar que para el desarrollo de un robot de servicio eficiente es necesario hacer uso de una variedad de conceptos, teorías, herramientas, técnicas e incluso adaptar tópicos de distintas áreas del conocimiento de manera creativa para lograr que los robots se desempeñen de manera similar a como lo hacemos los seres humanos. Esta tarea no es nada sencilla, involucra conocer un amplio panorama de disciplinas y tecnologías, así como también requiere tener ideas innovadoras para aplicar temas que podrían parecer no estar relacionados con la robótica o entre sí. Adicionalmente, es importante relacionar desarrollos previos para aprovechar el avance que otros expertos en sus áreas han aportado. Para poder comprender mejor el contexto del proyecto que se presenta en este trabajo, se dará una breve introducción a algunos temas y conceptos importantes relacionados con el mismo.

2.1. Robot de servicio Toyota HSR

Los Robots de Servicio Doméstico (*Domestic Service Robots*) son diseñados para ser utilizados en un ambiente hogareño en cualquier parte del mundo, para poder apoyar en el hogar, en el cuidado de los adultos mayores y/o personas con discapacidades. Para poder lograr esto se requiere mucha investigación tanto en algoritmos como en hardware para que puedan coexistir en la vida cotidiana con los humanos.

Toyota propone que la investigación y desarrollo se verá beneficiada al utilizar una plataforma robótica común para todos los investigadores ya que así se podrán compartir los resultados de sus investigaciones. Por este motivo desarrollaron una plataforma compacta y segura para la investigación, el *Human Support Robot* (HSR), el cual pude ser operado en un ambiente hogareño real. Además, Toyota ha proporcionado este robot a varias instituciones de investigación para formar una comunidad de desarrolladores de HSR [18].

Entre las instituciones que tienen convenio con Toyota para utilizar el HSR se encuentra el Laboratorio de BioRobótica del Posgrado de Ingeniería de la UNAM. El robot HSR se recibió en febrero de 2018 y desde entonces se formó un equipo (del que formo parte) para desarrollar las funcionalidades necesarias para operar este robot. Nuestro robot HSR fue bautizado con el nombre de **"Takeshi"** (figura 2.1) y junto con nuestro otro robot hecho en casa **"Justina"** han participado y ganado varios premios en competencias nacionales e internacionales. Las características más importantes de este robot se describen en el Apéndice A.

Figura 2.1: Robot HSR del Laboratorio de BioRobótica (Takeshi)

2.2. Competencias de robots

Una forma de impulsar el desarrollo de la robótica es a través de los concursos de robots. La idea es plantear problemas difíciles que sean un reto para los especialistas en robótica, de forma que al ir resolviendo estos retos se avanza en el desarrollo de robots, haciéndolos cada vez más poderosos e inteligentes y con esto ir logrando realizar tareas cada vez más complejas. Uno de los torneos de robótica más importantes del mundo es la RoboCup [19], que surgió hace 20 años en Japón, y se realiza cada año en diversas partes del mundo [20]. Los fundadores de la RoboCup plantearon un gran reto a largo plazo: que un equipo de robots futbolistas venza al campeón mundial de la FIFA en el año 2050 (*"By the middle of the 21st century, a team of fully autonomous humanoid robot soccer players shall win a soccer game, complying with the official rules of FIFA, against the winner of the most recent World Cup"*) [21]. La idea es que, aunque no se logre cumplir con esta meta, todos los avances que se realicen en el camino hacia la misma tengan un impacto importante en el desarrollo de la robótica y sus aplicaciones en general.

Con el paso de los años el ámbito de las competencias de robots se ha extendido hacia otras aplicaciones (además de fútbol) como robots para investigación urbana y rescate, robots de servicio doméstico y robots móviles en aplicaciones industriales, por lo cual además de competencias de robots futbolistas, la RoboCup incluye otras competencias como la competencia de RoboCup@Home, encaminada al desarrollo de los robots que puedan ayudar en las casas, así como también competencias para juniors, que buscan despertar el interés en la robótica en niños y jóvenes.

En 2006 se llevó a cabo la primera competencia RoboCup@Home, esta es una liga que se enfoca en el desarrollo de robots de servicio doméstico. En ella, robots de servicio personales tienen que realizar quehaceres y tareas en ambientes hogareños que van desde encontrar objetos hasta mezclar bebidas y cocinar o ayudar a hacer las compras. La competencia RoboCup@Home tiene como objetivo desarrollar tecnologías para robots de servicio y asistencia, los cuales son de gran relevancia para el futuro desarrollo de aplicaciones domésticas personales (figura 2.2).

Figura 2.2: Competencia RoboCup@Home

Esta es la más grande competencia internacional anual de robots de servicio autónomos. En un futuro no muy lejano, el éxito principal de los robots será la interacción cercana y efectiva entre humanos y robots, por lo que, además de seguir desarrollando sus capacidades autónomas, también debemos poner atención a desarrollar y mejorar la relación *humano-robot*. El reto es no solamente crear técnicas que le permitan a los robots ser efectivos en tareas limitadas o específicas sino encontrar la forma en la que puedan participar en la completa y compleja sociedad humana.

2.3. Simuladores para robótica

La interacción entre humanos y robots se ha convertido en tema interdisciplinario de investigación y desarrollo, adicionalmente la investigación en robótica no es fácil ni económica, por esta razón se ha optado por utilizar simulación.

Los simuladores son herramientas creadas para replicar aplicaciones robóticas del mundo real lo más fielmente posible, teniendo en cuenta todos los factores ambientales y físicos y probando todas las variables posibles. Un simulador de robótica se utiliza para crear una aplicación para un robot físico sin depender de la máquina real, ahorrando así costos y tiempo. En algunos casos, estas aplicaciones se pueden transferir al robot físico sin modificaciones.

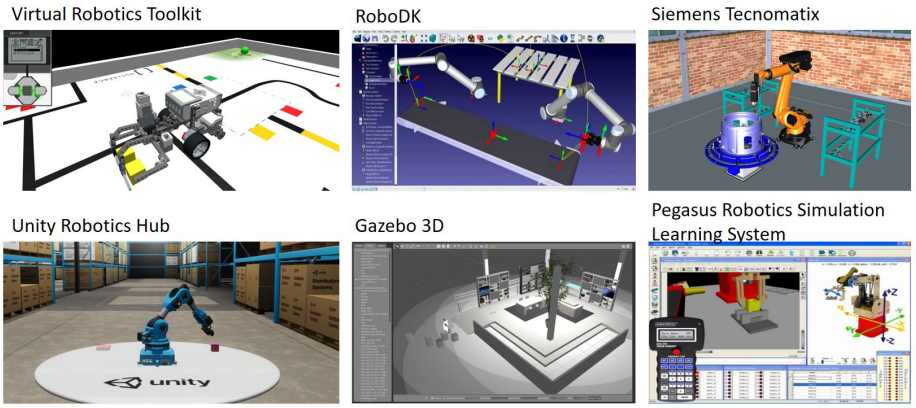


Figura 2.3: Algunos simuladores para robótica

Los simuladores de robots vienen en muchas formas (figura 2.3). Algunos sólo permiten la simulación 2D simple de aspectos específicos de la robótica, mientras que otros incluyen la simulación de entornos 3D realistas con fenómenos físicos complejos (como gravedad y fricción). La mayoría del software de simulación de robots es compatible con lenguajes de programación comunes como *C ++*, *Python*, *Java*, *MATLAB*, etc. por lo que son bastante accesibles. El simulador permite que los programas de robótica se escriban y depuren convenientemente fuera de línea con la versión final del programa probada en un robot real. El objetivo es solucionar cualquier defecto de diseño antes de instalar el sistema en el robot real. Una de las aplicaciones más populares para los simuladores de robótica es para el modelado y renderizado 3D de un robot y su entorno. Este tipo de software tiene un simulador con un robot virtual, que es capaz de emular el movimiento de un robot real en un entorno de trabajo real. Algunos simuladores de robótica utilizan un motor de física para generar movimientos más realistas del robot.

La simulación de robots ofrece diversos beneficios, por ejemplo, pruebas de concepto y diseño, las cuales son útiles para que los defectos no se incorporen a los sistemas robóticos. Este beneficio se obtiene cada vez que un robot procesa con éxito una tarea. La prueba de concepto y diseño garantiza que se instale el mejor sistema y que funcione según lo previsto. Otro beneficio es la reducción de costos, ya que la simulación ayuda a reducir el costo total de integración. Prin-

cipalmente, lo hace a través de la capacidad de simular aplicaciones del mundo real sin los costos físicos de hacerlo. La simulación de robot también reduce, y la mayoría de las veces incluso elimina, la necesidad de realizar ajustes en el sistema después de su instalación, ya que se ha demostrado que los procesos funcionan. Otro beneficio es la reducción de los tiempos de desarrollo, esto es dado que ayuda en la programación fuera de línea, donde la programación de un robot se completa antes de su instalación. En muchos casos, esto significa que el robot puede comenzar a realizar sus tareas el día en que se implementa, lo que acorta significativamente el tiempo de entrega general del sistema.

Por otro lado, con los simuladores se pueden automatizar pruebas y ejecutarlas un mayor número de veces, lo cual en un ambiente real implicaría tener que replicar manualmente las condiciones iniciales del ambiente muchas veces, esto, además de consumir mucho más tiempo, no garantiza que las condiciones sean exactamente las mismas y por tanto las pruebas no sean del todo confiables. Adicionalmente, se puede desarrollar aún sin tener el robot físicamente, por ejemplo, si por el costo de un robot no se dispone de él o no se tienen suficientes robots para todos los desarrolladores, se puede utilizar el simulador y posteriormente probarse en el robot físico la versión final del software.

En fechas recientes se ha incrementado el uso de estas herramientas. Por ejemplo, para desarrollar sistemas robóticos sin requerir del robot físico o desde casa. En el año 2020, con el inicio de la pandemia, las competencias nacionales e internacionales fueron pospuestas o suspendidas, sin embargo, en 2021 se han habilitado tecnologías para realizar las competencias remotas usando simuladores online y así no detener el desarrollo en el área. Algunas de las competencias que utilizaron esta tecnología son el Torneo Mexicano de Robótica [22] y el RoboCup 2021 [23]. Derivado de esto, se planea agregar una nueva liga en el RoboCup @Home llamada Simulation para el 2022 debido al éxito obtenido en 2021 de los simuladores online. En resumen, el campo de la robótica de servicio se ve ampliamente beneficiado por los simuladores como herramienta de visualización y de desarrollo, esto conlleva a un avance importante en la implementación de nuevas tecnologías al servicio de los humanos.

Existen actualmente varios simuladores para robótica, algunos de los cuales se enfocan a los comportamientos físicos (OpenHRP [24], Webots [25], The Player Project (Stage/Gazebo) [26]) y otros dedicados a los multi agentes que se centran en el efecto colectivo de un gran grupo de agentes autónomos (*agent-based model* [27]), sin embargo, en el contexto de interacción *humano-robot* no hay muchas opciones.

El simulador SIGVerse [28] es un ambiente virtual tridimensional basado en *Unity Engine* que permite simular interacciones entre agentes y sus ambientes (figura 2.4). En particular, SIGVerse permite simular interacciones de los robots (percepción y acción) en ambientes 3D. SIGVerse cuenta con características específicamente diseñadas para cubrir las necesidades en el desarrollo de sistemas

de interacción *humano-robot*, por lo que ofrece la mayoría de los elementos fundamentales para el modelado y simulación de robots.

Figura 2.4: Simulador SIGVerse

SIGVerse emula de manera destacada todos los componentes como navegación, colisiones, movimientos de partes del robot, cámaras y sensores, además de emular algunos aspectos de los ambientes reales como ruido, fricción, etc. Por otro lado, también se incorporan cuestiones de percepción para los agentes (sensores de visión, de audio y táctil, por ejemplo) para hacer más realistas las simulaciones. Además, cuenta con un modelado de robots y agentes humanos (avatares) con los cuales además se puede interactuar de manera verbal o bien por medio de gestos y movimientos, así como también se pueden enviar mensajes escritos por medio de su interfaz gráfica.

El sistema SIGVerse fue y sigue siendo desarrollado por el Dr. Tetsanuri Inamura del National Insitute of Informatics y es apoyado por varios colaboradores; es un simulador de realidad virtual gratuito basado en la nube, totalmente compatible con ROS (*Robot Operating System*) [29], fácil de usar y que permite personalización y mejoras. La última versión disponible [30] es la 3.0 la cual incorpora algunos cambios al utilizar Unity para la representación virtual. SIGVerse es un sistema de cliente/servidor que se basa en una tecnología de red. El servidor y los clientes comparten la misma escena, que se compone de modelos de objetos 3D como avatares, robots y muebles. Al transferir la información de los objetos registrados a través de Internet, los eventos de cada escena se pueden sincronizar. Los detalles de su arquitectura se describen en el Apéndice B.

Hace relativamente pocos años las competencias virtuales comenzaron como una sub categoría dentro de las ligas de las competencias internacionales, por

ejemplo, en 2018 en el World Robot Summit (WRS) en Japón [31], dentro de la Categoría de robots de servicio, se incluyó el Partner Robot Challenge/Virtual Space (figura 2.5).

Figura 2.5: World Robot Summit 2018 Virtual Space

Dado que SIGVerse es considerado un simulador estandarizado para HSR fue el simulador elegido para este evento en el cual se simularon varios ambientes hogareños donde un robot virtual realizaba tareas de manipulación como parte de las tareas de traer y colocar. También se utilizaron otras características del SIGVerse como la interacción con los cascos de realidad virtual, en esta prueba el robot le da instrucciones al humano (de voz por medio de los auriculares y de texto en los lentes del casco), y el humano tiene que seguir las instrucciones del robot y mover a su avatar dentro del ambiente virtual para completar la tarea. En este evento se tuvieron muy buenos resultados usando el simulador. Por todas sus características y resultados se eligió SIGVerse para el presente proyecto.

2.4. Teleoperación

Los robots de servicio pueden realizar tareas más complejas y precisas aprovechando la experiencia humana por medio de la teleoperación. Los sistemas teleoperados pretenden proporcionar medios técnicos para realizar las tareas deseadas en un ambiente remoto [32]. En un sistema de teleoperación eficiente, el operador debe poder asegurarse de que la tarea deseada es realizada adecuadamente en el ambiente remoto. Para tal fin, los sistemas de teleoperación deben superar una serie de barreras como la distancia, el tiempo de retardo, entre otros. Adicionalmente, la relación entre el operador y el ambiente remoto

afecta grandemente la eficiencia del sistema, esto significa que el desempeño del sistema puede ser ampliamente mejorado al permitir que el operador comprenda la interacción con el ambiente remoto de manera fácil e intuitiva.

Idealmente, el sistema debe hacer que el operador sienta como si el ambiente remoto fuera el mundo real. Existe la creencia de que la presencia o personificación está positivamente relacionada con el desempeño de las tareas [33], entonces, al aumentar el sentimiento de presencia, el desempeño en las tareas también se mejora. Además de mejorar el desempeño, el sentimiento de presencia también le da al operador la habilidad de realizar las tareas de una forma más natural, similar a las acciones del mundo real. Esto puede reducir la necesidad del operador de entrenar extensivamente para lograr operar exitosamente el robot.

Recientemente las interfaces inmersivas se están utilizando para inducir la sensación de estar dentro del robot [34] y con esto mejorar la experiencia en teleoperación [35]. La sensación de presencia es resultado de la inmersión [36], por lo tanto, las interfaces inmersivas se acompañan generalmente con dispositivos de visualización como los HMD (*Head Mounted Display*) los cuales consisten en un display combinado con un sistema de seguimiento de cabeza que mejora la inmersión y el control. Un método típico de proporcionar el sentimiento de presencia es mostrarle al operador el video de las cámaras del robot [37, 38], sin embargo, recientemente la Realidad Virtual (RV) ha probado ser una excelente opción para mostrar información del ambiente remoto al operador. Los sistemas de RV se están usando en la teleoperación de robots debido a su habilidad de permitir que los usuarios interactúen intuitivamente con los ambientes 3D. Gradecki define la RV como "una tecnología que permite al usuario visualizar un ambiente virtual desde cualquier punto y ángulo e interactuar con los objetos que componen dicho ambiente" [39]. Al mejorar la interface de teleoperación con ambientes virtuales, el operador puede experimentar mejor control del robot, permite que usuarios inexpertos controlen el robot, y aprovecha la experiencia de usuarios expertos en ambientes desafiantes [40].

Se tienen muchos beneficios al mejorar la interface de teleoperación con ambientes virtuales, sin embargo, la integración de robots con sistemas de RV es una tarea difícil, dado que se deben considerar varios factores, por ejemplo, la falta de interfaces estándar para conectar los sistemas operativos robóticos como ROS (*Robot Operating System*) con plataformas de realidad virtual que utilicen dispositivos HMD [41]. Por otro lado, se debe lidiar con los malestares causados por la RV como dolores de cabeza, mareos y náuseas ocasionados por el uso incorrecto de ambientes de RV [42].

El Internet esta volviéndose el medio de comunicación más común en teleoperación [43] dado que ofrece muchas ventajas como la facilidad de uso, la amplia accesibilidad y bajo costo y relativa confiabilidad. Pero a pesar de esos beneficios, esta tecnología de comunicaciones tiene sus desventajas, como la la-

tencia, la perdida de paquetes y otros [44]. Dado que la visualización remota involucra transmitir una gran cantidad de datos de video por la red, el reto es disminuir la inestabilidad en la comunicación, como la latencia o retraso en el tiempo, la cual es uno de los mayores problemas en teleoperación [37]. El retraso en el tiempo entre la entrada y la respuesta visual afecta grandemente la comunicación entre el maestro y esclavo en sistemas distribuidos sobre una red [45]. Este retraso puede variar desde unos pocos milisegundos a varios cientos de milisegundos, dependiendo de diferentes factores como la distancia o la infraestructura de comunicaciones. Los primeros experimentos para probar los efectos del retraso en el tiempo en teleoperación mostraron que la latencia deteriora el desempeño de la operación del robot [46]. Estudios más recientes mostraron que en un sistema con latencia mayor a 1 segundo, el operador generalmente trata de compensar el retraso con una estrategia de "mover y esperar" [47] lo cual hace que la operación del robot no sea óptima. Muchos esfuerzos se han hecho por reducir la latencia, como los algoritmos de comprensión de datos y optimización.

En resumen, un sistema de teleoperación debe tener una interfaz de operación adecuada que sea no solo capaz de visualizar el ambiente remoto, controlar al robot y proveer un sentimiento de presencia si no también tener una respuesta rápida en el tiempo.

En el laboratorio de BioRobótica se desarrolló un sistema de teleoperación de robots de servicio con una interfaz de realidad mixta donde el operador puede visualizar el ambiente real del robot o un ambiente virtual en 3D representándolo [48]. La realidad mixta es el concepto (no una tecnología particular) de combinar el mundo real y virtual [49]. Involucra mezclar el mundo real y el virtual en algún punto dentro del "continuo de la virtualidad" (*virtuality continuum*) el cual conecta el mundo completamente real con el ambiente completamente virtual [50] (figura 2.6).

Figura 2.6: Continuo de la Virtualidad [50]

El objetivo de agregarle un ambiente virtual es reducir la latencia en la comunicación al reducir la cantidad de información enviada por la red y mejorar la experiencia del usuario. Sin embargo, agregar el ambiente virtual ocasionó la necesidad de asegurarse de que la tarea se estuviera realizando adecuadamente, entonces la visualización del ambiente real por medio de las cámaras del robot

también se requería. Para solucionar esto, se agregó un mecanismo para cambiar entre los modos de visualización real y virtual. El diseño y desarrollo de este sistema de teleoperación requirió considerar diversos aspectos como tener una interfaz de operación amigable al usuario, lidiar con los medios de comunicación (como la transmisión, los protocolos y los problemas de retraso en el tiempo o latencia), crear un ambiente virtual personalizado fiel al ambiente real, seleccionar los dispositivos de realidad virtual adecuados, así como probar con operadores humanos.

El sistema permite a usuarios cotidianos efectuar tareas de navegación sin necesidad de entrenamiento intensivo. Por lo tanto, el robot se puede controlar remotamente a través de su interface de operación y el operador puede moverse o navegar el robot libremente en el ambiente remoto. Adicionalmente, el operador puede ver el ambiente remoto a través de las cámaras estereoscópicas del robot con un efecto 3D en el HMD.

Para lidiar con los problemas de latencia y prevenir que el operador tenga una experiencia de usuario desagradable, se puede cambiar el modo de visualización de 3D real a 3D virtual y continuar operando el robot en el ambiente virtual sin gran diferencia. Uno de los objetivos de esta visualización dual es que el usuario pueda navegar exitosamente en el ambiente remoto usando los modos real y virtual indistintamente sin problemas como mareos, colisiones con objetos o dañar componentes del robot o del ambiente remoto. Otro punto crucial a considerar son las condiciones de iluminación. El robot debería ser capaz de operar normalmente aun cuando las condiciones de iluminación cambian. Esto significa que, si el ambiente remoto se obscurece, la visualización real se vuelve inapropiada para continuar operando dado que el operador no podría visualizar el ambiente. Cuando se cambia al modo virtual, este cambio en la iluminación no debería afectar a la visualización. Sin embargo, el operador aún debe poder percibir las nuevas condiciones de luz para preservar el sentimiento de presencia y mejorar la experiencia de usuario.

El sistema desarrollado contiene una interfaz de operación con dos modos de visualización: el video real estéreo de las cámaras del robot y el ambiente virtual 3D mapeando al ambiente real (figura 2.7). La interfaz se muestra en un HMD para poner al operador en un espacio virtual que contiene ambos modos de visualización. Este HMD también controla los movimientos de la cabeza del robot al seguir los movimientos de la cabeza del operador. Para navegar y otras funciones, se utiliza un *joystick*. El robot de servicio puede realizar navegación y tareas simples autónomamente o puede cambiar a modo de teleoperación cuando se navega en lugares difíciles o cuando se realizan tareas complejas.

Sin embargo, este sistema aún tiene algunas oportunidades para mejoras y nuevas funcionalidades, por ejemplo, la habilidad de colocar objetos autónomamente. Esta tarea requiere de movimientos finos que son difíciles de ejecutar con teleoperación en el modo de visualización real (debido a la latencia) y son

Figura 2.7: Sistema de teleoperación con realidad mixta

prácticamente imposibles de realizar en el modo virtual ya que las condiciones y reacciones de objetos dinámicos reales (que no son fijos en el ambiente) no se reflejan automáticamente en el ambiente virtual dado que son complicados de predecir, y es en ésta área donde contribuye el trabajo de esta tesis.

2.5. Colocación de objetos

Mientras que ha habido significativamente mayor cantidad de trabajos previos en agarre de objetos (*grasping*), aún son relativamente pocos los trabajos realizados en colocación (*placing*). Decidir dónde poner un objeto o planeación de colocación es una tarea de razonamiento crucial, además la planeación de colocación es específicamente difícil en ambientes 3D.

Lozano-Perez et al [51] fueron los pioneros en proponer el problema de planeación de movimientos y agarre de objetos y en la siguiente década este problema se continuó investigando extensivamente [52, 53, 54, 55, 56, 57, 58]. Sin embargo, no ha habido mucha investigación en la planificación de colocación de objetos. Katz et al. [59] señalaron las tecnologías clave para los robots que trabajan en un entorno no estructurado y mencionaron el planificador de colocación de objetos. Berenson et al. [54] planearon la postura de agarre para una mano con múltiples dedos tomando en consideración la postura de colocación de un objeto. En este caso la postura de colocación de un objeto se determina

antes de planificar el movimiento de recoger y colocar (*pick-and-place*). Schuster et al. [15] y Jiang et al. [14] utilizaron la nube de puntos durante las tareas de recoger y colocar para identificar un área plana del entorno.

Para colocación robótica, un componente es planear y controlar el brazo para colocar el objeto sin tirarlo usando control de fuerzas o planeación a nivel de tareas [60, 17, 61, 62]. En cuanto a planeación de colocación de objetos, Harada et al. [63] proponen un enfoque geométrico estático el cual genera múltiples candidatos de las posturas de los objetos colocados en el entorno en función del análisis geométrico entre un objeto y el entorno. Otro tema relacionado es la planeación de empuje de objetos en áreas de colocación planas, aquí se propone empujar otros objetos para generar espacio suficiente para colocar [64].

Una de las primeras soluciones generales para colocar objetos novedosos en áreas de colocación complejas fue presentado por Jiang et al. [14, 65], ellos describen un *framework* basado en algoritmos de aprendizaje para calcular las posiciones óptimas para la colocación en áreas previamente conocidas que requiere una base de datos junto con etiquetas semánticas. Baumgartl et al. [66] difieren del trabajo de Jiang en la medida en que sólo utilizan información geométrica sobre el objeto y no utilizan ninguna etapa de aprendizaje. Otro enfoque es el de Gillespie et al. [67] donde proponen aprender automáticamente de casos en sesiones de resolución de problemas y reutilizarlos para resolver problemas de planeación de colocación nunca antes vistos.

Todas estas propuestas tienen puntos buenos y debilidades, sin embargo, la gran mayoría de ellas se enfoca en planeación, ya sea de movimientos, acciones, posturas, etc. lo cual suele demandar recursos computacionales importantes. Algunos de estos métodos también requieren control de fuerzas o planeación conjunta de posturas y acciones, etc. otros realizan análisis geométrico de objetos y entornos para determinar las áreas candidato para colocación y algunos más implican predicción de efectos al empujar objetos (fuerzas físicas).

Todas estas técnicas si bien parecen ser exitosas, requieren grandes recursos de cómputo para realizar todos los cálculos y ejecutar algoritmos y/o procesos involucrados, lo que también suele consumir bastante tiempo. En contraste, el método propuesto elimina la necesidad de cálculos y/o planeación al tener el conjunto de estrategias predeterminadas y probadas de atemano, pero sin perder su capacidad para generalizar a situaciones nuevas dentro de ciertos parámetros, esto además de disminuir los recursos necesarios también reduce el tiempo de ejecución, lo cual supone un mejor desempeño que los mencionados anteriormente.

2.6. Brecha de la realidad

Todos los seres humanos pueden aprender habilidades notablemente complejas que superan con creces la competencia y robustez de los robots más sofisticados cuando se trata de habilidades sensomotoras básicas como tomar o colocar objetos, esto se da gracias a la experiencia de aprender a lo largo de varios años cómo interactuar con el mundo que nos rodea. En cambio, para un robot, requerir tal experiencia de por vida basado en el aprendizaje es bastante difícil, el robot necesitaría operar por mucho tiempo de manera continua, autónoma e inicialmente con un bajo nivel de competencia antes de que pudiera volverse útil. Afortunadamente existe una herramienta a su disposición, los simuladores.

Es bastante factible simular muchos años de interacción robótica usando la tecnología moderna de procesamiento, simulación física y computación paralela. Adicionalmente, los resultados pueden tener anotaciones generadas automáticamente, lo cual es particularmente importante para tareas en las que el éxito es difícil de inferir. El desafío del entrenamiento simulado es que incluso los mejores simuladores disponibles no capturan perfectamente la realidad. Los modelos entrenados puramente en datos sintéticos no se generalizan al mundo real, ya que existe una discrepancia entre los entornos simulados y reales, en términos de propiedades tanto visuales como físicas. Cuanto más se aumenta la fidelidad de las simulaciones, más esfuerzo se debe dedicar para construirlas, tanto en términos de implementación de fenómenos físicos complejos como en términos de creación de contenido (por ejemplo, objetos, fondos) para poblar estas simulaciones. Esta dificultad se ve agravada por el hecho de que los poderosos métodos de optimización basados en el aprendizaje profundo son excepcionalmente competentes para explotar las fallas del simulador, es decir, cuanto más poderoso es el algoritmo de aprendizaje automático, más probabilidades hay de descubrir cómo "engañar" al simulador para que tenga éxito en formas que son inviables en el mundo real. La pregunta entonces es ¿cómo puede un robot utilizar la simulación para permitirle realizar tareas útiles en el mundo real?

La dificultad de transferir la experiencia simulada al mundo real a menudo se denomina Brecha de la Realidad (*Reality Gap*). La brecha de la realidad es una discrepancia sutil pero importante entre la realidad y la simulación que evita que la experiencia robótica simulada permita directamente un desempeño efectivo en el mundo real. La percepción visual a menudo constituye la parte más amplia de la brecha de la realidad ya que mientras que las imágenes simuladas continúan mejorando en fidelidad, las regularidades peculiares y patológicas de las imágenes sintéticas y la amplia e impredecible diversidad de imágenes del mundo real, hacen que cerrar la brecha de la realidad sea particularmente difícil cuando el robot debe utilizar la visión para percibir el mundo, como es el caso, por ejemplo, de muchas tareas de manipulación.

Los avances recientes para cerrar la brecha de la realidad con el aprendizaje profundo en visión por computadora para tareas como la clasificación de ob-

jetos y la estimación de poses brindan soluciones prometedoras. Por ejemplo, Shrivastava [68] y Bousmalis [69] exploraron la adaptación del dominio a nivel de píxel. Ganin[70], Bousmalis y Trigeorgis [71] se centraron en la adaptación del dominio a nivel de función. Estos avances requirieron repensar los enfoques utilizados para resolver el problema del cambio de dominio de la simulación a la realidad para la manipulación robótica también. Aunque varios trabajos recientes han buscado abordar la brecha de realidad en la robótica, a través de técnicas como la adaptación de dominio basada en el aprendizaje automático [72] y la aleatorización de entornos simulados [73], la transferencia efectiva en la manipulación robótica se ha limitado a tareas relativamente simples, como tomar objetos rectangulares de colores brillantes [74, 75] y movimiento en el espacio libre [76].

Antes de considerar la introducción de la experiencia simulada, ¿qué se necesita para que los robots aprendan a captar de manera confiable tales objetos nunca antes vistos con solo la experiencia del mundo virtual? En general se necesitan decenas a cientos de miles de intentos de comprensión, el equivalente a miles de horas de robot de experiencia en el mundo real. Aunque distribuir el aprendizaje entre varios robots acelera esto, las realidades de la recopilación de datos del mundo real, incluido el mantenimiento y el desgaste, significan que este tipo de esfuerzos de recopilación de datos aún requieren una cantidad significativa de tiempo real. Una alternativa atractiva es usar simuladores estándar y aprender habilidades sensomotoras básicas como tomar objetos en un entorno virtual. Entrenar a un robot sobre cómo captar la simulación se puede paralelizar fácilmente en cualquier número de máquinas y puede proporcionar una gran cantidad de experiencia en mucho menos tiempo (por ejemplo, horas en lugar de meses) y a una fracción del costo.

Si el objetivo es cerrar la brecha de la realidad para la manipulación robótica basada en la visión, se deben responder algunas preguntas críticas. Primero, ¿cómo diseñar la simulación para que la experiencia simulada parezca realista para una red neuronal? Y segundo, ¿cómo se debería integrar la experiencia real y la simulada de manera que maximice la transferencia al mundo real?

Capítulo 3

Aprendizaje profundo

3.1. Breve introducción

El aprendizaje profundo (*deep learning*) es un tipo de aprendizaje automático (*machine learning*) e inteligencia artificial (*artificial intelligence*) que imita la forma en que los humanos obtienen ciertos tipos de conocimiento, en otras palabras, se entrena a una computadora para que realice tareas de manera similar a los seres humanos, por ejemplo, el reconocimiento del habla, la identificación de imágenes o hacer predicciones. El aprendizaje profundo configura parámetros básicos sobre los datos y entrena a la computadora para que aprenda reconociendo patrones mediante muchas capas de procesamiento. El aprendizaje profundo ha mejorado la capacidad de las computadoras de clasificar, reconocer, detectar y describir, en resumen, de entender.

El aprendizaje profundo suele denominarse como aprendizaje neuronal profundo o redes neuronales profundas dado que muchos de los modelos de aprendizaje profundo son basados en las redes neuronales artificiales. Existen varios tipos de redes neuronales, como las redes neuronales convolucionales, las redes neuronales recurrentes, las redes neuronales de retroalimentación, etc. Cada una de ellas tiene aplicaciones y beneficios para diferentes casos de uso, pero todas coinciden en su funcionamiento base: introducir datos y dejar que el modelo averigüe por sí mismo si ha tomado la interpretación o decisión correcta sobre un elemento de datos dado.

Para entrenar una red neuronal se requieren cantidades masivas de datos dado que implican un proceso de prueba y error, ésta es la razón de que a partir de que la mayoría de las empresas adoptaron el análisis de big data y acumularon grandes cantidades de datos, las redes neuronales comenzaron a volverse populares rápidamente. Además, los datos usados durante la etapa de entrenamiento deben etiquetarse para que el modelo pueda ver si su suposición fue precisa. Entonces, aunque muchas empresas que utilizan big data tengan grandes canti-

dades de datos, no todos son útiles para entrenar modelos de redes neuronales profundas, solo los datos estructurados lo son.

La idea de utilizar el aprendizaje profundo requiere que los humanos estén dispuestos a perder el control de cierta manera, esto puede parecer contradictorio en un inicio, sin embargo, la ganancia obtenida es permitir que el sistema comience a aprender por si solo [77]. Esto hace que las redes neuronales profundas sean muy adecuadas para ser utilizadas con robots, ya que son flexibles y se pueden usar en *frameworks* que no pueden ser compatibles con otros modelos de aprendizaje automático [78]. Cada uno de los distintos tipos de modelos de aprendizaje profundo se componen de un conjunto de varias capas apiladas de modelos de regresión [79]. Dentro de estos modelos, distintos tipos de capas han experimentado evolución para muchos objetivos. Los investigadores en robótica están buscando opciones creativas, por ejemplo, aprovechando los datos de entrenamiento a través de la manipulación digital, entrenamiento automatizado y el uso de múltiples redes neuronales profundas para mejorar el rendimiento y reducir el tiempo de entrenamiento [80].

Un tipo de capa que requiere especial atención son las capas convolucionales (*convolutional layers*). A diferencia de las capas tradicionales que están completamente conectadas (*fully connected layers*), las capas convolucionales aplican los mismos pesos para operar en el espacio de entrada agrupado en subregiones rectangulares, esto provoca una reducción significativa del número total de pesos en la red neuronal, que es especialmente vital con imágenes que normalmente se componen de cientos de miles y millones de píxeles que requieren procesamiento [81]. La inspiración de las capas convolucionales provino de las neuronas corticales dentro de la corteza visual que sólo responden a los estímulos en un ambiente receptivo. Dado que la convolución simula dicho comportamiento, se puede esperar que las capas convolucionales sobresalgan en las aplicaciones de procesamiento de imágenes [82]. Actualmente, las redes neuronales convolucionales (CNN, del inglés *convolutional neural networks*) se han hecho conocidas y altamente efectivas como un modelo de aprendizaje profundo para muchas aplicaciones basadas en imágenes (figura 3.1). Estas aplicaciones comprenden: segmentación semántica de imágenes, escalado de imágenes mediante súper resolución, reconocimiento de escena, localización de objetos con imágenes, reconocimiento de gestos humanos y reconocimiento facial [83] [84].

Las imágenes no son la única forma de señal que ilustra la excelencia de las redes neuronales convolucionales. Su capacidad también es efectiva en cualquier forma de señal que demuestre proximidad *espacio-temporal*, por ejemplo, reconocimiento de voz, así como síntesis de voz y audio [86], naturalmente estos también han comenzado a ser dominantes en el ámbito del procesamiento de señales y muy utilizados en robótica. Los proyectos recientes incluso han comenzado a integrar señales de varias modalidades y combinarlas para un reconocimiento y percepción unificados. En última instancia, la filosofía que subyace y prevalece en la comunidad de aprendizaje profundo es que cada componente de un sistema

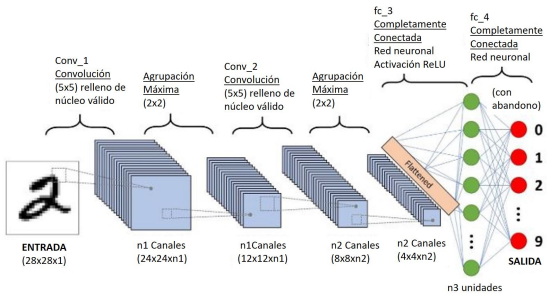


Figura 3.1: CNN para clasificar dígitos escritos a mano (Imagen tomada y traducida de [85])

complejo puede ser enseñado a "aprender". Por lo tanto, el poder real del aprendizaje profundo no proviene de aplicar sólo una estructura como parte de los sistemas de robótica si no al conectar componentes de todas estas estructuras para formar un sistema que aprende por completo [87]. Este es el punto donde el aprendizaje profundo comienza a tener un impacto tal que cada componente del sistema es capaz de aprender como un todo y es capaz de adaptarse a métodos sofisticados.

En términos de reconocimiento avanzado de objetos, las redes neuronales profundas han demostrado adaptarse cada vez más al reconocimiento y clasificación de objetos. Los ejemplos de aplicación avanzada incluyen el reconocimiento de objetos deformados y la estimación de su estado y pose para movimiento, tarea semántica y especificación de ruta, por ejemplo, moverse alrededor de la mesa [88]. Además, incluye el reconocimiento de los atributos de un objeto y una superficie por los cuales, por ejemplo, un objeto afilado podría presentar un peligro para los colaboradores humanos en ciertos entornos. Ante tales dificultades, los modelos de aprendizaje profundo se pueden usar en la aproximación de funciones de muestras de pares *entrada-salida*. Estas pueden ser las estructuras de aprendizaje profundo de propósito más general ya que existen varias funciones distintas en robótica que los investigadores pueden usar para aproximarse a partir de observaciones muestra [89].

Ciertos ejemplos de estas observaciones implican el mapeo de las acciones a los cambios correspondientes en el estado, el mapeo de estos cambios de estado a las acciones que pueden causarlo o el mapeo de la fuerza al movimiento. Si bien en ciertos casos las ecuaciones físicas particulares para tales funciones ya pueden estar definidas, hay varias situaciones en las que el entorno es altamente complejo para que dichas ecuaciones generen una precisión aceptable [90]. Sin embargo, en tales escenarios, el aprendizaje de la aproximación de funciones a partir de observaciones muestra puede proporcionar una precisión significativamente mejor. En otras palabras, las funciones aproximadas no necesitan ser continuas.

Sin embargo, los modelos de aproximación de funciones también son excelentes en las tareas de clasificación, por ejemplo, determinar el tipo de obstáculos ante un robot, la estrategia general de planificación de ruta adecuada para los entornos actuales o el estado de un determinado objeto complejo con el que el objeto está interactuando. Las redes neuronales profundas han reemplazado a otros modelos en detección y percepción, ya que son capaces de participar en operaciones directas con entradas altamente dimensionales en lugar de necesitar vectores de características basados en diseños manuales hechos por humanos [91]. Esto reduce la dependencia de los humanos, de modo que el tiempo de entrenamiento adicional se puede compensar parcialmente al reducir los esfuerzos iniciales de ingeniería.

Los sistemas unificados limitan las interacciones subóptimas entre sistemas normalmente separados al predecir los resultados físicos de escenas dinámicas utilizando sólo la visión. Esto se basa en las acciones de los humanos para poder predecir los resultados de una escena dinámica a partir de información visual, por ejemplo, una roca que cae e impacta otra roca [92]. Por lo tanto, en esta premisa se ha identificado que el aprendizaje profundo es efectivo en la gestión de la generación de datos multimodales en aplicaciones de sensores robóticos. Estas aplicaciones incluyen la integración de datos de sensores de visión y hápticos, incorporando datos de profundidad e información de imágenes de datos de cámara RGB-D.

Debido a la gran cantidad de *meta-parámetros*, las redes neuronales profundas han ganado hasta cierto punto una reputación de ser un desafío para que los inexpertos puedan utilizarlos de manera efectiva. Sin embargo, estos parámetros también ofrecen una flexibilidad significativa, que es un factor vital en su éxito general. Específicamente, la aplicación de estas técnicas ayudará a abordar desafíos avanzados de reconocimiento de objetos [93]. En resumen, estos algoritmos de aprendizaje profundo se utilizan en robótica para fomentar la capacidad de funcionalidad de control para robots que aprenden indirectamente a través de entradas provenientes de las cámaras en el mundo real.

Muchas tareas físicas de los robots inteligentes, como aprender estrategias para tomar objetos, manipular objetos flexibles, y conducción automática entre otros, han sido ampliamente beneficiados por el reciente desarrollo de las técnicas de aprendizaje automático. El principio común de estas técnicas de aprendizaje de robots es que preparan ejemplos de movimientos del cuerpo y decisiones en forma de un conjunto de datos (*dataset*). Entre más complejo el problema, más grande es el conjunto de datos requerido, llegando a ser de miles o millones de datos, por lo que resulta impráctico obtener esta cantidad de datos en ambientes reales. Entonces, uno de los mayores retos en la investigación de la interacción *humano-robot* (HRI, del inglés *human-robot interaction*) es el proceso de recolección del conjunto de datos para el aprendizaje automático requerido para aprender y modelar actividades humanas.

Las investigaciones experimentales que involucran Realidad Virtual (RV) y simulaciones están ganando terreno como metodologías potenciales para reducir los costos de la recolección de datos. Además de lograr la deseable reducción de costos al recolectar datos, existen otras ventajas significativas de esta propuesta, por ejemplo: Reducción de costos, ya que no se requieren instalaciones reales para realizar los experimentos. Provee condiciones de interacción ambiental y encarnada similares a los sujetos de prueba. Visualización de información arbitraria y situaciones que no puede ocurrir en la realidad, como la reproducción de experiencias pasadas. Fácil acceso a una interfaz inmersiva y natural para los sistemas de teleoperación avatar/robot.

También es importante controlar únicamente las condiciones experimentales que se van a comparar, en lugar de lidiar con otros factores. Sin embargo, es un desafío unificar completamente tales condiciones experimentales en un entorno real. Como requisito previo para el diseño de experimentos en entornos reales, se requieren esfuerzos sustanciales para unificar de manera óptima las condiciones de iluminación y acústicas, así como el comportamiento humano, además de los otros participantes en el experimento. Este desafío de unificación y control de las condiciones se puede abordar utilizando la realidad virtual, lo que será beneficioso para garantizar la calidad de la investigación y también para determinar una línea de base clara para evaluar el rendimiento de los robots.

En los últimos años, se han adoptado sistemas de realidad virtual y el aprendizaje de habilidades basado en demostraciones de teleoperación para realizar activamente teleoperaciones con el objetivo de reducir costes. La realidad virtual y las simulaciones se utilizan a menudo para compensar la falta de información al operar robots en entornos extremos, dado que se espera que ocurran retrasos y fallas de comunicación. En este caso, es necesario predecir y visualizar el futuro mediante simulaciones. Por lo tanto, es necesario cooperar con sistemas de predicción que se diferencian de las interfaces de usuario de realidad virtual y los subsistemas de visualización.

3.2. Aprendizaje por transferencia

El aprendizaje por transferencia (*transfer learning*) no es un modelo o técnica de aprendizaje automático, es más bien una metodología de diseño dentro del aprendizaje automático. El aprendizaje por transferencia es esencial en cualquier tipo de aprendizaje. A los seres humanos no se les enseña a resolver todos los problemas existentes, pero cuando cada uno se encuentra en situaciones nunca antes vistas se las ingenia para resolver la situación de manera adecuada utilizando la habilidad de aprender de otras experiencias y aplicando ese conocimiento a nuevos escenarios, esto es la base del aprendizaje por transferencia. El aprendizaje por transferencia y la generalización son bastante similares en concepto, pero la principal diferencia es que el aprendizaje por

transferencia transfiere el conocimiento entre diferentes tareas en lugar de generalizar a una tarea específica. Sin embargo, el aprendizaje por transferencia esta intrínsecamente conectado con la idea de generalización que es necesaria en todos los modelos de aprendizaje automático. En un modelo de aprendizaje automático tradicional, el objetivo principal es generalizar datos no vistos anteriormente usando los patrones aprendidos con los datos de entrenamiento. Con el aprendizaje por transferencia, se intenta impulsar este proceso de generalización partiendo de patrones que se han aprendido para una tarea diferente. Esencialmente, en lugar de comenzar el proceso de aprendizaje desde ceros (o al azar), comienza con patrones que se han aprendido para resolver una tarea diferente.

Las técnicas de aprendizaje profundo requieren cantidades masivas de datos para ajustar los millones de parámetros en una red neuronal, especialmente en el caso del aprendizaje supervisado, esto significa que necesita una gran cantidad de datos etiquetados (lo cual es muy costoso). Etiquetar imágenes suena trivial, pero, por ejemplo, en el procesamiento del lenguaje natural (NLP, del inglés *natural language processing*), se requiere conocimiento experto para crear un gran conjunto de datos etiquetados. El aprendizaje por transferencia es una forma de reducir el tamaño requerido de los conjuntos de datos para que las redes neuronales sean una opción viable. La idea general del aprendizaje por transferencia es utilizar el conocimiento aprendido de tareas para las que hay muchos datos etiquetados disponibles, en entornos donde sólo hay pocos datos etiquetados. La creación de datos etiquetados es costosa, por lo que aprovechar de manera óptima los conjuntos de datos existentes es clave.

El aprendizaje por transferencia utiliza como punto de partida un modelo preentrenado de una red neuronal para entrenar otra red con conjunto de datos pequeño con muchas características similares al modelo anterior. Se utiliza un modelo neuronal entrenado para un conjunto de datos robusto, como ImageNet [94] o COCO [95], y se hace el reentrenamiento o ajuste fino (*fine-tuning*) de las últimas capas, las capas de clasificación. De esta manera, se tiene un nuevo modelo que aprovecha las características extraídas del modelo anterior para clasificar únicamente las nuevas clases de interés [96].

3.3. Detección de objetos usando redes neuronales

La clasificación de imágenes es una de las aplicaciones más populares de las redes neuronales, pero también existen otros problemas de visión computacional que se pueden resolver usando redes neuronales dentro de los cuales la detección de objetos es uno de los más interesantes. La detección de objetos se asocia comúnmente a los vehículos autónomos en cuyos sistemas se utilizan visión computacional y otras tecnologías para generar una representación mul-

tidimensional del camino con todas sus características. También es muy usada en video vigilancia, especialmente en monitoreo de multitudes para prevenir ataques terroristas, contar personas para propósitos estadísticos o analizar la experiencia del cliente al usar rutas dentro de centros comerciales.

Para comprender correctamente el concepto de detección de objetos, se debe comenzar por comprender la clasificación de imágenes y otros conceptos básicos. La clasificación de objetos pretende asignar una imagen a una o varias categorías (por ejemplo, perro, gato, humano, etc.) esencialmente responde a la pregunta *¿qué hay en esta imagen?* La clasificación es una tarea para determinar qué objetos están presentes en una imagen o video. Se trata de entrenar modelos para que encuentren cuáles clases de objetos están presentes, es decir, para decidir si una imagen contiene o no un objeto o característica dada. Otra tarea importante es la localización, es decir, determinar la posición del objeto clasificado en la imagen. La localización permite ubicar los objetos dentro de la imagen, entonces la pregunta cambia a *¿qué hay y dónde está?* La detección de objetos combina la clasificación y la localización para determinar qué objetos están en la imagen y especificar en qué parte de la imagen se encuentran. Aplica clasificación a los distintos objetos y utiliza la posición X e Y del objeto en la imagen (o su centro) para dibujar un rectángulo a su alrededor conocido como *bounding box*.

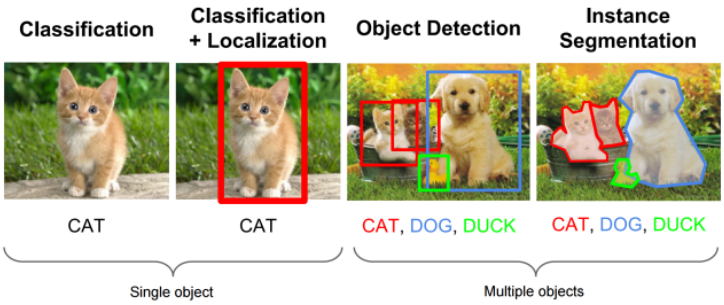


Figura 3.2: Clasificación, localización, detección de objetos y segmentación de instancias en imágenes (Imagen tomada de [97])

Los enfoques más populares de la visión por computadora son la clasificación y la detección de objetos, pero muchos casos de uso exigen analizar imágenes a un nivel inferior. Ahí es donde entra en juego la segmentación de imágenes. Cualquier imagen consta de información útil e inútil, según el interés del usuario. La segmentación de imágenes separa una imagen en regiones, cada una con su forma y borde particulares, delimitando áreas potencialmente significativas para su posterior procesamiento, como la clasificación y la detección de objetos. Es posible que las regiones no ocupen toda la imagen, pero el objetivo de la segmentación de imágenes es resaltar los elementos en primer plano y facilitar

su evaluación. La segmentación de imágenes proporciona detalles píxel por píxel de un objeto, lo que lo hace diferente de la clasificación y la detección de objetos. Ejemplos de estos conceptos se muestran en la figura 3.2.

Las redes neuronales convolucionales (CNN) representan un gran avance en el reconocimiento de imágenes. Son el algoritmo utilizado en aprendizaje automático principalmente en sistemas de detección y clasificación de objetos, segmentación de imágenes, procesamiento de lenguaje natural, etc. Las CNN se inspiran en procesos biológicos. Se basan en la investigación de Hubel y Wiesel [98] sobre la visión en gatos y monos. El patrón de conectividad en una CNN proviene de su investigación sobre la organización de la corteza visual. En el ojo de un mamífero, las neuronas individuales responden a los estímulos visuales sólo en el campo receptivo, que es una región restringida. Los campos receptivos de diferentes regiones se superponen parcialmente de modo que se cubre todo el campo de visión. En pocas palabras, las CNNs son un tipo de redes neuronales artificiales con aprendizaje supervisado que procesan sus capas imitando al córtex visual del ojo humano para identificar distintas características en las entradas que hacen que pueda identificar objetos.

Una CNN convoluciona características aprendidas con datos de entrada y utiliza capas convolucionales 2D. Esto significa que este tipo de red es ideal para procesar imágenes 2D. En comparación con otros algoritmos de clasificación de imágenes, las CNN en realidad utilizan muy poco procesamiento previo. Esto significa que pueden aprender los filtros que deben hacerse a mano en otros algoritmos. Las CNN utilizan la operación de convolución en lugar de una matriz de multiplicación en al menos alguna de sus capas [99], sin embargo, en algunos libros de aprendizaje profundo se implementa la función de correlación cruzada, pero la llaman convolución.

Las CNNs contienen una capa de entrada y una de salida y varias capas ocultas. Las capas ocultas generalmente consisten en capas convolucionales, capas de rectificación (*ReLU layer*), capas de reducción (*pooling layer*), y capas completamente conectadas (*fully conected*). Las capas convolucionales aplican una operación de convolución a la entrada. Esto pasa la información a la siguiente capa. Las capas de reducción combinan las salidas de grupos de neuronas en una sola neurona en la siguiente capa. Las capas de tipo completamente conectadas (*fully conected*) conectan cada neurona en una capa con cada neurona en la siguiente capa. Una arquitectura clásica de CNN se vería así:

Entrada –>Convolución –>ReLU –>Convolución –>ReLU –>Pooling –>ReLU –>Convolución –>ReLU –>Pooling –>Completamente conectada.

En la primera capa se presentan los datos de entrada (conjunto de pixeles de la imagen) y después se tienen una serie de capas ocultas que extraen características abstractas de la imagen (si se visualiza el contenido de las capas ocultas se pueden encontrar distintos patrones [99]). En la primera capa oculta

se encuentran los bordes de la imagen, en la segunda capa oculta se encuentran esquinas y contornos, y en la tercera capa oculta se pueden encontrar objetos específicos, formados por conjuntos de bordes y esquinas representativos. Por último, en la capa de salida se obtiene una descripción del objeto en términos de los patrones que contiene.

Una CNN funciona extrayendo características de las imágenes. Esto elimina la necesidad de extraer características manualmente, es decir, las características no son entrenadas si no que se aprenden mientras la red se entrena en un conjunto de imágenes. Esto hace que los modelos de aprendizaje profundo sean extremadamente precisos para tareas de visión por computadora. Las CNN aprenden a detectar características a través de decenas o cientos de capas ocultas. Cada capa aumenta la complejidad de las características aprendidas.

Una de las CNN más famosas y poderosas es ResNet [100], esta red neuronal profunda fue propuesta por Microsoft y venció en la competencia ILSVRC (*ImageNet Large Scale Visual Recognition Challenge*) en el año 2015. Desde entonces ResNet ha mostrando un destacado desempeño en tareas de reconocimiento.

La idea central de ResNet es aumentar el número de capas para mejorar su desempeño, sin embargo, simplemente aumentar la profundidad de la red no es suficiente ya que las redes profundas tienden a presentar el problema de la desaparición del gradiente (*vanishing gradient problem*), es decir, dado que el gradiente se retropropaga a las capas anteriores, la multiplicación repetida puede hacer que el gradiente sea infinitamente pequeño. Por tal motivo, entre más profunda es la red, su rendimiento comienza a saturarse o degradarse rápidamente. ResNet es capaz de entrenar cientos de capas y aún así obtener un buen desempeño. Esto es gracias a que intruduce una conección residual también llamada *"identity shortcut connection"*, la cual salta una o más capas y pasa a la siguiente directamente creando bloques residuales y mejorando así el proceso de aprendizaje (figura 3.3).

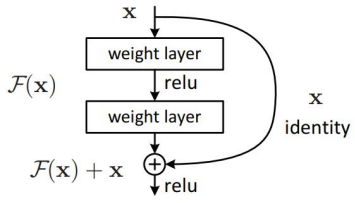


Figura 3.3: Bloque Residual en ResNet (Imagen tomada de [100])

La arquitectura ResNet cuenta con muchas variantes, esto significa que en todas ellas se aplica el mismo concepto pero con un número distinto de capas. Por ejemplo, se tiene ResNet-18, ResNet-34, ResNet-50, ResNet-101, ResNet-

110, ResNet-152, ResNet-164, ResNet-1202, etc. El nombre ResNet seguido de un número de dos o más dígitos significa una arquitectura ResNet con dicho número de capas de red neuronal. En la figura 3.4 se muestra las arquitecturas ResNet publicadas en el artículo original [100].

layer name	output size	18-layer	34-layer	50-layer	101-layer	152-layer
conv1	112×112	7×7, 64, stride 2				
conv2_x	56×56	3×3 max pool, stride 2				
		$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 64 \\ 3\times3, 64 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 64 \\ 3\times3, 64 \\ 1\times1, 256 \end{bmatrix}\times3$
conv3_x	28×28	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 128 \\ 3\times3, 128 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times4$	$\begin{bmatrix} 1\times1, 128 \\ 3\times3, 128 \\ 1\times1, 512 \end{bmatrix}\times8$
conv4_x	14×14	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 256 \\ 3\times3, 256 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times6$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times23$	$\begin{bmatrix} 1\times1, 256 \\ 3\times3, 256 \\ 1\times1, 1024 \end{bmatrix}\times36$
conv5_x	7×7	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times2$	$\begin{bmatrix} 3\times3, 512 \\ 3\times3, 512 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$	$\begin{bmatrix} 1\times1, 512 \\ 3\times3, 512 \\ 1\times1, 2048 \end{bmatrix}\times3$
	1×1	average pool, 1000-d fc, softmax				
FLOPs		1.8×10^9	3.6×10^9	3.8×10^9	7.6×10^9	11.3×10^9

Figura 3.4: Variantes de ResNet publicadas en [100]

En resumen, una CNN comienza con una imagen de entrada a la cual le aplica varios filtros diferentes para crear un mapa de características. Posteriormente aplica una función ReLU para aumentar la no linealidad y aplica una capa de agrupación a cada mapa de características. Por último, aplana las imágenes agrupadas en un vector largo e introduce el vector en una red neuronal artificial completamente conectada para procesar las características a través de la red. La capa final completamente conectada proporciona la "votación" de las clases que se buscan. Se entrena a través de la propagación hacia adelante y la propagación hacia atrás durante muchas, muchas épocas. Esto se repite hasta que se tiene una red neuronal bien definida con pesos entrenados y detectores de características.

La detección de imágenes implica una CNN que detecte una cantidad limitada de objetos (no pudiendo detectar objetos que antes no hubiera visto) y poder determinar en qué posición (dentro de la imagen) se encuentran. Entonces para entrenar estos modelos, se debe indicar la clase del objeto (por ejemplo perro ó gato) y además la posición dentro de la imagen, X, Y, el ancho y alto del objeto. Por ejemplo, suponiendo que se tiene una red CNN preentrenada para detectar perros y gatos con una muy buena taza de aciertos (como Alexnet, Resnet o VGG) y a esta red se le pasa una imagen nueva, devolverá "perro" o "gato" o incluso se le puede agregar una tercera salida "otros" por si se le pasa la foto de algo que no sepa reconocer. Si a esta red preentrenada, se le pasa una imagen con 2 perros será incapaz de detectarlos, puede que no detecte ni siquiera a uno. Si se le pasa una imagen con perros y gatos, tampoco los podrá identificar y mucho menos localizar.

En estos casos se puede utilizar alguno de los algoritmos existentes para detección de objetos, los cuales pueden ser divididos en dos grupos: algoritmos basados en clasificación y algoritmos basados en regresión. Los algoritmos basados en clasificación son implementados en dos partes, primero seleccionan las regiones de interés en una imagen y luego clasifican esas regiones usando redes neuronales convolucionales. Esta solución puede ser lenta porque se deben de hacer predicciones para cada región. Uno de los algoritmos más conocidos de este tipo es el *Region-based Convolutional Neural Network* (RCNN) y sus variantes. Los algoritmos basados en regresión en vez de seleccionar partes interesantes de la imagen, predicen clases y *bounding boxes* para toda la imagen en una sola corrida del algoritmo. Los dos algoritmos más conocidos de este tipo son YOLO (*You Only Look Once*) y su familia de algoritmos y SSD (*Single Shot Multibox Detector*). Estos son comúnmente usados para detección de objetos en tiempo real dado que en general sacrifican un poco de precisión por mejoras visibles en velocidad.

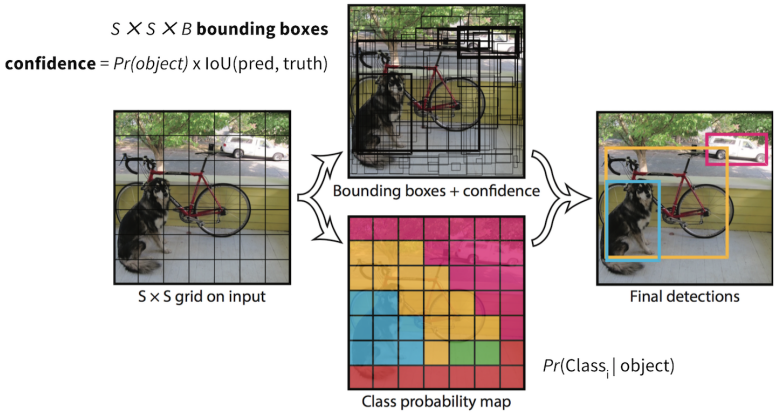


Figura 3.5: Funcionamiento de YOLO (Imagen tomada de [101])

YOLO fue descrito en 2015 por Joseph Redmon et al. [102]. Esta red hace una única pasada a la red convolucional y detecta todos los objetos para los que ha sido entrenada para clasificar. Al ser un sólo cálculo y sin necesidad de iterar, logra velocidades nunca antes alcanzadas incluso con computadoras no tan potentes. Esto permite detección sobre video en tiempo real de cientos de objetos en simultáneo y hasta su ejecución en dispositivos móviles. YOLO divide la imagen en una cuadrícula de S x S y en cada una de las celdas predice N posibles *"bounding boxes"* y calcula el nivel de certidumbre (o probabilidad) de cada una de ellas, es decir, se calculan S x S x N diferentes cajas, la gran mayoría de ellas con un nivel de certidumbre muy bajo. Después de obtener estas predicciones se procede a eliminar las cajas que estén por debajo de un límite.

A las cajas restantes se les aplica un paso de *"non-max suppression"* que sirve para eliminar posibles objetos que fueron detectados por duplicado y así dejar únicamente el más confiable de ellos (figura 3.5). YOLO utiliza una red CNN llamada Darknet, aunque también puede ser entrenada con cualquier otra red convolucional (como AlexeyAB/darknet y Darkflow que son otras implementaciones). Se le critica que, si bien es rápida, suele tener menor porcentaje de aciertos frente a las R-CNN. Pero con el paso del tiempo fueron evolucionando las versiones YoloV2, V3, V4 y recientemente V5 que están enfocadas a mejorar esa precisión de las *bounding boxes*, a la vez que mantienen su rapidez.

3.4. Redes neuronales con ramas de atención

Las redes convolucionales profundas (CNN) tienen un gran desempeño en varias tareas de reconocimiento de imágenes, sin embargo, es difícil interpretar el proceso de toma de decisiones de las mismas. La explicación visual permite que los humanos comprendan dichas decisiones, más no representa una contribución al desempeño de la CNN.

Las redes con ramas de atención (ABN, del inglés *attention branch network*) [103] utilizan los mapas de atención para la explicación visual, los cuales representan valores de respuesta altos en donde se ubica la atención en reconocimiento de imágenes. Esta región de atención mejora significativamente el desempeño de la CNN al introducir un mecanismo de atención que se enfoca en una región especifica de la imagen. Entonces, las ABN pueden ser aplicadas a muchas tareas de reconocimiento de imágenes gracias a la inclusión de una rama para el mecanismo de atención que además es entrenable tanto para explicación visual como para el reconocimiento de imágenes.

Las ABN constan de tres componentes o módulos principales: extractor de características (*feature extractor*), rama de atención (*attention branch*) y rama de percepción (*perception branch*). El extractor de características contiene múltiples capas convolucionales que extraen el mapa de características dada la imagen de entrada, la rama de atención genera la localización de la atención en un mapa de atención usando un mecanismo de atención. La rama de percepción obtiene las probabilidades de cada clase al recibir los mapas de características y atención de los módulos anteriores (figura 3.6).

La función de pérdida de entrenamiento $L(X_i)$ es una simple suma de las pérdidas de ambas ramas expresado en la Ecuación 3.1.

$$L(X_i) = L_{att}(X_i) + L_{per}(X_i) \tag{3.1}$$

Donde $L_{att}(X_i)$ representa la pérdida de la rama de atención dada una entrada X_i y $L_{per}(X_i)$ es la pérdida de la rama de percepción.

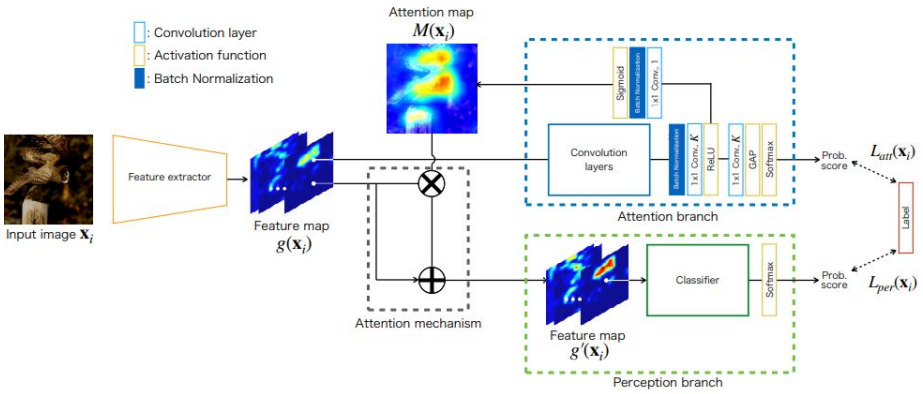


Figura 3.6: Estructura de las ABN (Imagen tomada de [103])

Las pérdidas de entrenamiento para ambas ramas son calculadas por la combinación de la función softmax y la entropía cruzada en tareas de clasificación de imágenes (Ecuación 3.2).

$$J = -\sum_{i=1}^{n} y_i^* \log p(y_i) \tag{3.2}$$

A diferencia de los mecanismos de atención que generalmente están integrados en la red principal de manera no supervisada, en las ABN son ramas dedicadas que generan un mapa de atención a partir de las etiquetas de salida. Una rama de atención está diseñada para enfocarse en regiones específicas de una característica al generar un mapa de atención. Su mapa de atención se utiliza para mejorar el rendimiento de la red. Tal enfoque se ha utilizado previamente en algunos trabajos [104] [105] en los que se utilizaron modalidades lingüísticas y visuales para generar instrucciones de búsqueda y predecir un objetivo para buscar.

Capítulo 4

Selector de Estrategias

Se eligió el robot HSR como plataforma de hardware dadas sus características (detalles en el Apéndice A) y la disponibilidad de dicho robot en nuestro laboratorio. Sin embargo, se requiere una gran cantidad de datos para entrenar redes neuronales profundas, además de que dichos datos deben contar con ciertas características para producir buenos resultados. Por estas razones, se decidió utilizar el simulador SIGVerse (detalles en el Apéndice B), el cual proporciona un ambiente virtual simulando adecuadamente el robot HSR en un ambiente virtual doméstico.

Una de las ventajas de usar un simulador es hacer el proceso de obtención del conjunto de datos (*dataset*) más ágil y confiable. Además, el proceso de etiquetado del conjunto de datos se puede automatizar y estandarizar. Cuando se utiliza el robot real, esta tarea suele ser realizada manualmente por un humano experto lo cual es subjetivo. Al usar el simulador, éste se puede programar para tener un criterio de etiquetado unificado, no subjetivo ni dependiente de un experto. Otro punto importante a considerar es el riesgo de que el robot físico choque con objetos, muebles o personas, ya que además de lastimar a alguien también se puede dañar al robot mismo o el ambiente. Esto se evita totalmente con el robot y ambiente simulados.

Al utilizar SIGVerse podemos asegurar que el proceso de transferir nuestro desarrollo a un HSR real será prácticamente transparente, dado que SIGVerse fue diseñado para los HSR y emula de manera destacada todos sus componentes (como navegación, colisiones, movimientos de partes del robot, cámaras y sensores) además de emular algunos aspectos de los ambientes reales como ruido, fricción, etc. Adicionalmente, la programación del robot virtual es la misma que en el real, ya que ambos usan ROS (y sus módulos para desarrollar sus sistemas), por lo que un sistema que ya funciona en un HSR real puede trabajar con nuestros programas desarrollados en el ambiente virtual y viceversa (este proceso ya ha sido realizado con éxito para el evento WRS antes mencionado).

4.1. Conjunto de datos

Se desarrolló una simulación (usando como base SIGVerse) para generar un contexto de colocación de objetos para el HSR, con esto se puede obtener un conjunto de datos de gran escala al automatizar el proceso con objetos, posiciones, destinos y ubicaciones aleatorias, además de contar con etiquetado automático usando el mismo criterio en cada caso. Se utilizó SIGVerse como plataforma base ya que trabaja con la misma configuración y software que el HSR real, esto lo hace ideal para obtener datos significativamente buenos ya que asegura un comportamiento similar en el hardware del HSR (como movimientos del robot y cámaras).

Para comprender mejor todo el contexto de este proyecto se definen algunos conceptos: el objeto que será colocado en el destino es el *objetivo* (figura 4.1a), el mueble, mesa o superficie donde el objetivo será colocado es el *destino* (figura 4.1b), los otros objetos que ya se encuentran en el destino son los *obstáculos* (figura 4.1b). Los objetivos y obstáculos son objetos de la vida cotidiana. El objetivo debe ser de tamaño adecuado para poder ser sostenido por el HSR.

(a) Objetivo (b) Destino con obstáculos

Figura 4.1: Ejemplos de objetivo, destino y obstáculos

Una *estrategia de colocación* es una serie de movimientos simples del robot para colocar un objetivo en el destino. Por ejemplo: *Moverse un paso a la izquierda, levantar el brazo del robot, mover el brazo hacia el frente, bajar el brazo y abrir la mano para soltar el objeto.* Se tiene de antemano un conjunto reducido de estrategias de colocación que han sido probadas anteriormente. Se considera una *colisión* cuando el robot o el objetivo chocan con algún obstáculo al momento de la colocación, sin embargo, una colisión puede ser sólo un pequeño roce o un choque violento, en este caso consideramos una colisión como "dañina" si la velocidad de colisión o choque es mayor a un cierto umbral (0.1 m/s) el cual se puede modificar si así se desea (por ejemplo, más alto 0.99) para

que el movimiento sea ejecutado solamente si la probabilidad de éxito es muy grande. El *éxito* se define como colocar el objetivo en el destino usando una de las estrategias sin tener colisiones dañinas, es decir, sin derribar, tirar o dañar otros objetos, muebles, el robot o algo en el ambiente.

Se utilizaron 5 superficies destino, $F = \{f_i | i = 1, ..., 5\}$, es decir, 5 muebles distintos donde los objetos podían ser colocados. También se utilizaron 27 objetos cotidianos como candidatos a obstáculos $O = \{o_i | i = 1, ..., 27\}$ los cuales también son candidatos a objetivo $T = \{t_i | i = 1, ..., 27\}$ (figura 4.2).

Figura 4.2: Muebles con superficies destino y objetos candidatos a objetivo y obstáculos

La simulación inicia con una escena virtual compuesta de un ambiente doméstico con el HSR en una posición dada (figura 4.3). Para cada escena se coloca un mueble (mesa, escritorio, repisa, etc.) con superficie destino f_i enfrente del HSR a una distancia que el robot puede alcanzar. Sobre esta superficie un número aleatorio (entre 0 y 27) de objetos aleatorios (del conjunto O) se seleccionan y se colocan en una posición aleatoria (para que aparezcan dispersos sobre f_i). El objetivo a colocar se encuentra en la mano del robot.

Figura 4.3: Escena virtual inicial

Cuando la escena termina de configurarse y está lista, se envía un mensaje al robot para posicionar su cabeza, para enfocar el área objetivo sobre f_i, entonces obtiene las imágenes de profundidad (*Depth*) y RGB de sus cámaras (tamaño 640 x 480) (figura 4.4).

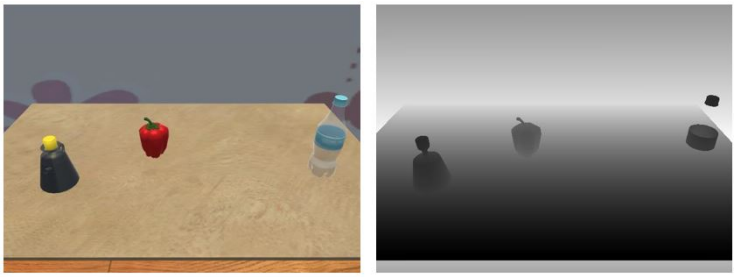


Figura 4.4: Imágenes RGB y de profundidad obtenidas por el robot

Una vez capturadas las imágenes, el robot procede a intentar colocar el objetivo en el área destino utilizando una estrategia simple de colocación (por ejemplo moverse unos pasos a la derecha y extender su brazo hacia el destino); no se utilizó ningún algoritmo específico ni evasión de obstáculos, solamente colocar el objetivo usando una de las estrategias disponibles al azar. Cada intento es calificado automáticamente con 0 puntos si fue un fracaso o 100 puntos si fue un éxito. El éxito significa que el robot es capaz de colocar el objetivo en el área destino sin colisiones graves. Una colisión se da cuando el robot o el objetivo chocan con algún obstáculo de O. La calificación se convierte en etiquetas para la escena dada y se envía toda la información al robot. Posteriormente se genera una nueva escena y el proceso comienza de nuevo usando una configuración de muebles, obstáculos y objetivo distintos. Por lo tanto, un conjunto de datos de tamaño N estará compuesto por N imágenes de profundidad, N imágenes RGB y N etiquetas.

Utilizar diferentes superficies donde colocar objetos vuelve complicado usar sólo imágenes RGB ya que los colores, texturas y materiales de los mismos hacen difícil hacer generalizaciones del ambiente. En estos casos el uso de imágenes de profundidad puede ser más adecuado. Sin embargo, cuando se tienen objetos transparentes o complejos que no pueden ser correctamente detectados por las cámaras de profundidad, las imágenes RGB son más confiables (figura 4.5). El uso de diferentes tipos de datos (imágenes RGB y de profundidad) permite tener información más detallada sobre el ambiente que usando solo un tipo, esto es muy útil al entrenar una red, en nuestro caso los datos RGB serán muy útiles cuando no se cuente con suficientes datos de profundidad como en los objetos transparentes o huecos o cuando simplemente se pierda algo de información de profundidad.

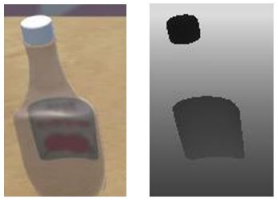


Figura 4.5: Objetos Transparentes (RGB y profundidad)

Para este proyecto es muy importante tener diversidad en el conjunto de datos para no trabajar datos similares, por ejemplo, en un mismo mueble con los mismos objetos y posiciones, este es el objetivo de tener datos con diferentes superficies, obstáculos y objetivos. Al tener esa variedad en los datos se puede generalizar el modelo y asegurar que no sólo funciona con una escena o configuración predefinida.

4.2. Arquitectura del Selector

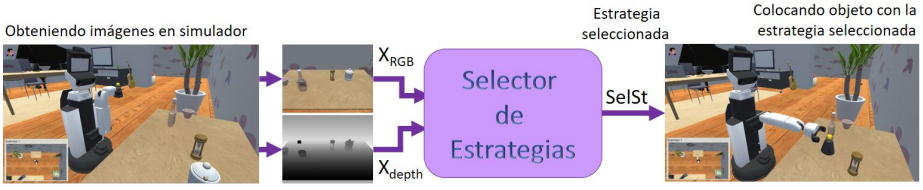


Figura 4.6: Esquema general del Selector de Estrategias

El esquema general del Selector de Estrategias se muestra en la figura 4.6. Primero, el robot llega con el objetivo en su mano y se posiciona frente a la superficie destino. Una vez en la posición adecuada mueve la cabeza para que las cámaras estén dirigidas hacia la superficie destino y procede a capturar las imágenes de profundidad y RGB. El conjunto de entradas del Selector $x(i)$ se define como:

$$x(i) = \{x_{RGB}(i), x_{depth}(i)\} \tag{4.1}$$

Donde x_{RGB} y x_{depth} denotan las imágenes RGB y de profundidad.

El robot envía las entradas hacia el Selector de Estrategias, el cual después de procesarlas, hace una predicción y genera la estrategia que tiene mayores probabilidades de éxito ($SelSt$). Esta estrategia es recibida por el robot, el cual procede a ejecutar los movimientos correspondientes para colocar el objetivo en el destino, idealmente con éxito.

Internamente, el Selector se compone de 3 partes esenciales (figura 4.7): el módulo de preprocesamiento, la red neuronal profunda y el discriminador. El módulo de preprocesamiento toma las imágenes obtenidas por el robot (X_{RGB} y X_{depth}) y realiza el proceso descrito en la sección 4.3 para que éstas puedan ser usadas como entradas en la red neuronal profunda.

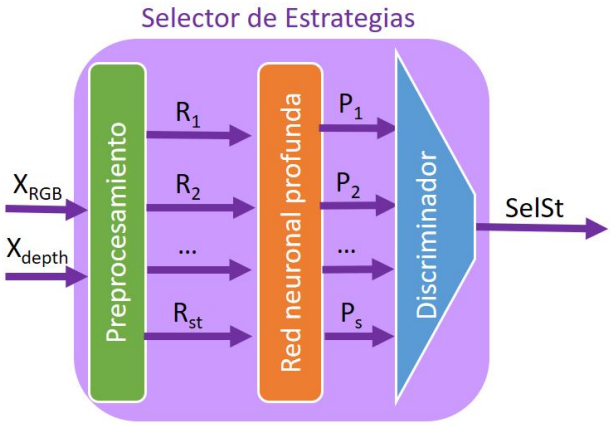


Figura 4.7: Arquitectura del Selector

La red neuronal profunda recibe como entradas las imágenes procesadas ($R_n(i)$). Esta red neuronal fue previamente entrenada con el conjunto de datos descrito anteriormente y produce la predicción de éxito o fracaso para cada una de las estrategias existentes ($P_n(i)$). Finalmente, el discriminador simplemente elige la estrategia que presente la mayor posibilidad de éxito para enviarlo al robot como salida del Selector ($SelSt$).

4.3. Preprocesamiento

Los datos obtenidos en la simulación deben ser preprocesados para poder ser utilizados por la red neuronal profunda. Las entradas para el módulo de preprocesamiento son las imágenes $x_{RGB}(i)$ y $x_{depth}(i)$.

El tamaño de las imágenes originales es 640 x 480, sin embargo, los movimientos de colocación de cada estrategia son fijos, eso significa que por cada estrategia hay una región de interés (ROI, del inglés *region of interest*) fija dentro de la imagen. Es decir, el robot no es capaz de colocar el objetivo en cualquier parte del área destino dados los movimientos fijos para cada estrategia. Entonces, consideremos el área destino como la sección de la imagen en donde el robot puede colocar el objetivo, es decir, el área donde el robot "alcanza" con su brazo dada su posición actual y los movimientos de colocación de la estrategia elegida.

Dicha área destino es más pequeña que la original, entonces se divide la imagen original en secciones o parches (*patches*) y se selecciona el más adecuado para cada estrategia, este parche corresponde a la región de interés (ROI) de la estrategia (figura 4.8).

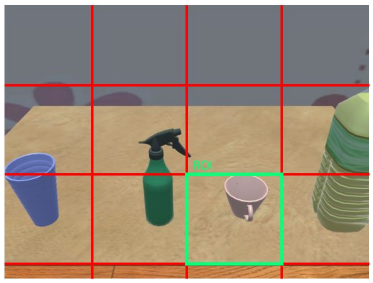


Figura 4.8: Imagen dividida en parches y ROI

Dado un numero S de estrategias, las ROIs se pueden obtener dividiendo cada una de las entradas $x_{RGB}(i)$ y $x_{depth}(i)$ en S secciones correspondientes a cada estrategia. El resultado de esta división es un conjunto de ROIs de RGB y profundidad $R_n(i)$ para cada estrategia.

$$R_n(i) = \{R_n^{RGB}(i), R_n^{depth}(i)\}, 1 \leq n \leq S \tag{4.2}$$

Para encontrar el tamaño de parche más apropiado se probaron varias configuraciones para éstos usando distintas posiciones dentro de la imagen original y tamaños de los parches (128, 160, 192), esto con el fin de utilizar solamente información relevante y de menor tamaño. Inicialmente se trabajó con los parches originales obtenidos de las imágenes (RGB y profundidad) y se tuvieron buenos resultados, sin embargo, se decidió probar con ResNet (preentrenado) para el extractor de características (detalles en la sección 4.4), en este caso se optó por redimensionar las imágenes para poder utilizar algunas herramientas donde ya se tienen implementadas dichas redes, en este caso se redimensionaron a 224 x 224.

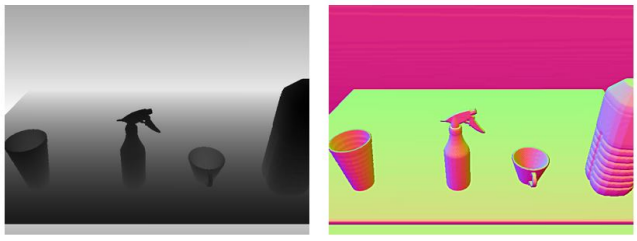


Figura 4.9: Imagen de pronfundidad original (izquierda) y con *surface normals* (derecha)

Por otro lado, inspirados en la literatura de visión computacional, las ROI de profundidad ($R_n^{depth}(i)$) se transformaron a una imagen a color basándose en las normales de superficie (*surface normals*) [106]. Usando este método las imágenes de profundidad ahora se pueden procesar con una arquitectura de red convencional para imágenes RGB (figura 4.9). Las imágenes transformadas con éste método, son imágenes RGB donde los colores representan la profundidad.

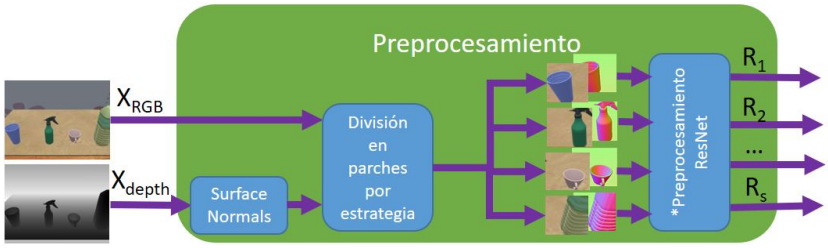


Figura 4.10: Módulo de preprocesamiento

En resumen, las imágenes originales se procesan de la siguiente manera (figura 4.10): Las imágenes de profundidad se convierten a imágenes SN (del inglés *surface normals*) usando el método descrito anteriormente. Una vez convertidas, tanto las RGB como las SN son divididas en parches o ROIs por cada estrategia, opcionalmente se les aplica el preprocesamiento para usar redes preentrenadas tipo ResNet y finalmente las salidas son los conjuntos de ROIs de RGB y SN denotados como:

$$R_n(i) = \{R_n^{RGB}(i), R_n^{SN}(i)\}, 1 \leq n \leq S \tag{4.3}$$

4.4. Red neuronal profunda

Inspirados en [103], [106], [107] y [108], la red neuronal profunda realiza una predicción del éxito o fracaso al colocar objetos usando cada una de las N estrategias de un conjunto de movimientos fijos. Esta red (figura 4.11) se divide en N subredes en cada una de las cuales se tienen dos flujos principales, uno para las entradas RGB y otro para las SN. En cada flujo hay un extractor de características (EC) tipo CNN y una rama de atención (RA) tipo ABN para finalmente unir ambas en una rama de percepción (RP).

Entradas y salidas de la red

Como se mencionó anteriormente, las entradas de la red son las imágenes de RGB y SN de las ROIs $R_n(i)$. Esto significa que dada una imagen del destino i, el conjunto de entradas $x(i)$ se define como:

$$x(i) = \{R_n^{RGB}(i), R_n^{SN}(i)\} \tag{4.4}$$

Donde $R_n^{RGB}(i)$ y $R_n^{SN}(i)$ representan las ROIs RGB y SN respectivamente.

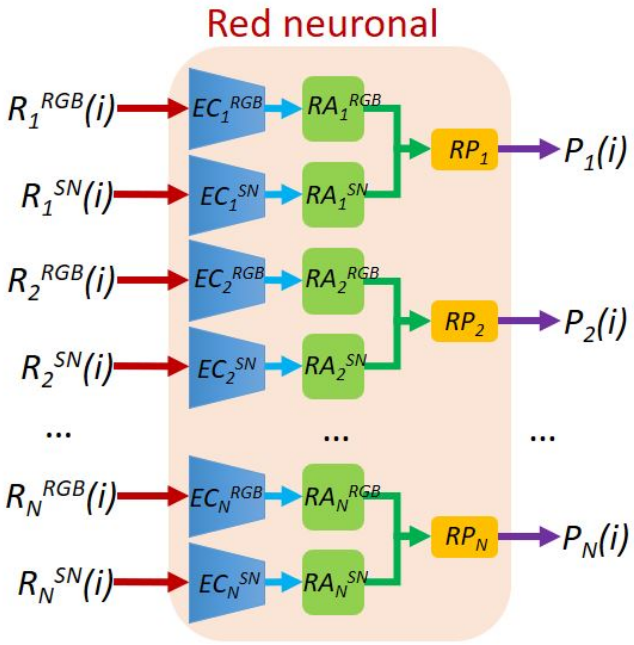


Figura 4.11: Diagrama general del modelo de la red

La salida de la red es un conjunto de predicciones $P_n(i)$ (una por cada estrategia), es decir, la probabilidad de colocación sin colisiones dañinas (por ejemplo $P_n(x|y) = 0.2$). Donde x representa el éxito al colocar usando la estrategia n y y la muestra dada (imágenes de la escena).

$$P_n(i) = \{P_n(i)\}, 1 \leq n \leq S \tag{4.5}$$

Modelo de la red

Ambos extractores de características (RGB y SN) están conformados por varias capas convolucionales residuales para extraer los mapas de características (*feature maps* $f_m(i)$) a partir de las imágenes (RGB y SN) de entrada.

$$f_m(i) = \{f_m^{RGB}(i), f_m^{SN}(i)\} \tag{4.6}$$

Para los extractores de características se usaron las primeras 25 capas de un modelo ResNet-50 preentrenado (figura 4.12) y se utilizaron técnicas de aprendizaje por transferencia para obtener mapas de características robustos.

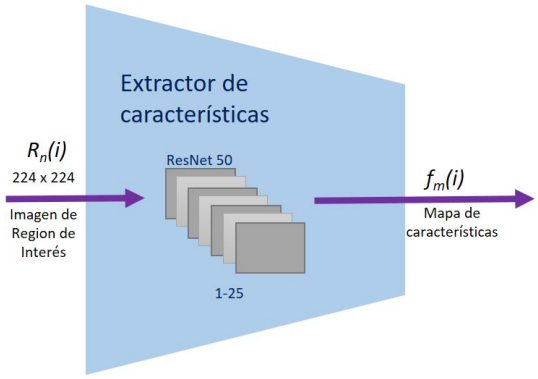


Figura 4.12: Extractor de características (para RGB y SN)

Las ramas de atención inspiradas en [103] tienen como entradas los mapas de características $f_m(i)$ y como salidas la predicción de la rama en cuestión $P_{att}(i)$ así como la ubicación de la atención en un mapa de atención de características ponderadas $w_m(i)$. Cada rama de atención tiene un bloque de capas convolucionales, las cuales son bloques residuales dispuestos de manera similar a la arquitectura ABN [103] con un mecanismo de atención (figura 4.13).

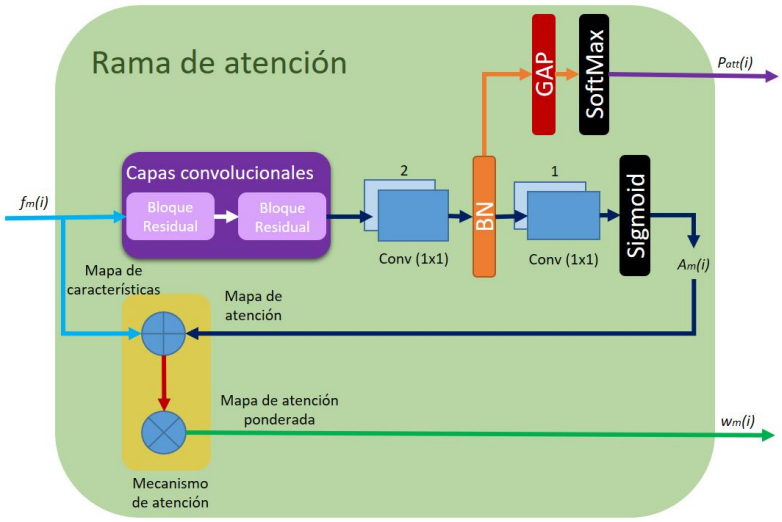


Figura 4.13: Rama de atención (para RGB y SN)

Los bloques residuales están conformados por capas convolucionales residuales como se muestra en la figura 4.14.

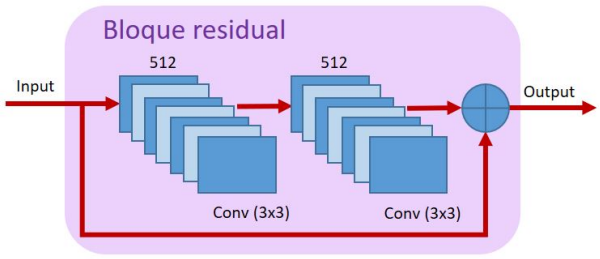


Figura 4.14: Bloques residuales

Después de las capas convolucionales se tiene una capa convolucional con una normalización por lotes (*batch normalization*) y a partir de aquí se tienen dos flujos, uno para obtener la predicción para la rama $P_{att}(i)$ (ecuación 4.7) y el otro para obtener el mapa de atención $a_m(i)$ (ecuación 4.8). En el primer flujo se tienen un promedio global grupal (*global average pooling*) para reducir dimensiones y una función de activación SoftMax para generar las predicciones $P_{att}(i)$. En el segundo, se tiene una capa convolucional extra con una función de activación simgoide para generar el mapa de atención $a_m(i)$.

$$P_{att}(i) = \{P_{att}^{RGB}(i), P_{att}^{SN}(i)\} \tag{4.7}$$

$$a_m(i) = \{a_m^{RGB}(i), a_m^{SN}(i)\} \tag{4.8}$$

Con los mapas de características, se obtiene la localización de la atención en los mapas de atención $a_m(i)$. Finalmente, el mecanismo de atención utiliza los mapas $f_m(i)$ y $a_m(i)$ y la ecuación 4.9 para resaltar el mapa de características en el pico del mapa de atención mientras se evita que la región de menor valor del mapa de atención se degrade a cero [103]. Con esto se genera un mapa de atención de características ponderadas $w_m(i)$ (ecuación 4.10).

$$w_m(i) = (1 + a_m(i)) \cdot f_m(i) \tag{4.9}$$

$$w_m(i) = \{w_m^{RGB}(i), w_m^{SN}(i)\} \tag{4.10}$$

Los mapas de atención ponderada $w_m(i)$ y las predicciones para cada rama $P_{att}(i)$ son las salidas de las ramas de atención. Ambas ramas (RGB y SN) funcionan de la misma manera.

Las entradas para la rama de percepción son los mapas de atención ponderada $w_m(i)$. La salida es la probabilidad $P_N(i)$ de éxito para cada estrategia (figura 4.15).

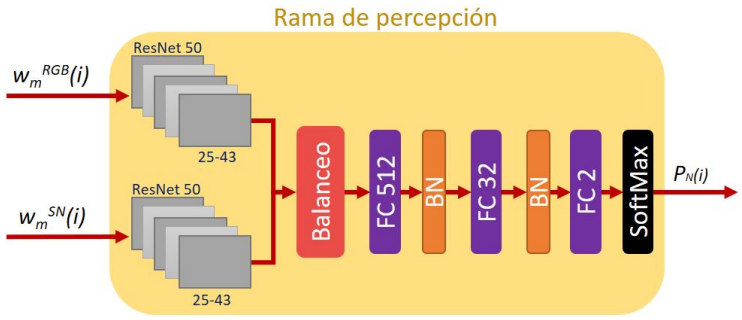


Figura 4.15: Rama de percepción

Los mapas de atención ponderada ($w_m^{RGB}(i), w_m^{SN}(i)$) son procesados en paralelo por dos segmentos ResNet-50 (capas 26-43) idénticos y entrenables. Se descubrió que en algunos casos la rama SN generaba mejores resultados que la rama RGB y en otros casos era a la inversa, por lo que se decidió añadir una capa de balanceo (con un mecanismo de atención temporal [107]) para obtener mejores resultados al ponderar selectivamente las ramas dependiendo de qué tan buenos fueron los resultados de cada rama. De otro modo los malos resultados de una rama podrían hacer que la red completa tuviera un rendimiento inferior incluso cuando los resultados de la otra rama fueran buenos. Entonces, los dos flujos gemelos son unidos con el mecanismo de balanceo dentro de la rama de Percepción [108]. Después del mecanismo de balanceo se entra a una serie capas totalmente conectadas (*fully connected layers* de 512, 32 y 2) cada una seguida de normalizaciones por lotes (*batch normalization*) y finalmente a una función de activación SoftMax para obtener la predicción final $P_N(i)$.

El mecanismo de atención temporal utilizado [107] pondera selectivamente las entradas $w_m^{RGB}(i)$ y $w_m^{SN}(i)$ en diferentes momentos. Este mecanismo usa pesos de atención para realizar un promedio ponderado (ecuación 4.11), estos pesos permiten que la red enfatice las características que son más importantes para predecir la salida.

$$A = \sum \alpha_\pi \cdot F_\pi \tag{4.11}$$

Donde:

π son todos los diferentes tipos de entrada, en este caso se tienen tipo RGB y SN, entonces $\pi = \{RGB, SN\}$

F son los mapas de atención de características ponderados ($w_m^{RGB}(i)$ y $w_m^{SN}(i)$)

α son los pesos de atención y son calculados a partir de los mapas de atención ponderada como se muestra en la ecuación 4.12.

$$\alpha_k = \frac{\exp(e_k)}{\sum_{\pi=1}^{L} \exp(e_\pi)} \tag{4.12}$$

Donde e_k es un escalar calculado con la ecuación 4.13 y $L = 2$ ya que solo se tienen entradas de tipo RGB y SN.

$$e_k = V_k^T \cdot \tanh(W_k \cdot F_k + b_k) \tag{4.13}$$

F_k son los mapas de atención de características ponderados y la matriz de pesos W_k y V_k así como el vector de sesgo (bias) b_k son parámetros entrenables. De la ecuación 4.12 se puede obtener α_{RGB} y α_{SN} :

$$\alpha_{RGB} = \frac{\exp(e_{RGB})}{\exp(e_{SN}) + \exp(e_{RGB})} \alpha_{SN} = \frac{\exp(e_{SN})}{\exp(e_{SN}) + \exp(e_{RGB})} \tag{4.14}$$

Y de la ecuación 4.13 se puede obtener e_{RGB} y e_{SN} :

$$e_{RGB} = V_{RGB}^T \tanh(W_{RGB} \cdot F_{RGB} + b_{RGB}) e_{SN} = V_{SN}^T \tanh(W_{SN} \cdot F_{SN} + b_{SN}) \tag{4.15}$$

Ahora se usan las ecuaciones 4.14 y 4.15 para realizar un promedio ponderado de la ecuación 4.11.

$$A_{RGBD} = \alpha_{RGB} \cdot F_{RGB} + \alpha_{SN} \cdot F_{SN} \tag{4.16}$$

Finalmente quedan balanceados los mapas de atención de características (Ecuación 4.16).

Función de pérdida

Para predecir la probabilidad $p(y)$ de éxito al colocar el objeto con la estrategia dada, la función de pérdida J_{total} de la red está dada por la ecuación 4.17.

$$J_{total} = \lambda_{RGB} J_{RGB} + \lambda_{SN} J_{SN} + \lambda_{per} J_{per} \tag{4.17}$$

Donde λ_{RGB}, λ_{SN} y λ_{per} son los diferentes pesos de pérdida. La notación genérica J para J_{RGB}, J_{SN} y J_{per} es la función de pérdida de entropía cruzada y se expresa como:

$$J = -\sum_n \sum_m y_{nm}^* \log p(y_{nm}) \tag{4.18}$$

Donde: y_{nm}^* representa la etiqueta dada a la m-ésima dimensión de la n-ésima muestra y y_{nm} representa su predicción.

4.5. Discriminador

Las entradas del discriminador son el conjunto de predicciones $P_N(i), 1 \leq N \leq S$ donde S es el número de posibles estrategias. Cada predicción representa la probabilidad de éxito de una muestra dada i para cada estrategia. Con estas predicciones, el discriminador elegirá una estrategia que optimice la posibilidad de éxito (ArgMax). La salida es la estrategia seleccionada, la cual es enviada al robot para ser ejecutada.

Capítulo 5

Integración de sistemas

Como se mencionó en la sección 2.4, en el laboratorio de BioRobótica se desarrolló un sistema de teleoperación con una interfaz de realidad mixta. Dicho sistema fué presentado como tesis de maestría [48] y posteriormente me fué asignado para su mantenimiento. A partir de entonces, el sistema fué modificado, añadiéndole algunas mejoras, así como también fué integrado con el Selector para proveerlo de la habilidad de colocación autónoma. Esta integración de sistemas, por un lado sirve como caso de prueba para Selector y por el otro como mejora al sistema de teleoperación. En este capítulo se describen las características originales de los sistemas involucrados, así como también las nuevas funcionalidades desarrolladas y su integración con el Selector.

5.1. Sistema de teleoperación

El sistema de teleoperación [48] cuenta con una interfaz de usuario inmersiva la cual presenta visualización del ambiente real y también de un ambiente virtual mapeado que representa fielmente al real. El objetivo de este sistema es que su interfaz de operación sea fácil de usar, se pueda usar en diferentes plataformas y que no consuma demasiados recursos. El sistema se compone de dos partes principales (figura 5.1), el dispositivo remoto y la interfaz de operación. El dispositivo remoto es un robot de servicio con una *laptop* en su espalda la cual lo controla y tiene una conexión a internet para comunicarse con el operador (figura 5.1 izquierda). Para controlar el robot remotamente se cuenta con una interfaz de operación inmersiva la cual se despliega en un HMD o casco de realidad virtual 3D (figura 5.1 derecha). Para la interfaz de operación existen dos posibles configuraciones de *hardware* (A y B). La configuración A (figura 5.1 esquina superior derecha) es un HMD conectado con cable a una PC y un *joystick* también cableado. La configuración B (figura 5.1 esquina inferior derecha) es un HMD que utiliza un teléfono inteligente para la visualización y un *joystick* conectado por Bluetooth. En ambas configuraciones se requiere una conexión a internet estable.

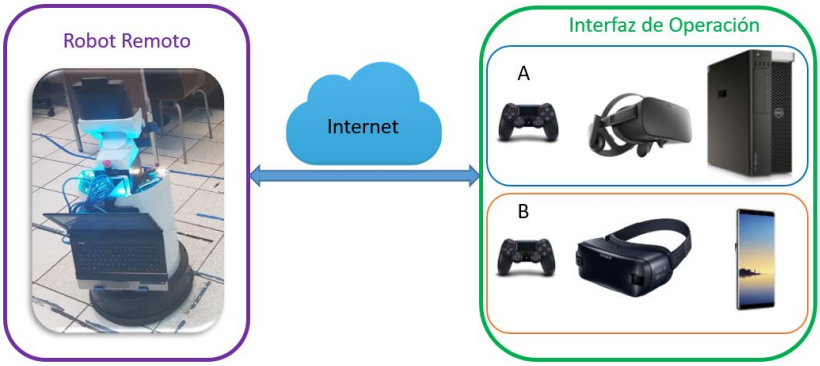


Figura 5.1: Partes principales del sistema de teleoperación. Robot remoto (izquierda) e interfaz de operación (derecha) con dos configuraciones de *Hardware* (A y B)

Para lidiar con los problemas de retraso en la transmisión de video se propone una mezcla de modos de visualización, un modo de video real y un modo de ambiente virtual. Los recursos necesarios para comunicarse con el robot se reducen gracias al ambiente virtual, además, cuando se necesita visualizar el ambiente real (como en los casos tradicionales de teleoperación) se puede cambiar el modo de visualización del virtual al real o viceversa. Esta dualidad permite tener lo mejor de ambos mundos. El ambiente real fue previamente mapeado en el ambiente virtual, esto significa que los ambientes real y virtual coinciden perfectamente. Por un lado, cuando se usa el modo virtual, solamente se requiere transmitir la ubicación del robot real lo cual reduce la cantidad de información enviada, mejora la eficiencia, agiliza el control y proporciona una mejor experiencia de teleoperación. Por otro lado, el modo de video real puede ser activado si hay necesidad de mostrar lo que el robot visualiza, es decir, este modo permite que el operador observe el ambiente real a través de las cámaras del robot y corrobore que el robot está comportándose adecuadamente.

El robot de servicio utilizado es nuestro robot HSR de Toyota "Takeshi". Se utilizan cascos de realidad virtual o HMD (del inglés *head mounted display*) para generar la sensación de inmersión. Los HMDs consisten en una pantalla dividida en dos para mostrar una imagen diferente para cada ojo del usuario dentro de un par de lentes. El campo de vista es ampliado por lo que la imagen parece estar a varios metros delante del usuario [109]. Estos cascos permiten ver el mundo virtual directamente en sus pantallas y generan la sensación inmersiva dentro del ambiente virtual. Para controlar e interactuar con el ambiente virtual se utiliza un *joystick*, el Dualshock 4 [110]. Este *joystick* es la cuarta generación de la línea de controles desarrollados por Sony Interactive Entertainment para la familia PlayStation. Cuenta con varios botones para comandos, así como también puede ser conectado vía Bluetooth o USB a dispositivos distintos al

PlayStation. Estas características permiten que el *joystick* pueda ser usado con PCs y dispositivos móviles con Android. El *hardware* utilizado es comercial y no muy especializado lo cual hace al sistema accesible a una mayor audiencia.

Para probar el desempeño del sistema en distintas plataformas, se eligieron dos HMDs, un Oculus Rift [111] conectado a una PC para la configuración A y un Samsung GearVR [112] con un teléfono inteligente Samsung Galaxy Note 8 para la configuración B. El Oculus Rift es un casco de realidad virtual desarrollado y manufacturado por Oculus VR. Sus características técnicas lo hacen ideal para la inmersión de este sistema. El Samsung GearVR también es un casco de realidad virtual creado por Samsung Electronics en colaboración con Oculus VR y es compatible con los dispositivos móviles de Samsung, las pantallas de estos dispositivos se utilizan para mostrar las imágenes y sus acelerómetros y giroscopios se usan para rotación y rastreo. La capacidad de este dispositivo de funcionar con dispositivos móviles lo hacen fácil de utilizar en contextos domésticos. La PC usada en la configuración A es una Dell Precision Tower 7810 con Sistema Operativo Windows 10 de 64 bit, Procesador Intel(R) Xeon(R) CPU E5-2650 v4 2.2GHz, Memoria RAM de 16 GB, GPU Quadro P400 con memoria de 8GB GDDR5 y 1792 núcleos CUDA. Este sistema utiliza ROS (*Robot Operating System*) para controlar al robot y sockets para conectar la interfaz con el robot. ROS [29] es un *middleware* de robótica, es decir un conjunto de bibliotecas de *software* y herramientas para construir aplicaciones robóticas. Se utilizó la versión Kinetic corriendo sobre el sistema operativo Ubuntu (Linux) 16.04 LTS.

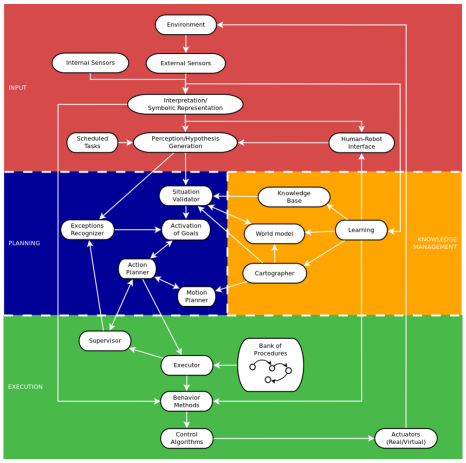


Figura 5.2: Diagrama general de ViRbot (Imagen tomada de [113])

Para la navegación autónoma, tareas complejas y los comportamientos del robot se utilizó una versión de ViRbot (*VIrtual and Real roBOt sysTem*) adaptada al HSR de Toyota. ViRbot [113] es una arquitectura robótica híbrida para

operar robots autónomos (figura 5.2). El sistema ViRbot interactúa con ROS para formar una plataforma de desarrollo de sistemas inteligentes para visión por computadora, procesamiento de señales digitales, planificación automática, control automático e interacción *humano-robot*. ViRbot se adaptó para trabajar con las bibliotecas propietarias de HSR y esta versión adaptada ha sido probada con Takeshi en concursos internacionales de robots como RoboCup, donde obtuvo el segundo lugar en 2018.

Unity 3D [114] es un motor de videojuegos que permite la creación de juegos para varios sistemas y dispositivos como PC, Xbox, Play Station, Nintendo, Android, iOS, Web, etc. Además, Unity 3D utiliza *C#* y *javascript* como lenguajes base para crear scripts que modelen comportamientos. En cuanto a los modelos 3D, es posible utilizar varios formatos como *.obj*, *.fbx* y *.mb*. Todas estas características hacen que Unity sea ideal para este sistema. La versión de Unity 3D utilizada fue 2018.2.21f1, que se ejecuta en el sistema operativo Windows 10 Professional.

Para generar un entorno 3D inmersivo, el video estéreo de las cámaras estereoscópicas del robot se transmite a la interfaz y se muestra en el HMD. En la interfaz, el operador puede visualizar un entorno real en 3D (modo de video) o cambiar a un entorno virtual en 3D (modo virtual). Además, el movimiento de la cabeza del operador en el HMD se mapeó al movimiento de la cabeza del robot, de modo que pudiera explorar libremente el entorno remoto moviendo la cabeza (figura 5.3).

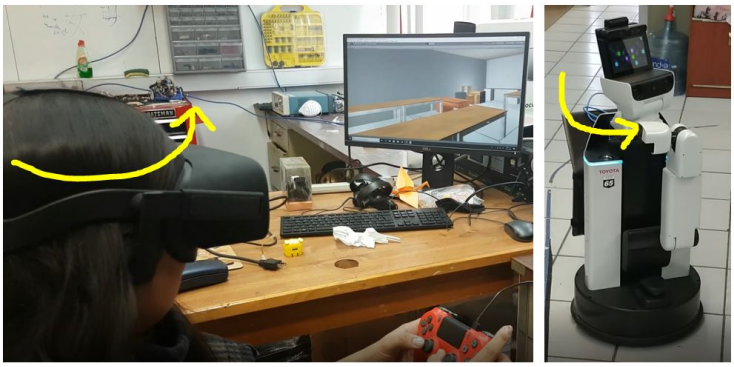


Figura 5.3: Interfaz de operación (izquierda) y robot teleoperado (derecha). El movimiento de la cabeza del operador controla el movimiento de la cabeza del robot

La información del movimiento de la cabeza se envía a ROS usando sockets para imitar este movimiento en el robot. Al recibir la información por sockets, ésta se publica como mensajes de ROS para ser ejecutados por el robot. Además, se considera cuidadosamente la precisión en la detección del movimiento del

HMD ya que, si no se maneja adecuadamente, incluso el más mínimo movimiento de la cabeza del operador podría desencadenar un movimiento de la cabeza del robot y esto podría causarle mareos. Para resolver esto, se implementó un umbral para la detección de movimiento, es decir, cuando el movimiento está por debajo del umbral, no se envía la información para evitar esta desagradable experiencia. Para controlar los movimientos restantes del robot, los botones y el stick del *joystick* se mapearon en Unity. Mediante este mapeo, Unity recopila información sobre la dirección y la velocidad de los movimientos del *joystick*. Esta información se envía a ROS para convertirla en comandos de movimiento para el robot.

Para la experiencia inmersiva en 3D, las imágenes de las cámaras estéreo del robot se muestran en los lentes del HMD correspondientes a cada ojo del usuario. Para esto se establece una conexión con el robot para recibir las imágenes en Unity. El robot puede capturar video con sus cámaras frontal y estéreo, pero en formato de imagen, esto significa que el robot no puede capturar video, sino que captura imágenes estáticas. Estas imágenes son bastante grandes ya que la visión estéreo requiere imágenes de ambas cámaras estéreo (izquierda y derecha). Por tanto, es necesario transmitir una gran cantidad de datos a través de la red. Para hacer frente a la calidad de los datos del video recibido, se aplican técnicas de mejora de la imagen. Estas técnicas se utilizan comúnmente para mejorar la apariencia visual de las imágenes, en términos de nitidez, distorsión, contraste, etc.

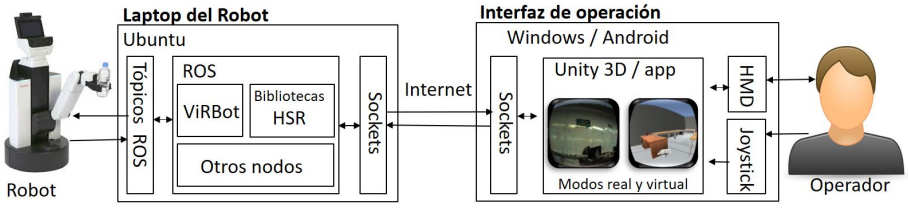


Figura 5.4: Diagrama general de funcionamiento del sistema de teleoperación.

La *laptop* conectada al robot (figura 5.4 izquierda) contiene los nodos ROS y está conectada al robot a través de un cable Ethernet y a Internet de forma inalámbrica para conectarse con la interfaz. El dispositivo que tiene la interfaz (figura 5.4 derecha) se puede conectar a Internet a través de Ethernet, red Wi-Fi o red móvil (en el caso de Samsung Gear VR). La comunicación debe establecerse desde el dispositivo que ejecuta la interfaz Unity a la computadora que ejecuta ROS.

La computadora ROS recibe los datos y los convierte en información para ser publicada en ROS para que la ejecute el robot. Cuando recibe información de los nodos, la envía a través de sockets al dispositivo Unity 3D conectado.

El entorno virtual fue diseñado y desarrollado para realizar la teleoperación del HSR. Entonces, el HSR real se replicó en el entorno virtual mediante un modelo HSR de Unity del simulador SIGVerse. Para abstraer el espacio real en el espacio virtual, se requirieron escalas correspondientes adecuadas. Por lo tanto, el modelo de robot se utilizó como referencia de escala para hacer coincidir los entornos reales y virtuales. Esta abstracción se logró utilizando el robot para mapear el espacio por donde iba a navegar. Utilizando el sistema láser del robot con módulos ViRbot y sus adaptaciones para HSR, se obtuvo un mapa del entorno real navegando el robot en el espacio real. Esta imagen de mapa fue transferida a Unity 3D en una posición y escala adecuadas. Con esta imagen como base para el piso, los objetos fijos como paredes, techo, piso o muebles de gran tamaño fueron representados en el entorno virtual mediante modelos 3D en una posición fija correspondiente a su posición real.

Para ubicar al robot en el espacio virtual, el robot real utiliza sus sensores y bibliotecas ROS para ubicarse y transmitir su posición a la interfaz para replicarlo en el entorno virtual. La posición del robot en relación con el mapa se solicita continuamente. Una vez obtenida la posición, se envía al entorno virtual a través de sockets. En el entorno virtual, la información es recibida y asignada al modelo de robot para posicionarlo en la ubicación correspondiente.

Figura 5.5: Escena virtual en Unity

La implementación del sistema está dividida en dos partes principales, Unity y ROS. En Unity se tiene una escena que contiene todos los elementos de la interfaz. Esta escena se presenta en el HMD. Una cámara virtual mapeada con el HMD se coloca en una posición inicial dentro de la escena (figura 5.5). La cámara está mirando hacia el entorno virtual para visualizar todos los elementos virtuales en la escena. Un par de planos 2D se colocan en la posición correspondiente con el HMD para cada ojo. Esos planos mostrarán el video real al usuario. Las imágenes de las cámaras estéreo se utilizan como textura para estos planos, lo que proporciona el efecto 3D.

Al seleccionar el modo de visualización real, los planos se habilitan y el video se muestra en los planos 2D. Al cambiar al modo virtual, los planos se desactivan y se puede observar el entorno virtual.

Para controlar el robot, Unity monitorea el *joystick* para detectar el movimiento del stick o la presión de un botón y convertir esto en mensajes que se envía a través de sockets. Un proceso similar se aplica al movimiento de la cabeza, pero monitoreando el movimiento del HMD desde Unity. Los mensajes generados se envían a ROS para ser procesados y ejecutados y a cambio se recibe nueva información del robot en Unity para actualizar la posición de la cámara virtual y la rotación de acuerdo con las posiciones de la base y la cabeza del robot real. En ROS, los mensajes llegan a través de sockets, luego se convierten en comandos y se envían al robot para su ejecución. En respuesta, la información actualizada del robot se convierte en mensajes y se devuelve a Unity a través de sockets.

Para la transmisión de video se utiliza el paquete ROS *"Web Video Server"*. En pocas palabras, el servidor de video web abre un puerto local y espera las solicitudes HTTP entrantes, cuando se solicita una transmisión de video de un tópico de imagen ROS a través de HTTP, se suscribe al tópico correspondiente y crea una instancia del codificador de video. Los paquetes de video en bruto codificados se envían al cliente. Solo se requiere una URL con algunos parámetros para conectarse al nodo [115].

El sistema fue probado en diversos contextos e incluso mostrado como demo en algunos eventos de robótica internacionales teniendo un buen nivel de aceptación. Las pruebas sugieren que el sistema logró mostrar con éxito video en 3D de las cámaras del robot en el HMD y proporcionar al operador una experiencia inmersiva convincente. Los usuarios interactuaron intuitivamente con la interfaz y el entorno remoto, pero informaron de algunas dificultades para visualizar el entorno 3D real. Los retrasos momentáneos en la pantalla de video hicieron que cambiaran a una estrategia de "moverse y esperar". De lo contrario, experimentaban mareos y otros efectos incómodos. De cualquier manera, la mayoría de ellos estaban satisfechos con la experiencia de "ver lo que veía el robot". La capacidad de mover la cabeza del robot con la cabeza fue bien recibida, y poder visualizarlo en 3D fue una gran adicción. Incluso con la estrategia de "moverse y esperar", la exploración visual y la navegación del robot tuvieron éxito.

El mecanismo de cambio entre modos *real-virtual* se implementó en un botón de *joystick*, lo que permitió a los usuarios cambiar de modo a voluntad. Al cambiar al modo virtual, los usuarios informaron una mejora significativa en las tareas de visualización y navegación. Por un lado, los usuarios prefirieron usar el modo virtual para la navegación y solo cambiaron al modo real para confirmar que habían llegado a la ubicación deseada. Por otro lado, se prefirió la exploración visual en modo real incluso con el tema de la espera. Afirmaron que el modo real les permitió apreciar con detalle la ubicación remota real.

De acuerdo a lo observado, la navegación y otros movimientos del robot real fueron imitados por el robot virtual adecuadamente. En resumen, los usuarios expresaron que el sistema era fácil de aprender a manipular gracias al control *joystick*, que brindaba la sensación de controlar un videojuego. La visualización en 3D fue una de las características que más disfrutaron los usuarios, una parte significativa de ellos nunca había experimentado o tenía poca experiencia con una pantalla 3D montada en la cabeza como Oculus. La visualización real fue bien recibida, pero los retrasos provocados por la latencia molestaron a los usuarios. Las principales quejas fueron mareos, fatiga y algo de frustración al intentar realizar una tarea con rapidez. Al cambiar al modo virtual, todos los usuarios sintieron una mejora sustancial, el mareo y la fatiga desaparecieron por completo. El desempeño también mejoró y la frustración disminuyó. Una vez que los usuarios se sintieron cómodos con el modo de visualización dual, cambiaron a voluntad de un modo a otro para completar las tareas deseadas. La mayoría de los usuarios quedaron satisfechos con la experiencia y declararon que utilizarían este sistema en situaciones de la vida real. También expresaron felicidad al usar el sistema, ya que a veces se sentía como una experiencia de videojuego. Además, los comentarios de los usuarios fueron de gran utilidad a la hora de pulir las funciones del sistema y planificar otras nuevas.

Los robots de servicio teleoperados son una excelente opción para combinar la inteligencia humana y las habilidades de movimiento con las funciones de los robots de servicio. Sin embargo, aún quedan muchas mejoras y nuevas funcionalidades posibles para este sistema, como por ejemplo, mejoras en modelos 3D para una experiencia más realista en el entorno virtual, reconocimiento de objetos reales dinámicos y representación virtual, reconocimiento de personas y su representación con avatares en el entorno virtual para agregar interacción *humano-robot*, añadir más funcionalidades autónomas como colocación de objetos, por mencionar algunas. Este sistema se sigue manteniendo y desarrollando para incrementar sus funcionalidades, por ejemplo, se consideró hacerle varias mejoras al integrarlo con el Selector de Estrategias y el proyecto de detección de objetos que se describe a continuación.

5.2. Sistema de detección de objetos

Otro proyecto desarrollado en el laboratorio de BioRobótica consiste en un sistema capaz de generar una base de datos de imágenes de objetos cotidianos etiquetadas a partir de un video. Estas imágenes posteriormente se utilizaron para entrenar una CNN tipo YOLO, la cual forma parte de un sistema de detección de objetos que se utiliza para que el robot *Takeshi* pueda manipular objetos autónomamente [96]. Este proyecto se divide principalmente en dos sistemas de visión computacional, uno que genera el conjunto de escenas sintéticas etiquetadas para reentrenar a la CNN YOLOv3 y el otro que plantea técnicas para detectar y manipular objetos autónomamente con el robot de servicio.

El sistema de creación del conjunto de escenas sintéticas se compone a su vez de dos módulos, ambos desarrollados en lenguaje *C++*. El primero hace el procesamiento de bajo y medio nivel de una secuencia de imágenes provenientes del video y el segundo crea el conjunto de escenas ya etiquetado para reentrenar a la CNN YOLOv3 [96]. El video del cual se segmentan los objetos debe contener el objeto de interés colocado sobre un fondo controlado. Este fondo controlado debe ser de un color uniforme y fácil de segmentar en el espacio de color HSV. La captura debe ser de tal manera que se abarquen los 360° alrededor del objeto y de 0 a 90° en el eje vertical del objeto. El sistema elige los parámetros de color dependiendo del video de entrada. El color del fondo controlado debe ser elegido de manera que contraste con el objeto a segmentar y que dicho objeto no contenga colores similares a los del fondo para evitar quitar pixeles del objeto en el momento de la segmentación.

Para realizar la segmentación de objetos, se aplican algunas técnicas de procesamiento de imágenes como transformaciones, filtros y otros algoritmos para detectar bordes del objeto en cuestión, luego se efectúan algunas operaciones morfológicas para unir los bordes del contorno y delimitar adecuadamente el objeto, etc. Después de varios procesos se obtiene la imagen del objeto segmentado (figura 5.6). Con éste primer módulo se pueden generar aproximadamente 2500 imágenes de objetos segmentados diferentes por cada video de aprox. 45 segundos.

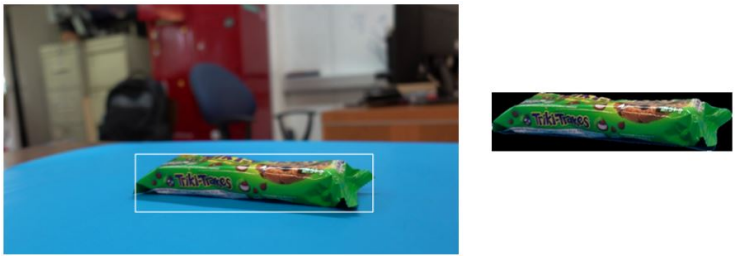


Figura 5.6: Segmentación de objetos. Izquierda: Objeto con fondo controlado. Derecha: Objeto segmentado (Imágenes tomadas de [96])

El segundo módulo utiliza el conjunto de imágenes segmentadas obtenido y un conjunto de escenas reales para crear el conjunto de escenas sintéticas las cuales servirán para entrenar la CNN YOLOv3. Las escenas reales son imágenes capturadas de escenas donde podrían encontrarse los objetos en cuestión, por ejemplo, mesas, escritorios, muebles, etc.

Para iniciar el proceso, se aplican técnicas de aumento de datos (desplazamientos, cambios de tamaño, etc.) a los objetos segmentados obtenidos en el módulo anterior. Luego se colocan los objetos procesados en la escena real, para lograr esto se tiene un modo manual, donde el usuario debe seleccionar el lugar donde se pueden colocar objetos (lo cual resulta un poco complejo y tardado)

y un modo automático donde el sistema coloca los objetos usando una técnica de collage (en renglones y columnas por toda la imagen). Al mismo tiempo se genera un archivo de texto con los datos de etiquetado de cada objeto en la escena (compatible con Darknet YOLO). Posteriormente a la mitad de las escenas sintéticas generadas se les aplican otras técnicas de aumento de datos como cambio de brillo y contraste y difuminación para robustecer el conjunto resultante (figura 5.7).

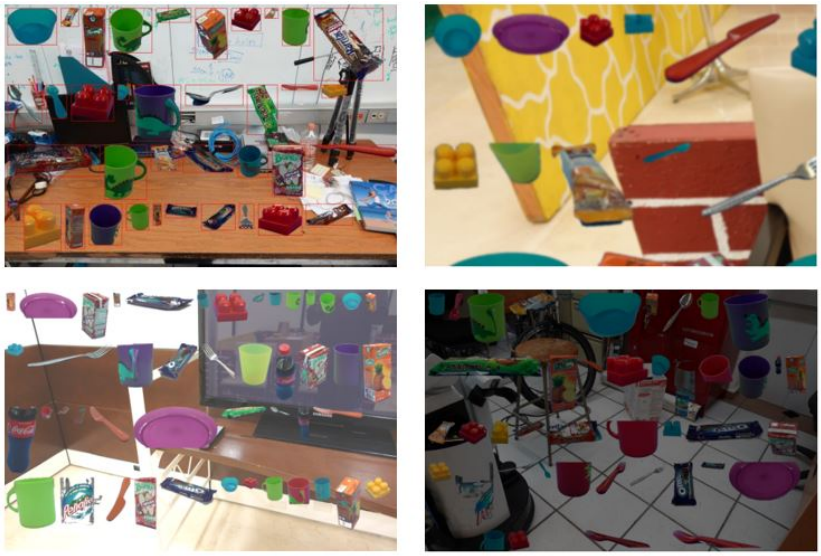


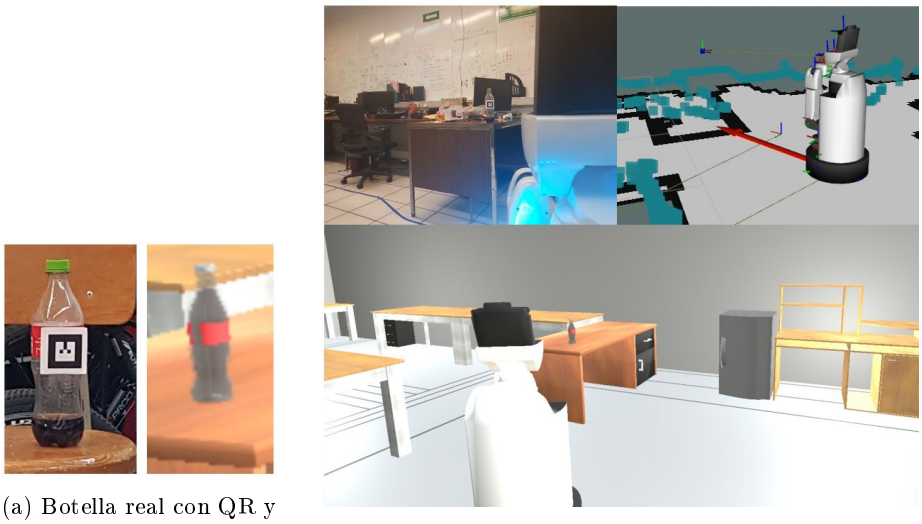
Figura 5.7: Escenas sintéticas generadas con bounding boxes, difuminación, cambio de brillo y contraste (Imágenes tomadas de [96])

El siguiente paso es usar el conjunto de escenas sintéticas generado para reentrenar la CNN usando transferencia de conocimiento de un modelo YOLOv3 preentrenado con el conjunto de datos de ImageNet [94]. Una vez configurados adecuadamente los parámetros del modelo se procede a entrenar la red partiendo de los pesos preentrenados con el conjunto de ImageNet. Con esto se pretende que la red reconozca los objetos que se encuentran en las imágenes sintéticas.

El sistema de detección de objetos además de detectar los objetos también proporciona la posición del objeto para posteriormente manipularlo. Para lograr esto, contiene un nodo ROS que interactúa con DarkNet-ROS [116] para hacer la detección más precisa. Este nodo captura una imagen RGB-D de la cámara del robot y la publica en un tópico configurado para DarkNet ROS, luego espera el resultado del nodo de detección y evalúa los objetos detectados, finalmente da las coordenadas de posición del objeto para poder hacer la manipulación.

5.3. Representación de objetos reales en ambientes virtuales

Existen objetos en el ambiente en el que se desempeña el robot de servicio los cuales no tienen una posición fija o definida ya que pueden ser movidos o cambiados de posición en el ambiente, esto significa que son objetos dinámicos. Estos objetos dinámicos son fácilmente visualizados por el robot a través de sus cámaras, sin embargo, cuando se navega en un ambiente virtual mapeado al ambiente real como en el sistema de teleoperación mencionado anteriormente, es difícil representar dichos objetos en el ambiente virtual.



(a) Botella real con QR y su representación virtual

(b) Localización de objetos usando etiquetas QR

Figura 5.8: Representación de objetos usando QR (Imágenes tomadas de [48])

En el sistema de teleoperación original [48], se intentó abordar este problema utilizando etiquetas con códigos QR (Quick Response code) pegadas sobre los objetos dinámicos (figura 5.8a) y usar algunas bibliotecas de ROS que detectan y ubican en el ambiente dichas etiquetas y usar esta información para representarla en el ambiente virtual [48]. Para este caso se utilizó la biblioteca *"ar track alvar"* la cual permite crear etiquetas QR (códigos de barra en forma de matriz) y posteriormente identificarlas y ubicarlas en el espacio usando las cámaras del robot. A cada objeto dinámico se le asigna una etiqueta para identificarlo y se pega la etiqueta al objeto, posteriormente, cuando el robot navega y detecta con sus cámaras el QR de la etiqueta, obtiene la información del objeto en cuestión (tipo, posición, orientación). Esta información se envía al ambiente remoto para poder representarlo en la ubicación equivalente dentro del mundo virtual (figura 5.8b).

Este desarrollo original si bien fué exitoso requería siempre de etiquetas QR pegadas en los objetos para que los objetos pudieran ser "visualizados" adecuadamente en el mundo virtual. Para evitar el uso de dichas etiquetas pegadas en los objetos, se implementó como nueva funcionalidad (agregada al sistema original) el reconocimiento de objetos con el sistema de detección de objetos para su representación en el ambiente virtual (figura 5.9). Haciendo las adaptaciones necesarias en ambos sistemas, el sistema de detección de objetos reconoce los objetos en las imágenes provenientes de las cámaras del robot y manda la información de la clase de objeto y ubicación para que el ambiente virtual lo represente en la posición correspondiente.

Figura 5.9: Representación virtual de objetos reconocidos en el ambiente real

5.4. Integración de sistemas con el Selector

El Selector de Estrategias y los sistemas de teleoperación y detección de objetos funcionan adecuadamente por su cuenta, sin embargo, para aprovechar los beneficios que todos ellos ofrecen, se debe realizar una integración de sistemas y con esto obtener un sistema mayor y más eficiente que cada uno por separado. La idea general es integrar el sistema de teleoperación con el Selector de Estrategias y el sistema de detección de objetos para mejorar sus funcionalidades.

La integración propuesta funciona de la siguiente manera (figura 5.10): El robot opera en el ambiente real remoto en donde con sus cámaras obtiene imágenes del ambiente y los objetos reales, el sistema de teleoperación envía las imágenes obtenidas y la información del robot a través de Internet para que la interfaz de operación las reciba y las visualice o bien las represente en el ambiente virtual. Al mismo tiempo, las imágenes de la cámara del robot son enviadas al sistema de detección de objetos en donde los objetos son reconocidos y la información obtenida también es enviada al sistema de teleoperación para su representación en el ambiente virtual (dado que en el ambiente real los objetos ya existen, por lo que no es necesaria ninguna modificación para visualizarlos). Los objetos detectados son representados en el ambiente virtual usando la información recibida y el ambiente virtual pre-mapeado existente.

Hasta este punto, el usuario ya es capaz de visualizar los objetos dinámicos en el mundo virtual en la posición correspondiente al mundo real. Entonces para poder realizar la colocación de un objeto en el ambiente remoto autónomamente, se envían las imágenes del ambiente virtual con los objetos generados para que el Selector de Estrategias pueda hacer sus predicciones y elegir la estrategia de colocación adecuada. Una vez seleccionada la estrategia, el Selector la envía al robot para su ejecución.

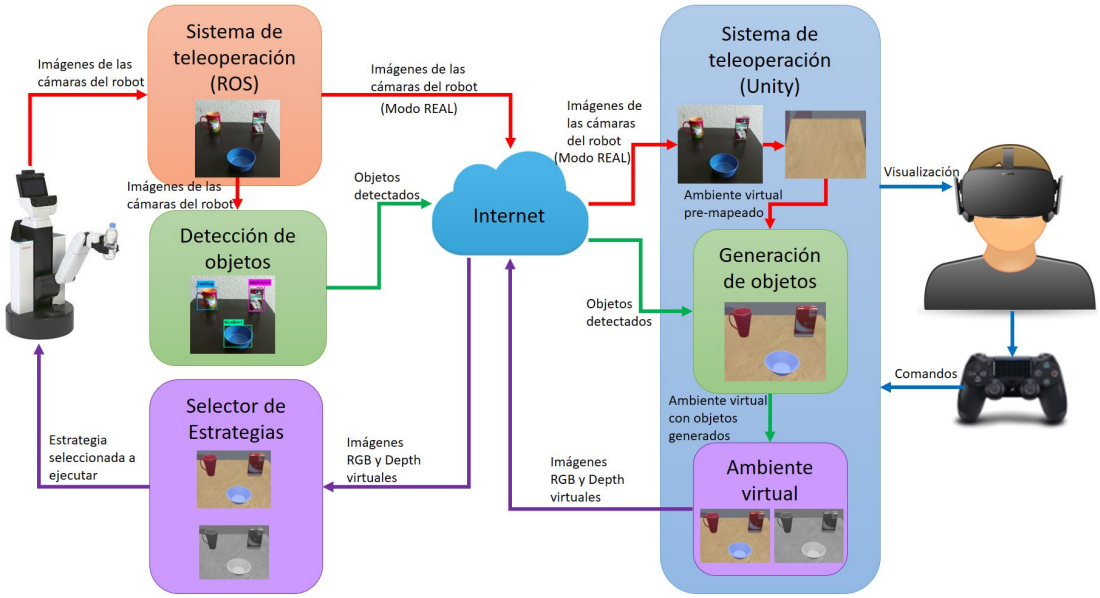


Figura 5.10: Diagrama de integración de sistemas

Como es de esperar en cualquier proyecto de integración de sistemas, surgen varios retos a superar para lograr la correcta integración y funcionamiento del sistema resultante, algunos de ellos son relativamente fáciles y otros pueden requerir realizar algunas modificaciones o adaptaciones creativas a uno o todos los sistemas. Para el presente caso, se describen brevemente las situaciones más relevantes que surgieron al integrar los 3 sistemas.

Primeramente, fue necesario adecuar un ambiente de trabajo en el cual todos los sistemas pudieran funcionar adecuadamente y sin problemas de compatibilidad. Algunos sistemas no tuvieron inconvenientes y otros tuvieron que migrarse, por ejemplo, el simulador SIGVerse en su versión 3 (junio de 2021) requiere correr en dos equipos, uno con Windows 10 y Unity 2018.4.11 y el otro con Ubuntu 16 y ROS Kinetic (figura 5.11). Sin embargo, para evitar la necesidad de usar dos computadoras con dos sistemas operativos distintos, se hizo la migración de Windows a Ubuntu, utilizando la versión de Unity Hub 2.2.2 para Ubuntu.

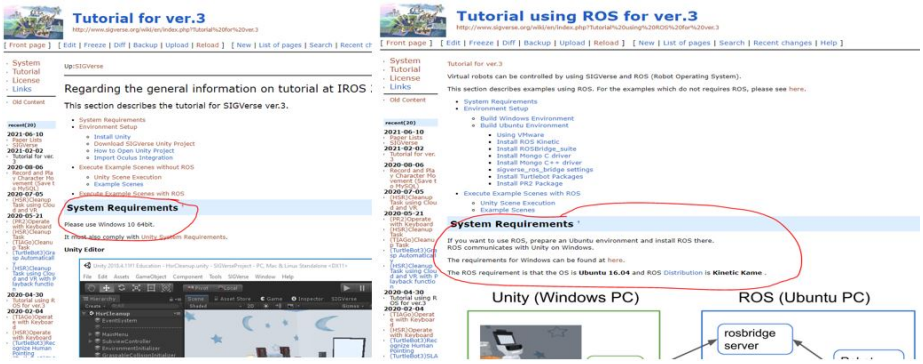


Figura 5.11: Requerimientos para SIGVerse

El siguiente punto a considerar fueron las versiones con las que trabajaba SIGVerse, por ejemplo, en el caso de ROS se migró de Kinetic a Melodic, que es la versión con la que trabaja Takeshi hasta el momento, también cabe mencionar que se migró de Ubuntu 16 a Ubuntu 18 por la misma razón. Después de algunas configuraciones, adecuaciones y pruebas, finalmente se logró que SIGVerse funcionara en Ubuntu 18, ROS Melodic y Unity sobre Ubuntu (figura 5.12).

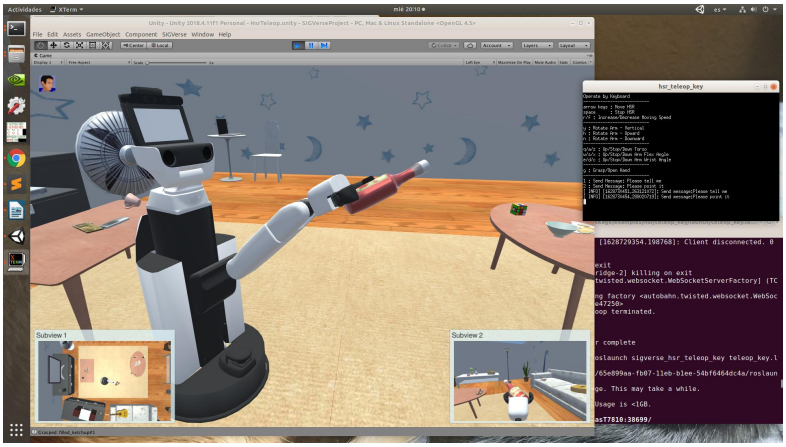


Figura 5.12: SIGVerse en Ubuntu

Como nota adicional, el simulador ya había sido migrado de Windows a Ubuntu para acelerar el proceso de obtención del conjunto de datos del Selector de Estrategias, dado que el proceso era más lento cuando se utilizaban dos computadoras y una conexión entre ellas (incluso con cable directo), pero esto se había realizado en Ubuntu 16 con ROS Kinetic.

Para el sistema de reconocimiento de objetos fue necesario instalar y configurar el paquete DarkNet YOLO y YOLO ROS, una vez hecho esto, se probó que funcionara correctamente en el mismo ambiente donde ya se tiene SIGVerse y el Selector. Primero se probó con modelos y pesos preentrenados con ImageNet y posteriormente se probaron los modelos y pesos del sistema de reconocimiento de objetos desarrollado en el laboratorio (figura 5.13).

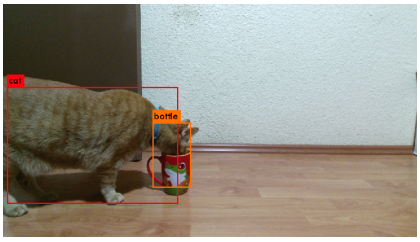


Figura 5.13: Reconocimiento con DarkNet YOLO

Una vez que DarkNet YOLO y SIGVerse funcionaron adecuadamente en el mismo ambiente, se realizaron adecuaciones para que las imágenes del simulador pudieran ser usadas con YOLO para el reconocimiento de objetos, es decir, que hiciera el reconocimiento de objetos en el ambiente virtual. Dadas las similitudes de los objetos reales a los virtuales del simulador, se logró que funcionara aceptablemente el reconocedor con imágenes capturadas en SIGVerse (figura 5.14).

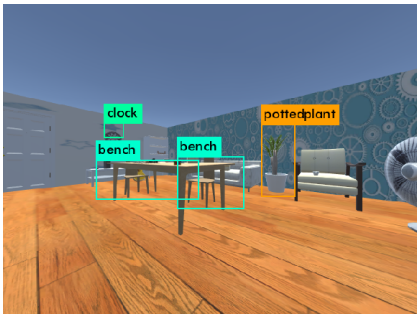


Figura 5.14: Reconocimiento usando DarkNet YOLO en SIGVerse

Para el sistema de teleoperación se requiere el paquete de ROS *"Web Video Server"*. Este paquete abre un puerto local y espera las solicitudes HTTP entrantes. Cuando se solicita un tópico de imagen vía HTTP, se suscribe al tópico correspondiente y crea una instancia del codificador de video. Los paquetes de video en bruto codificados se envían al cliente. Para conectarse al nodo con un navegador se usa la URL: *http://localhost:8080/stream?topic=ROS_TOPIC*. Para probar su correcto funcionamiento, se utilizó con el simulador SIGVerse,

para visualizar sus cámaras de manera similar a como se hace con el robot real (figura 5.15).

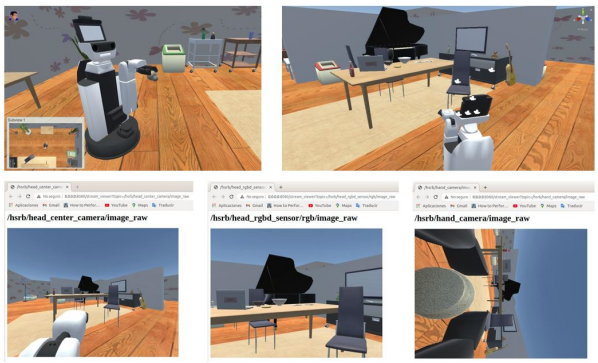


Figura 5.15: *Web Video Server*. Arriba: Escena Unity desde dos puntos de vista. Abajo: Visualización de 3 cámaras del robot

Para poder realizar pruebas durante la pandemia de 2020-2021, dado que era complicado acceder al robot real por la situación de confinamiento, se utilizó un Kinect 2 (el Kinect para Xbox One) para suplir las cámaras del robot y hacer algunas de las pruebas. Para poder utilizar el Kinect 2 como cámara habilitada para ROS se requirió primero instalar los controladores para que Ubuntu reconociera el dispositivo y se pudiera comunicar adecuadamente. En este caso se utilizó el driver *"libfreenect2"* [117] para poder obtener la información de las cámaras del Kinect, este driver cuenta con una herramienta para visualizar las imágenes obtenidas por el Kinect 2 en varias modalidades (figura 5.16).

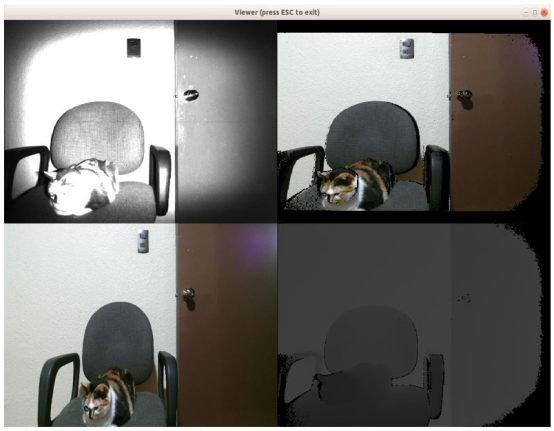


Figura 5.16: Herramienta de visualización para Kinect 2

Adicionalmente se requiere de una librería para comunicar el Kinect 2 con ROS, se utilizó *"IAI Kinect2"* [118] para poder mandar las imágenes de video del kinect en tópicos de ROS como lo hacen las cámaras del robot real (figura 5.17).

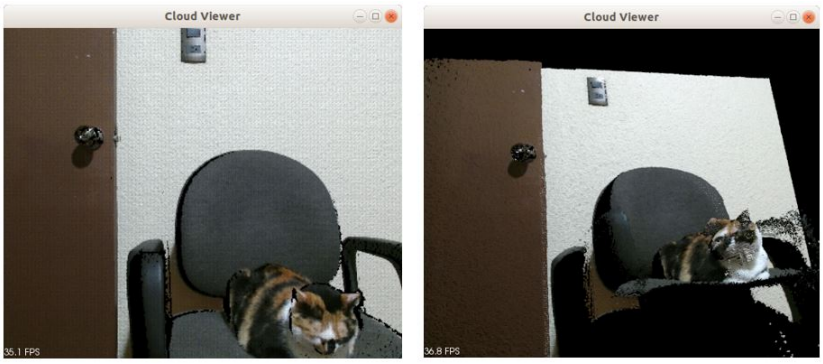


Figura 5.17: Visualización de Kinect 2 usando ROS

Una vez listo el ambiente con todos los sistemas a utilizar probados y funcionando adecuadamente, se procedió a implementar la integración explicada anteriormente y los experimentos que se detallan en la siguiente sección.

Capítulo 6

Experimentos y resultados

6.1. Entrenamiento de la red neuronal profunda

Para entrenar adecuadamente la red neuronal se utilizó el conjunto de datos (*dataset*) obtenido con el proceso descrito en la sección 4.1. Antes de usar el conjunto de datos para entrenar la red neuronal, todo el conjunto de datos se sometió al proceso de pre-procesamiento descrito en la sección 4.3.

Para los experimentos se utilizaron 4 estrategias simples ($S = 4$) definidas por los siguientes movimientos del robot:

1. Moverse un paso a la izquierda, estirar el brazo hacia el frente y abajo, abrir la mano para soltar el objeto.
2. No moverse, estirar el brazo hacia el frente y abajo, abrir la mano para soltar el objeto.
3. Moverse un paso a la derecha, estirar el brazo hacia el frente y abajo, abrir la mano para soltar el objeto.
4. Moverse dos pasos a la derecha, estirar el brazo hacia el frente y abajo, abrir la mano para soltar el objeto.

El número de estrategias puede ser incrementado dependiendo de las capacidades del robot y movimientos totalmente probados. En este caso existen 4 ROIs correspondientes a cada estrategia, entonces, la imagen original se divide en 4 ROIs : $R_1(i), R_2(i), R_3(i), R_4(i)$ (Fig. 6.1).

El tamaño original de las imágenes es de 640×480 y el de las ROIs de 192×192. Además del proceso para transformar las imágenes ROIs de profundidad a *surface normals*, todas las ROIs (RGB y SN) fueron reescaladas a 224×224 y procesadas para ResNet en los casos en los que se utilizó ResNet-50 preentrenado como parte del modelo de la red y en los experimentos base.

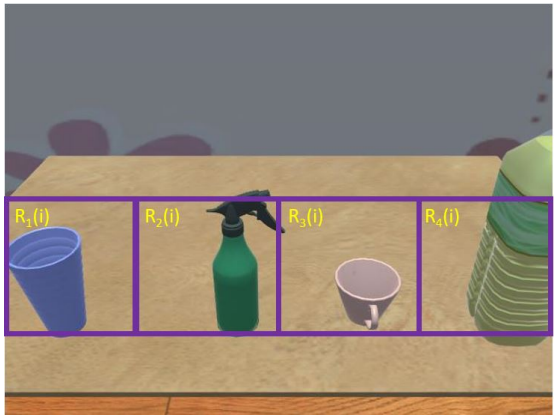


Figura 6.1: Imagen dividida en 4 ROIs

En dichos casos, los modelos preentrenados fueron inicializados con los pesos de ImageNet [119] y se refinó usando el conjunto de datos creado.

El conjunto de datos preprocesados se dividió en tres subconjuntos (típicos para entrenar redes neuronales): entrenamiento (*training* 80 %) , validación (*validation* 10 %) y pruebas (*test* 10 %). El tamaño total del conjunto de datos es de 15, 028 muestras, el conjunto de entrenamiento es de 12, 022 muestras y los conjuntos de validación y pruebas de 1, 503 muestras cada uno. Todos los modelos fueron entrenados usando este conjunto de datos (DS15K).

Las características generales del conjunto de datos (DS15K) son las siguientes:

- Recolectado usando la simulación basada en SIGVerse.
- Utiliza un HSR virtual para realizar la tarea de colocación.
- Escenas generadas automáticamente usando una variedad de objetos y muebles cotidianos.
- Conjunto reducido de 4 estrategias de colocación simples consistentes en movimientos simples y probados.
- Tamaño del conjunto de datos 15,028 muestras (*train:valid:test* = 8:1:1).
- Imágenes RGB y *Depth* (640 x 480) preprocesadas.
- Etiquetado automático usando el mismo criterio basado en un umbral de colisiones.
- Archivo de etiquetas Y binarios (éxito o fracaso).

Los modelos fueron implementados con TensorFlow y Keras. El modelo ResNet-50 de Keras se dividió en la capa 25 para generar el extractor de características y para la rama de percepción se usaron las capas siguientes hasta la 43 (figura 6.2). El entrenamiento de la red se llevó a cabo usando un GPU Nvidia Quadro P4000 con 8 GB de memoria. El método de evaluación utilizado es la exactitud (accuracy) sabiendo que usamos $X = 2$ clases diferentes. Los modelos fueron guardados cuando se obtenía la mejor exactitud (accuracy) en el conjunto de validación.

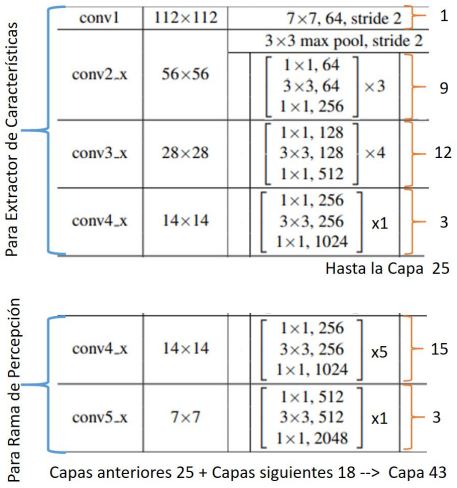


Figura 6.2: Modelo ResNet-50 dividido para el extractor de características y la rama de percepción

Primero se realizaron algunos experimentos base usando redes tipo *Vanilla CNN*. Para este caso se implementó una red propia tipo *Vanilla* para probar el enfoque propuesto en este proyecto. Cabe mencionar que en estos experimentos base solamente se tienen un tipo de entradas, ya que no cuenta con los dos flujos ni la arquitectura mostrada anteriormente, simplemente una red *Vanilla CNN* consistente en algunas capas convolucionales(*Conv2D*) seguidas de agrupaciones máximas (*Max Pooling*) y posteriormente un aplanamiento (*Flattening*) para concatenar los metadatos seguidos de algunas capas totalmente conectadas (*Fully Connected*) seguidas de normalizaciones por lotes (*Batch Normalization*).

Se hicieron pruebas usando como entrada solamente las imágenes RGB (3 canales), luego solo las imágenes de profundidad (1 canal) y por último se utilizó una entrada RGB-D (4 canales) agregando los canales de RGB y *Depth*. Se hicieron algunas variaciones como por ejemplo el tamaño del batch para buscar las mejores condiciones. Inicialmente no se tenía el conjunto de datos completo, ya que obtenerlo llevó algo de tiempo. Entonces se tomó un subconjunto más pequeño (4,500 muestras) al que denominaremos DS4500.

Usando este nuevo conjunto de datos se hicieron los entrenamientos iniciales. Una vez que el conjunto de datos completo (DS15K) estuvo listo, se probó esta misma red con dicho *dataset*, nuevamente probando con RGB, *Depth* y RGB-D. Los resultados se muestran en las figuras 6.3.

Dataset	DS4500								
Modelo	Vanilla (Propio)								
Batch Size	32								
DEPTH	Epochs	100	RGB	Epochs	100	RGB-D	Epochs	100	
	LR	0.0001		LR	0.0001		LR	0.0001	
Closed Test	Validation	Test	Closed Test	Validation	Test	Closed Test	Validation	Test	
0.8497253	0.8307692	0.8241758	0.9799451	0.7956044	0.8	0.8162088	0.8197802	0.8131868	
0.8711538	0.8241758	0.8153846	0.8980769	0.7736264	0.7846154	0.8387363	0.8021978	0.8087912	
0.8502747	0.810989	0.7978022	0.9376374	0.7824176	0.7824176	0.8151099	0.810989	0.8065934	
0.8417582	0.8043956	0.8087912	0.9741758	0.7846154	0.7912088	0.8428571	0.8197802	0.8021978	
0.8615385	0.7956044	0.8197802	0.9494505	0.7802198	0.7846154	0.8098901	0.7978022	0.8153846	
0.8548901	**0.8131868**	**0.8131868**	**0.9478571**	**0.7832967**	**0.7885714**	**0.8245604**	**0.8101099**	**0.8092308**	AVG
0.0115014	**0.0143279**	**0.0103086**	**0.0328152**	**0.0080152**	**0.0071892**	**0.0150831**	**0.0100235**	**0.0052472**	STD

(a) Conjunto de datos DS4500, batch size de 32, tasa de aprendizaje de 0.0001 y 100 épocas.

Dataset	DS4500								
Modelo	Vanilla (Propio)								
Batch Size	64								
DEPTH	Epochs	300	RGB	Epochs	300	RGB-D	Epochs	300	
	LR	0.0001		LR	0.0001		LR	0.0001	
Closed Test	Validation	Test	Closed Test	Validation	Test	Closed Test	Validation	Test	
0.8508242	0.8153846	0.8131868	0.967033	0.7714286	0.767033	0.9041209	0.8021978	0.832967	
0.8576923	0.8131868	0.7912088	0.9835165	0.7714286	0.7846154	0.8947802	0.8153846	0.8241758	
0.8302198	0.8043956	0.7978022	0.9854396	0.7802198	0.7538462	0.8543956	0.7956044	0.8087912	
0.8120879	0.7956044	0.7934066	0.9076923	0.7626374	0.7978022	0.8697802	0.8021978	0.7934066	
0.8274725	0.810989	0.7956044	0.9642857	0.7736264	0.7824176	0.881044	0.7912088	0.8131868	
0.8356593	**0.8079121**	**0.7982418**	**0.9615934**	**0.7718681**	**0.7771429**	**0.8808242**	**0.8013187**	**0.8145055**	AVG
0.0184909	**0.0080152**	**0.0087084**	**0.0315889**	**0.0062935**	**0.0169957**	**0.0197357**	**0.0091414**	**0.0151154**	STD

(b) Conjunto de datos DS4500, batch size de 64, tasa de aprendizaje de 0.0001 y 300 épocas.

Dataset	DS15K								
Modelo	Vanilla (Propio)								
Batch Size	64								
DEPTH	Epochs	300	RGB	Epochs	300	RGB-D	Epochs	300	
	LR	0.0001		LR	0.0001		LR	0.0001	
Closed Test	Validation	Test	Closed Test	Validation	Test	Closed Test	Validation	Test	
0.8285643	0.7957418	0.7950765	0.9851938	0.835662	0.8283433	0.9624854	0.83167	0.8203593	
0.8866245	0.8250166	0.8310047	0.9928464	0.8369927	0.8250166	0.9031775	0.8230206	0.823686	
0.8471136	0.823686	0.8243513	0.9307104	0.83167	0.8210246	0.9538346	0.825682	0.8210246	
0.8406255	0.8243513	0.837658	0.9990018	0.839654	0.8250166	0.9544169	0.83167	0.8097139	
0.8636666	0.8196939	0.8223553	0.9842788	0.8323353	0.8270126	0.9387789	0.8183633	0.8117099	
0.8533189	**0.8176979**	**0.8220892**	**0.9784063**	**0.8352628**	**0.8252828**	**0.9425387**	**0.8260812**	**0.8172987**	AVG
0.0225209	**0.0124473**	**0.0162538**	**0.027332**	**0.0033133**	**0.0027673**	**0.0236114**	**0.005735**	**0.0061808**	STD

(c) Conjunto de datos DS15K, batch size de 64, tasa de aprendizaje de 0.0001 y 300 épocas.

Figura 6.3: Resultados para la red propia tipo *Vanilla*

En estos resultados se muestran en cada tabla los resultados de 5 entrenamientos, donde la columna *Closed Test* se refiere a los resultados de predicción usando el conjunto de entrenamiento mismo, la columna *Validation* los resultados usando el conjunto de validación (usado para evaluar el desempeño y mejorar en las siguientes épocas o iteraciones) y la columna *Test* los resultados usando

el conjunto de pruebas (datos no vistos antes por la red en el entrenamiento). Se realizaron más experimentos (además de los reportados) para encontrar los parámetros óptimos para tasa de aprendizaje, tamaño del lote, numero de épocas etc. Los mismos se fueron modificando de acuerdo a los resultados observados. Se reportan los más relevantes.

Posteriormente se decidió probar cambiando la red *Vanilla CNN* por una tipo *ResNet* preentrenada, en este caso se utilizó un modelo ResNet50 de Keras preentrenado con Imagenet. Para poder utilizar este modelo, las entradas de *Depth* tuvieron que ser transformadas a imágenes tipo RGB (de 3 canales) por lo que se convirtieron a imágenes SN con el proceso descrito anteriormente. Entonces se procedió a probar nuevamente con entradas RGB y después con entradas SN. Para el caso de las RGB-D se utilizaron las RGB y SN (6 canales) uniéndolas con un proceso de concatenación de canales para obtener una entrada adecuada para la red (3 canales). Los resultados obtenidos se muestran en la figura 6.4.

Dataset	DS15K								
Modelo	ResNet50 (Keras)								
Batch Size		64					Batch Size		32	
DEPTH	Epochs	100	RGB	Epochs	100		RGB-D	Epochs	100	
	LR	0.0001		LR	0.0001			LR	0.0001	
Closed Test	Validation	Test	Closed Test	Validation	Test		Closed Test	Validation	Test	
0.9987523	0.8403194	0.8176979	0.9990018	0.8409847	0.8270126		0.9985859	0.8403194	0.8143713	
0.9989186	0.8369927	0.82169	0.9988355	0.8389887	0.8196939		0.9984196	0.8423154	0.8223553	
0.997255	0.8389887	0.83167	0.9987523	0.8250166	0.8183633		0.9911828	0.8303393	0.8203593	
0.9984196	0.8369927	0.8203593	0.9984196	0.8323353	0.8157019		0.9982532	0.8429807	0.825682	
0.99817	0.837658	0.8183633	0.9985027	0.835662	0.8183633		0.9979205	0.8409847	0.8150366	
0.9983031	**0.8381903**	**0.8219561**	**0.9987024**	**0.8345975**	**0.819827**		**0.9968724**	**0.8393879**	**0.8195609**	AVG
0.0006539	**0.0014424**	**0.0056573**	**0.0002396**	**0.0062873**	**0.0042706**		**0.0031901**	**0.0051665**	**0.00483**	STD

Figura 6.4: Resultados para la red ResNet50 de Keras, con el conjunto de datos DS15K, batch size de 64, tasa de aprendizaje de 0.0001 y 100 épocas.

Lo siguiente fue implementar una red tipo ABN similar a las que formarán los flujos gemelos en la red propuesta. Esta red *ABN Simple* al igual que las anteriores solamente tiene un flujo de entrada por lo que nuevamente se probaron las entradas RGB, *Depth* y RGB-D. Al igual que en las redes tipo ResNet anteriores, las entradas de tipo *Depth* tuvieron que ser procesadas para ser utilizadas dado que las arquitecturas de estas redes contienen secciones de ResNet50 preentrenadas. El proceso fue similar, es decir, convertir a SN las entradas *Depth* y en el caso de las RGB-D convertirlas a 3 canales.

En estos experimentos se probaron varias capas para recortar el modelo ResNet50 para el extractor de características y la rama de percepción, encontrándose los mejores resultados para los cortes en las capas 25 y 43 (que fueron también usados en el modelo propuesto final). Los resultados se muestran en las figuras 6.5.

Dataset	DS15K					
Modelo	Simple ABN (ResNet)					
Batch Size		48				
DEPTH	Epochs	50				
	LR	0.0005				
Closed Test		Validation		Test		
ABN	Total	ABN	Total	ABN	Total	
0.9961737	0.9974214	0.8476381	0.8536261	0.8469727	0.8576181	
0.9919315	0.999085	0.8403194	0.8569528	0.8529607	0.8589488	
0.92364	0.9957578	0.825682	0.8556221	0.8270126	0.8609448	
0.9950091	0.9977541	0.8483034	0.8562874	0.8483034	0.8509647	
0.9991682	0.9991682	0.8562874	0.8629408	0.8496341	0.8529607	
0.9811845	**0.9978373**	**0.843646**	**0.8570858**	**0.8449767**	**0.8562874**	AVG
0.0322726	**0.0013993**	**0.011524**	**0.0035017**	**0.0102858**	**0.0041816**	STD

(a) ABN Profundidad (Depth)

Dataset	DS15K					
Modelo	Simple ABN (ResNet)					
Batch Size		48				
RGB	Epochs	50				
	LR	0.0005				
Closed Test		Validation		Test		
ABN	Total	ABN	Total	ABN	Total	
0.8281484	0.8955249	0.8303393	0.8662675	0.8330007	0.8702595	
0.8423723	0.9845284	0.8383234	0.8742515	0.835662	0.8682635	
0.863833	0.9210614	0.8582834	0.8695941	0.8596141	0.8729208	
0.9752121	0.99817	0.8729208	0.8755822	0.8735862	0.8675981	
0.8627516	0.8959408	0.8569528	0.8715902	0.8656021	0.8589488	
0.8744635	**0.9390451**	**0.8513639**	**0.8714571**	**0.853493**	**0.8675981**	AVG
0.0582567	**0.0490915**	**0.0169993**	**0.0037164**	**0.0182052**	**0.0052599**	STD

(b) ABN RGB

Dataset	DS15K					
Modelo	Simple ABN (ResNet)					
Batch Size		48				
RGB-D	Epochs	50				
	LR	0.0005				
Closed Test		Validation		Test		
ABN	Total	ABN	Total	ABN	Total	
0.9992514	0.9993346	0.8722555	0.8755822	0.8689288	0.8622754	
0.9831975	0.9975046	0.8715902	0.8695941	0.8629408	0.8589488	
0.9780403	0.9850274	0.8502994	0.8662675	0.8443114	0.8649368	
0.9968391	0.9987523	0.8709248	0.8722555	0.8702595	0.8609448	
0.7610215	0.9201464	0.744511	0.8722555	0.7677977	0.8642715	
0.9436699	**0.9801531**	**0.8419162**	**0.871191**	**0.8428476**	**0.8622754**	AVG
0.1024945	**0.0340569**	**0.0552281**	**0.0034763**	**0.0432146**	**0.0024446**	STD

(c) ABN RGB-D (RGB + Depth)

Figura 6.5: Resultados para la red tipo ABN Simple

Estos resultados se componen de dos partes o salidas, siendo la primera (denotada como ABN) la salida de la rama de atención $P_{att}(i)$ y la segunda (denotada como Total) la salida final de la red $P_N(i)$ (después de la rama de percepción). Se muestran ambos resultados para observar más fácilmente el efecto de la rama de percepción, dado que en la mayoría de los casos la salida Total mejora la rama ABN.

A continuación, se probó unir dos ramas de atención (una para RGB y otra para *Depth*) para generar una sola red ABN múltiple (denotada como Multi ABN). Inicialmente las ramas se unieron con una concatenación simple en la rama de percepción. Se probaron varias capas para el recorte del modelo ResNet50 en ambas ramas para observar el efecto en los resultados y comprobar que las capas seleccionadas como finales (25 y 43) eran las óptimas. En estos resultados (figuras 6.6) se muestran 3 salidas, dos para las ramas ABN: ABN-Depth ($P_{att}^{SN}(i)$) y ABN-RGB ($P_{att}^{RGB}(i)$ y la salida final de la red : Total ($P_N(i)$).

Dataset	DS15K									
Modelo	Multi ABN (ResNet) 22 y 34									
Batch Size	48									
Epochs	50									
LR	0.0005									
Closed Test				Validation			Test			
ABN-Depth	ABN-RGB	Total		ABN-Depth	ABN-RGB	Total	ABN-Depth	ABN-RGB	Total	
0.996423224	0.9960905	0.99925137		0.853626081	0.8549568	0.8715902	0.850964736	0.8522954	0.8689288	
0.970637165	0.9967559	0.99925137		0.849634065	0.8496341	0.8662675	0.838988689	0.8536261	0.8502994	
0.992347363	0.9982532	0.99925137		0.846972721	0.8735862	0.8762475	0.834996673	0.8682635	0.8542914	
0.99900183	0.9992514	0.99925137		0.860944777	0.8642715	0.8709248	0.856952761	0.8502994	0.8516301	
0.998835468	0.9991682	0.99925137		0.840984697	0.8682635	0.8695941	0.836992681	0.8616101	0.8649368	
0.99144901	**0.9979038**	**0.99925137**		**0.850432468**	**0.8621424**	**0.8709248**	**0.843779108**	**0.8572189**	**0.8580173**	AVG
0.011940347	**0.0014267**	**0**		**0.007459482**	**0.0097603**	**0.0036137**	**0.009634745**	**0.0075215**	**0.0083843**	STD

(a) Usando modelo ResNet50 recortado en las capas 22 y 43

Dataset	DS15K									
Modelo	Multi ABN (ResNet)									
Batch Size	48									
Epochs	50									
LR	0.0005									
Closed Test				Validation			Test			
ABN-Depth	ABN-RGB	Total		ABN-Depth	ABN-RGB	Total	ABN-Depth	ABN-RGB	Total	
0.992513725	0.9954251	0.99908501		0.839654024	0.8589488	0.8715902	0.847638057	0.8629408	0.8642715	
0.998502745	0.9992514	0.99925137		0.848303392	0.8669328	0.8722555	0.846307385	0.8675981	0.8702595	
0.994759607	0.9986691	0.99916819		0.852960745	0.8675981	0.8695941	0.854956753	0.8636061	0.8742515	
0.996589586	0.9982532	0.99925137		0.848968729	0.8656021	0.8715902	0.852295409	0.8742515	0.8642715	
0.84162369	0.9174014	0.99109965		0.831004657	0.8829009	0.8689288	0.817697937	0.8755822	0.8702595	
0.964797871	**0.9818**	**0.99757112**		**0.84417831**	**0.8683965**	**0.8707917**	**0.843779108**	**0.8687957**	**0.8686627**	AVG
0.068892098	**0.03603**	**0.00361832**		**0.008819154**	**0.0088066**	**0.0014424**	**0.014991483**	**0.0058836**	**0.0043272**	STD

(b) Usando modelo ResNet50 recortado en las capas 25 y 43

Figura 6.6: Resultados para la red ABN Múltiple

Los resultados de ambas ramas de atención son importantes para la clasificación, sin embargo, en algunos casos la rama ABN-D mostro menor desempeño que la ABN-RGB, y por tanto el desempeño final disminuía. Para tratar de mejorar el resultado final de la red se decidió implementar el mecanismo de balanceo para evitar que los resultados finales se vieran afectados cuando una rama tenía menor desempeño (figura 6.7).

Dataset	DS15K								
Modelo	Multi ABN Balanceado (ResNet)								
Batch Size	48								
Epochs	30								
LR	0.0005								
Closed Test			Validation			Test			
ABN-Depth	ABN-RGB	Total	ABN-Depth	ABN-RGB	Total	ABN-Depth	ABN-RGB	Total	
0.865330228	0.8735651	0.92646814	0.857618096	0.8662675	0.8576181	0.866267465	0.8636061	0.8649368	
0.999168192	0.9992514	0.99925137	0.852960745	0.8669328	0.8709248	0.844311377	0.8682635	0.8656021	
0.850440858	0.8350524	0.89286308	0.838988689	0.8343313	0.8675981	0.858948769	0.8369927	0.8649368	
0.999168192	0.9992514	0.99925137	0.856952761	0.8602794	0.8656021	0.843646041	0.8636061	0.8722555	
0.997338213	0.9973382	0.99841956	0.845642048	0.8496341	0.8669328	0.852295409	0.8556221	0.8602794	
0.942289137	**0.9408917**	**0.96325071**	**0.850432468**	**0.855489**	**0.8657352**	**0.853093812**	**0.8576181**	**0.8656021**	AVG
0.077232821	**0.0802071**	**0.05033964**	**0.007975711**	**0.0137114**	**0.0049432**	**0.009680582**	**0.0123938**	**0.0042861**	STD

Figura 6.7: Resultados para la red ABN Múltiple con Balanceo

Por último, se hicieron pruebas haciendo una pequeña modificación a la unión de las ramas de atención, en estos experimentos simplemente se sumaron los flujos de ambas ramas dentro de la rama de percepción (a diferencia de la concatenación y el balanceo de los experimentos anteriores). Los resultados se muestran en la figura 6.8.

Dataset	DS15K								
Modelo	Multi ABN sumados (ResNet) 25 y 43								
Batch Size	48								
Epochs	50								
LR	0.0005								
Closed Test			Validation			Test			
ABN-Depth	ABN-RGB	Total	ABN-Depth	ABN-RGB	Total	ABN-Depth	ABN-RGB	Total	
0.998835468	0.9991682	0.99925137	0.854956752	0.8729208	0.8769128	0.860944777	0.8642715	0.8669328	
0.999168192	0.9992514	0.99908501	0.857618097	0.8609448	0.8735862	0.844976712	0.8622754	0.8602794	
0.996755947	0.9970887	0.99925137	0.856952761	0.8695941	0.8742515	0.852960745	0.8695941	0.8656021	
0.998669107	0.9989186	0.99925137	0.854956753	0.8695941	0.8682635	0.842315369	0.8616101	0.8662675	
0.998253202	0.9985027	0.99925137	0.851630073	0.8556221	0.8729208	0.851630072	0.8556221	0.8656021	
0.998336383	**0.9985859**	**0.9992181**	**0.855222887**	**0.8657352**	**0.873187**	**0.850565535**	**0.8626747**	**0.8649368**	AVG
0.00094292	**0.0008862**	**7.4399E-05**	**0.002333423**	**0.0071875**	**0.0031419**	**0.007312645**	**0.0050364**	**0.0026613**	STD

Figura 6.8: Resultados para la red ABN Múltiple con Suma.

6.2. Pruebas del Selector con el simulador

Una vez entrenada la red neuronal e integrada con los demás módulos del selector, se procedió a crear una nueva simulación para probar la arquitectura final del Selector. El Selector fue creado de manera que se pudiera cambiar la red neuronal sin afectar a los demás módulos ni a su funcionamiento general, es decir, el selector debe seguir funcionando independientemente del tipo de red neuronal usada (*Vanilla*, *ResNet*, *ABN Simple o ABN Múltiple*). Esto se diseñó así para tener mayor flexibilidad y poder probar con alguna otra arquitectura de red en el futuro.

Para estos experimentos se creó una nueva simulación, muy similar a la creada para la obtención del conjunto de datos. Esta difiere de la anterior en que la estrategia a utilizar por el robot para colocar el objeto no es aleatoria si no proporcionada por el Selector.

En resumen, las características de la simulación para pruebas son las siguientes:

- Utiliza las 4 estrategias descritas en la sección 6.1.
- Se utilizaron los mismos objetos y muebles descritos en la sección 4.1.
- Inicia con una escena virtual creada igual que la simulación para la recolección del conjunto de datos (es decir, ambiente doméstico con el HSR en una posición dada, superficie destino enfrente del HSR a una distancia que puede alcanzar, un número aleatorio de objetos dispersos sobre la superficie destino y el objetivo en la mano del robot, figura 6.9).

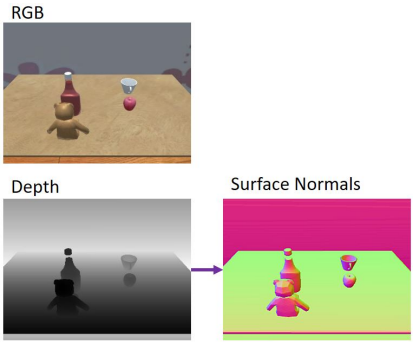
Figura 6.9: Escena virtual inicial

- Cuando la escena está lista se notifica al robot para posicionar su cabeza y enfocar el área destino para obtener las imágenes de profundidad (*Depth*) y RGB de sus cámaras (figura 6.10).

Figura 6.10: El robot posiciona su cabeza para visualizar el área destino

- El robot utiliza las imágenes obtenidas como entradas para el Selector (a partir de este punto es diferente a la simulación anterior).

- El Selector procesa las imágenes y genera una estrategia que presumiblemente tiene la mayor probabilidad de éxito (figura 6.11).



(a) Imágenes RGB y *Depth* (izquierda) y SN (derecha).

```
P R E D I C T I N G
MODEL :  /home/pumas/Angelica/proyecto/MODELS/model/DS15K/v
an.RGB1.model_01062020_1644.h5
DATA TYPE :  R

SUCCESS          FAILURE
_____________________________________
 0.8671           0.1329
 0.5218           0.4782
 1.0000           0.0000
 0.8355           0.1645
3
SELECTED STRATEGY :  3
```

(b) Estrategia seleccionada

Figura 6.11: Selector. Entradas: Imágenes (RGB y Depth) Salida: Estrategia seleccionada

- El robot utiliza la estrategia proporcionada por el Selector y ejecuta los movimientos asociados a ésta (figura 6.12).

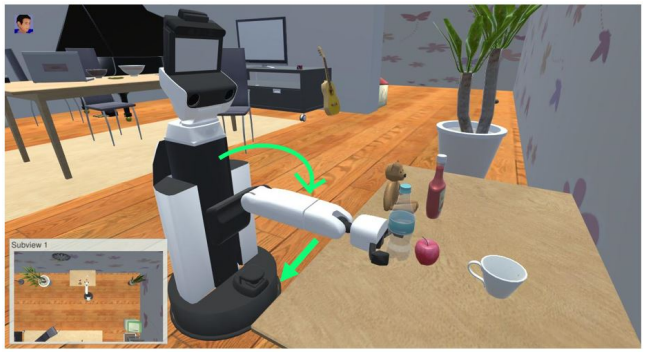


Figura 6.12: El robot ejecuta los movimientos de la estrategia

- El simulador califica la colocación del objetivo de la misma manera que en la simulación de obtención de datos.

Usando esta nueva simulación se probaron los modelos entrenados con mejores resultados. Para todos los tipos de redes (*Vanilla*, *ResNet*, *ABN Simple*, *ABN Múltiple*), se utilizaron los modelos que tuvieron mejor desempeño en la validación y pruebas. Se hicieron 10 simulaciones (5 de validación y 5 de prueba) de 100 escenas cada una y se tomaron los promedios.

Primero se probaron los modelos base *Vanilla* propio usando respectivamente las entradas RGB, *Depth* y RGB-D. Los resultados se muestran en la figura 6.13.

Dataset	DS15K								
Modelo	Vanilla (Propio)								
Batch Size	64								
DEPTH	Epochs	300	RGB			RGB-D			
	LR	0.0001							
Model Validation	Model Test	Promedio	Model Validation	Model Test	Promedio	Model Validation	Model Test	Promedio	
84	90	**87**	84	81	**82.5**	85	84	**84.5**	
79	86	**82.5**	91	85	**88**	78	83	**80.5**	
82	83	**82.5**	87	85	**86**	85	81	**83**	
81	82	**81.5**	86	81	**83.5**	84	81	**82.5**	
84	84	**84**	89	78	**83.5**	83	86	**84.5**	
82	**85**	83.5	**87.4**	**82**	84.7	**83**	**83**	83	AVG
2.121320344	**3.1622777**	2.15058132	**2.701851217**	**3**	2.2527761	**2.915475947**	**2.1213203**	1.6583124	STD

Figura 6.13: Resultados del Simulador con la Red Vanilla propia.

A continuación se probaron los modelos ResNet50 de Keras, también con entradas RGB, *Depth* y RGB-D. Los resultados se muestran en la figura 6.14.

Dataset	DS15K								
Modelo	ResNet50 (Keras)								
Batch Size	64					Batch Size		32	
DEPTH	Epochs	100	RGB			RGB-D			
	LR	0.0001							
Model Validation	Model Test	Promedio	Model Validation	Model Test	Promedio	Model Validation	Model Test	Promedio	
80	79	**79.5**	82	83	**82.5**	60	67	63.5	
73	84	**78.5**	88	89	**88.5**	56	59	57.5	
73	85	**79**	84	90	**87**	57	59	58	
83	79	**81**	79	86	**82.5**	66	68	67	
79	81	**80**	85	76	**80.5**	62	63	62.5	
77.6	**81.6**	79.6	**83.6**	**84.8**	84.2	**60.2**	**63.2**	61.7	AVG
4.449719092	**2.792848**	0.9617692	**3.361547263**	**5.6302753**	3.3837849	**4.024922359**	**4.2661458**	3.9780649	STD

Figura 6.14: Resultados del Simulador con la Red ResNet50 de Keras

Posteriormente se probaron los modelos de ABN Simple, nuevamente con entradas RGB, *Depth* y RGB-D. Los resultados se muestran en la figura 6.15.

Dataset	DS15K								
Modelo	Simple ABN (ResNet)								
Batch Size	48								
DEPTH	Epochs	50	RGB			RGB-D			
	LR	0.0005							
Model Validation	Model Test	Promedio	Model Validation	Model Test	Promedio	Model Validation	Model Test	Promedio	
84	85	**84.5**	87	87	**87**	80	82	81	
75	78	**76.5**	87	86	**86.5**	85	83	84	
75	88	**81.5**	87	93	**90**	89	88	88.5	
86	79	**82.5**	89	86	**87.5**	93	78	85.5	
87	85	**86**	84	82	**83**	87	80	83.5	
81.4	**83**	82.2	**86.8**	**86.8**	86.8	**86.8**	**82.2**	84.5	AVG
5.941380311	**4.3011626**	3.63318042	**1.788854382**	**3.9623226**	2.5149553	**4.816637832**	**3.7682887**	2.7613403	STD

Figura 6.15: Resultados del Simulador con la Red ABN Simple

Finalmente se probó el modelo propuesto ABN Múltiple, en este caso las entradas son 2: RGB y *Depth*. Los resultados se muestran en la figura 6.16.

Dataset	DS15K		
Modelo	Multi ABN (ResNet) 25 y 43		
Batch Size		48	
RGB-D	Epochs	50	
	LR	0.0005	
Model Validation	Model Test	Promedio	
82	89	**85.5**	
90	88	**89**	
89	87	**88**	
88	86	**87**	
89	89	**89**	
87.6	**87.8**	87.7	AVG
3.209361307	**1.3038405**	1.4832397	STD

Figura 6.16: Resultados del Simulador con la Red ABN Múltiple

Para resumir los resultados, se muestran las siguientes tablas donde se muestran los promedios de los entrenamientos y de las simulaciones (figura 6.17).

Dataset	DS15K		
Epochs	**300**		
Batch size	**64**		
LR	**0.0001**		
Modelo	Entradas	Accuracy	Simulador %
Vanilla (propio)	Depth	82.20±1.62	83.5±2.15
	RGB	82.52±0.27	84.7±2.25
	RGB-D	81.72±0.61	83.0±1.65

Dataset	DS15K		
Epochs	**100**		
Batch size	**64**		
LR	**0.0001**		
Modelo	Entradas	Accuracy	Simulador %
ResNet50	Depth	82.19±0.56	79.6±0.96
	RGB	81.98±0.42	84.2±3.38
	RGB-D	81.95±0.48	61.7±3.97

Dataset	DS15K			
Epochs	**50**			
Batch size	**48**			
LR	**0.005**			
Modelo	Entradas	Attention	Accuracy	Simulador %
SimpleABN	Depth	84.49±1.02	85.62±0.41	82.2±3.63
	RGB	85.34±1.82	86.75±0.52	86.8±2.51
	RGB-D	84.28±4.32	86.22±0.24	84.5±2.76

Dataset	DS15K				
Epochs	**50**				
Batch size	**48**				
LR	**0.005**				
Modelo	Entradas	Attention Depth	Attention RGB	Accuracy	Simulador %
MultiABN	RGB-D	84.37±1.49	86.87±0.58	86.86±0.43	87.7±1.48

Figura 6.17: Resumen de resultados

6.3. Pruebas de integración de sistemas

Para probar los sistemas integrados se diseñó un experimento donde el robot real es teleoperado para posicionarse frente a una mesa o mueble donde planea realizar una colocación de objetos autónoma usando el selector. El robot captura imágenes reales del ambiente y los objetos del área destino (usando el Kinect2).

Usando estas imágenes el sistema de detección de objetos reconoce los objetos y sus ubicaciones y transmite la información al ambiente virtual para representar los objetos reales como objetos virtuales (figura 6.18).

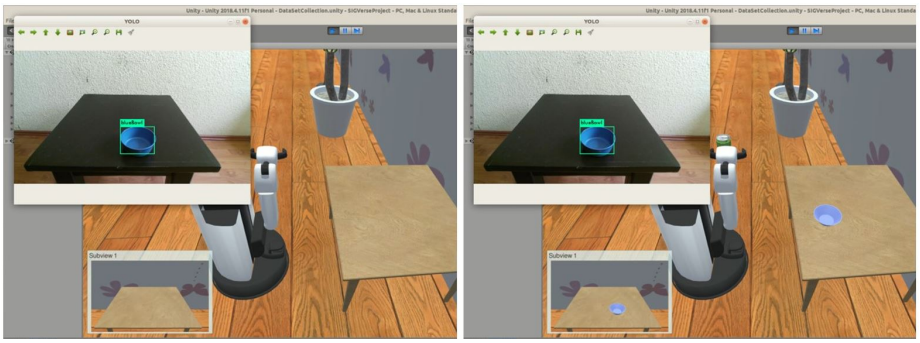


Figura 6.18: El robot envía imágenes reales para reconocer objetos y representarlos en el ambiente virtual

Una vez representados correctamente los objetos, el Selector procede a procesar las nuevas imágenes procedentes del ambiente virtual para hacer la predicción de la estrategia óptima y una vez elegida se manda a ejecutar por el robot (figura 6.19). Debido al limitado acceso al robot real, para estas pruebas se continuó utilizando el simulador para representarlo, cabe mencionar que este simulador es distinto del ambiente virtual del sistema de teleoperación por lo cual simplemente suple temporalmente al robot real. Sin embargo, el paso del simulador al robot real ha sido probado anteriormente y se puede asumir que la ejecución en el robot real es prácticamente la misma que en el simulador.

Figura 6.19: El robot coloca los objetos autónomamente usando la representación virtual de los objetos

Este experimento se repitió, pero ahora agregando otro objeto real para probar que ambos objetos detectados fueran representados en el ambiente virtual, para este nuevo experimento, los dos objetos fueron correctamente detectados y representados. Posteriormente la colocación del objeto teniendo los dos objetos obstáculo también fue exitosa (figura 6.20).

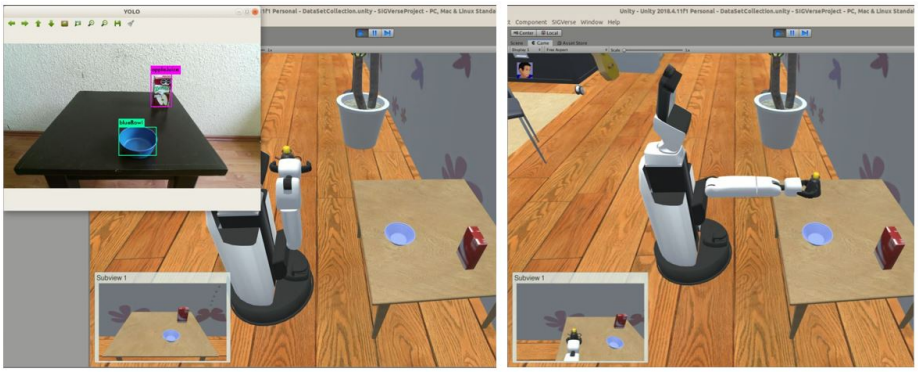


Figura 6.20: Experimento usando dos objetos reales para su representación y colocación

El siguiente experimento fue quitar uno de los objetos ya existentes y a su vez colocar uno nuevo en otra posición para observar la nueva representación y probar la colocación con las nuevas características (figura 6.21). Cabe destacar que la estrategia de colocación predicha y ejecutada fué diferente en algunos de los experimentos derivado de la configuración de los obstáculos en el área destino. Nuevamente el experimento fue exitoso.

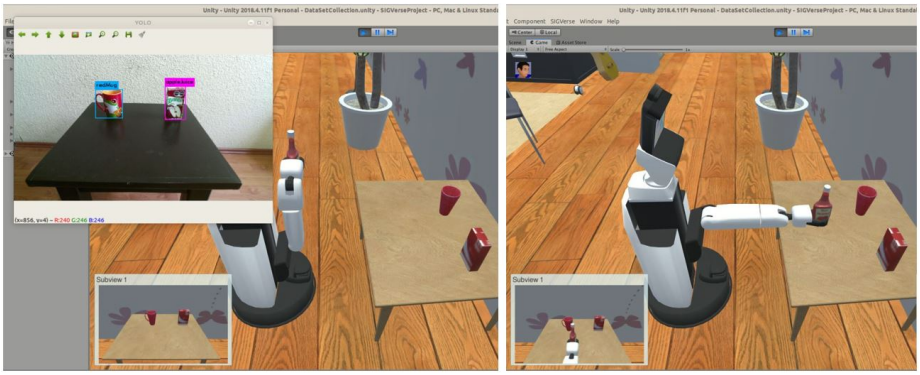


Figura 6.21: Experimento con dos objetos reales en diferente configuración para su representación y colocación

Finalmente se colocaron los 3 objetos para el siguiente experimento y se procedió al igual que en los casos anteriores. Los objetos fueron representados adecuadamente y la colocación también fue exitosa (figura 6.22).

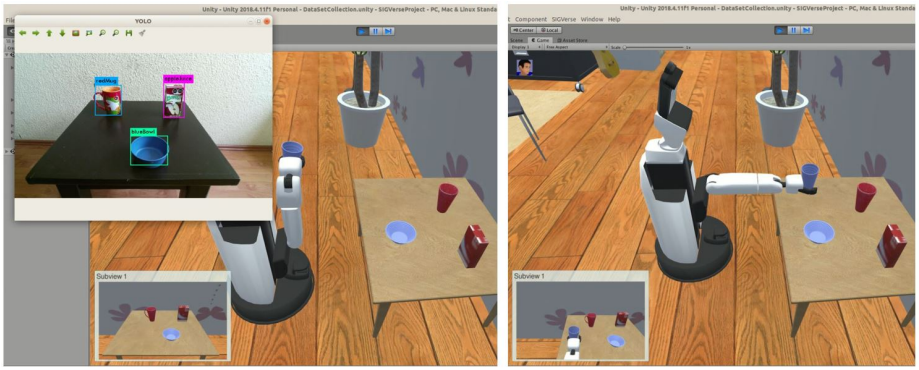


Figura 6.22: Experimento con 3 objetos reales para su representación y colocación.

En resumen, se colocaron varios objetos reales en diferentes posiciones para ser detectados y representados en el ambiente virtual (figura 6.23). Con esto se pretende probar la integración con el sistema de detección y la funcionalidad añadida de representación de objetos virtuales. Posteriormente se usó esta representación para llevar a cabo la colocación autónoma utilizando el Selector de Estrategias incorporado y ejecutar los movimientos de colocación exitosamente.

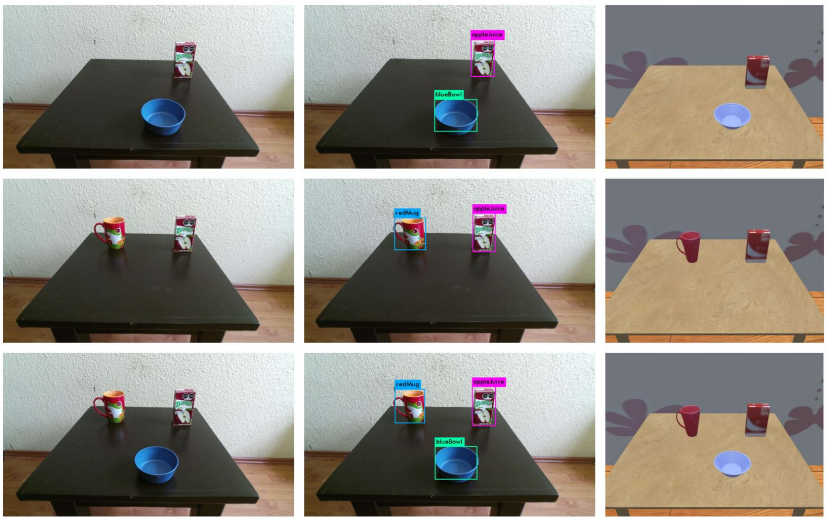


Figura 6.23: Reconocimiento de objetos reales en diferentes configuraciones y sus representaciones en el ambiente virtual

Capítulo 7

Conclusiones

7.1. Conclusiones

Colocar objetos con un robot de servicio es una tarea muy desafiante de la que dependen tareas más complejas. Se deben tener en cuenta una serie de consideraciones al colocar objetos como el entorno, obstáculos, restricciones de movimiento, etc. Uno de los aspectos más importantes a los que se debe prestar especial atención son las colisiones dañinas entre objetos, con el robot o con algún elemento del entorno. Las colisiones pueden ser desde un toque leve hasta un choque violento. Cuando la velocidad de la colisión excede un umbral, se puede considerar una colisión dañina, ya que podría dañar los objetos involucrados en ella.

Otro punto a considerar son los movimientos del robot. Para realizar una colocación exitosa, el robot tiene que manejar una gran cantidad de información para realizar una serie de cálculos y planificación para encontrar un conjunto apropiado de movimientos para colocar. Esos movimientos componen una estrategia. La estrategia seleccionada podría volverse muy compleja o ineficaz si no se ha probado completamente y esto podría resultar en dañar el robot, los objetos o cualquier otra cosa en el ambiente. Al tener un pequeño conjunto de estrategias completamente probadas, la tarea se puede realizar de forma segura y en menos tiempo. El único problema es seleccionar la mejor estrategia para aplicar en cada situación de colocación.

El Selector de Estrategias propuesto utiliza un pequeño conjunto de estrategias simples e imágenes RGB-D para seleccionar la estrategia que tiene la mayor probabilidad de éxito y la menor probabilidad de colisión dañina. La arquitectura del Selector contiene tres componentes principales: el módulo de preprocesamiento, la red neuronal y el discriminador. El módulo de preprocesamiento recorta las imágenes RGB-D de entrada en imágenes más pequeñas que se centran en la región donde una estrategia determinada colocaría el objeto al

ejecutarse. Se aplica un procesamiento de imágenes adicional a estas imágenes recortadas antes de pasarlas a la red neuronal.

En la red neuronal, las imágenes ingresadas se utilizan para predecir la probabilidad de éxito de cada estrategia. Estas predicciones se pasan al último componente, el Discriminador, que selecciona la estrategia que tiene la mayor probabilidad de éxito y la devuelve al robot para ejecutar los movimientos de la estrategia seleccionada. El modelo de red neuronal propuesto tiene dos flujos, uno para el RGB y otro para las imágenes de profundidad. Cada flujo tiene un extractor de características y una rama de atención. Las dos corrientes se fusionan en una rama de percepción mediante un mecanismo de balanceo.

Para los experimentos, se utilizó una simulación de entorno virtual con el fin de recopilar un conjunto de datos lo suficientemente grande para entrenar el modelo (se usaron técnicas de transferencia de conocimiento para el extractor de características y los ajustes finos se hicieron con este conjunto de datos). El tamaño del conjunto de datos recopilado es de aproximadamente 15,000 muestras. La simulación para recopilar el conjunto de datos utilizó una estrategia aleatoria al colocar, de modo que el conjunto de datos tenga un número equilibrado de muestras de cada estrategia. Además, la proporción de muestras positivas-negativas es de aproximadamente el 50 %.

Los modelos propuestos y los modelos de base se entrenaron con este conjunto de datos y los resultados se muestran en la sección anterior. De dichos resultados se pueden resaltar los siguientes hallazgos:

- En general usando alguna de las redes neuronales (base o propuesta) el Selector cumple su función, aún con las más simples, ya que, sin usar una red, es decir aleatoriamente, la tasa de éxito es aproximadamente 50 % y usando una red neuronal base es de alrededor de 75 %, sin embargo, cabe destacar que con la red propuesta el desempeño llega hasta casi 90 %.
- La arquitectura del selector es flexible, se le puede sustituir la red neuronal sin necesidad de mayores cambios.
- En el caso de las redes con entrada simple, en la mayoría de los casos la rama RGB muestra mejor desempeño, esto parece ser consistente con el hecho de que las CNN son óptimas para trabajar con imágenes RGB.
- Los valores para los *closed test* idealmente deben acercarse a 1, ya que son prediciones para datos conocidos, por lo que entre más alto es este valor (cercano a 1) más confiables son los demás resultados.
- La red neuronal mejora un poco con el balanceo a diferencia de la concatenación. Con el balanceo se puede notar que siempre la rama total es más alta que las dos ramas, esto indica que si se tiene un mejor desempeño en las ramas, la total aumenta.

- Aunque las redes ResNet50 son muy complejas y llevaría mucho tiempo entrenarlas desde cero, se utilizó transferencia de conocimiento al utilizar las capas de extracción de características con los pesos preentrenados en ImageNet y solamente se entrenaron las capas de ABN y percepción. Gracias a esto no es necesario tener un conjunto de datos de tamaño del orden del millón, aunque posiblemente con un dataset un poco mayor podrían lograrse mejores resultados.

Los últimos experimentos de las redes neuronales se realizaron implementando el modelo propuesto, es decir, las ABN múltiples (multi ABN) teniendo una rama ABN para *Depth* (ABN-D) y otra para RGB (ABN-RGB), ésta nueva estructura también mostró mejoras con respecto a la ABN simple (sólo una rama ABN). En este caso, ABN-D y ABN-RGB tienen distintas regiones de atención mientras que la primera se enfoca en los bordes y partes altas como puntos de colisión potenciales, la segunda se enfoca en el espacio libre en el área destino, colores y formas. Estas predicciones funcionarán de manera similar en ambientes reales ya que la cámara del simulador está obteniendo información de profundidad de la misma forma que el robot real lo obtendría en un ambiente 3D real. La única diferencia sería la calidad de las imágenes adquiridas por el robot real.

Para validar estos resultados, se construyó una segunda simulación para probar la arquitectura del Selector completa. En esta simulación, el componente de la red neuronal se reemplazó con los modelos de base y el propuesto para obtener el rendimiento de todos los modelos en el Selector. Esta simulación da un vistazo de cómo funcionaría el Selector cuando se implemente en un robot de la vida real. A partir de los resultados de la simulación, se puede observar que la tasa de éxito al colocar tiene una mayor desviación estándar, esto se debe a que al ejecutar la colocación incluso en un entorno simulado, las características externas pueden afectar el resultado final. Sin embargo, los resultados de la red y la simulación arrojan resultados similares, validando la premisa de que el Selector con la red neuronal Multi ABN Balanceada tiene un mejor desempeño que otros modelos.

Adicional a los resultados reportados, se pudo observar que en la simulación de prueba hubo casos en los que las predicciones son correctas, sin embargo, al momento de colocar el robot llegaba a rozar otros objetos lo cual ocasionaba una colisión y el simulador lo evaluaba como falla, sin embargo, al observar la escena, el objeto había sido colocado exitosamente. Posiblemente modificar el umbral de colisiones podría ayudar a mejorar el desempeño. Probablemente en un escenario real, estos casos podrían ser considerados como exitosos por un experto observando la prueba. Además, también se encontraron algunos errores ocasionados por pequeños bugs del simulador como objetos que se atraviesan con otros (errores en los modelos 3D) o quedan flotando misteriosamente en el ambiente (fallos en la simulación de la física), etc.

Por último, se probó el funcionamiento del Selector en colaboración con otros sistemas existentes en el laboratorio de BioRobótica, el sistema de teleoperación y el sistema de detección de objetos. Los tres sistemas (teleoperación, detección y Selector) se integraron adecuadamente, mostrando que forman una sinergia beneficiosa para el desempeño y funcionalidades de los mismos.

Considerando todo lo anterior, se pueden responder las preguntas planteadas en la hipótesis de la siguiente manera:

- *¿Es posible colocar el objeto exitosamente usando un conjunto reducido de estrategias de colocación simples?* y *¿Es posible utilizar recursos simples como las imágenes del entorno obtenidas de las cámaras del robot para predecir el éxito de los movimientos del robot?*

  El Selector de Estrategias de Posicionamiento de Objetos propuesto toma imágenes RGB-D del lugar de colocación usando las cámaras del robot (RGB y Profundidad) para elegir una estrategia de colocación simple para colocar el objeto en el área deseada. Una estrategia de colocación simple es una secuencia de movimientos del robot para colocar un objeto, por ejemplo, moverse unos pasos a la izquierda, estirar el brazo del robot y soltar el objeto. La estrategia de movimiento se selecciona de un pequeño conjunto de estrategias predefinidas y previamente probadas. Con esta información el Selector predice la probabilidad de éxito de cada estrategia y selecciona la óptima para ejecutar, lo cual asegura la mayor probabilidad de éxito para la situación presente, es decir, para las condiciones del entorno observadas por el robot. Por otro lado, los recursos para obtener las imágenes RGB-D son solamente las cámaras del robot (RGB y Profundidad) lo cual hace que el Selector no demande muchos recursos. Entonces, la respuesta a ambas preguntas es SI.

- *¿Se pueden evitar movimientos del robot que resulten dañinos para el mismo robot o los elementos del ambiente usando dichas predicciones?* y *¿Se puede seleccionar una estrategia de colocación simple y probada que prevea resultados óptimos sin tener que realizar cálculos y planeación complejos?*

  A grandes rasgos, el Selector contiene en su interior una red neuronal profunda la cual fue entrenada utilizando un conjunto de datos obtenidos en un ambiente virtual con un simulador. Para el modelado de la red neuronal se diseñó y entrenó un modelo para predecir el éxito al colocar objetos para un conjunto reducido de estrategias tomando en cuenta las colisiones dañinas entre objetos y/o el entorno. Este problema se abordó como un problema de clasificación en el que se infiere la probabilidad de colisiones dañinas dada una escena visual de la superficie destino donde poner un objeto. Una vez obtenidas las predicciones por estrategia, se utilizan para seleccionar la mejor estrategia (dentro del conjunto reducido de estrategias simples) con la cual se prevén resultados óptimos, es decir, aquellos que tienen mayor probabilidad de éxito (o menor de fracaso). Al tener un

conjunto reducido de movimientos del robot totalmente probados y confiables se pretende mejorar el tiempo de ejecución al colocar los objetos y aumentar la probabilidad de éxito, esto elimina la necesidad de realizar cálculos y planeación de cualquier tipo, ya que simplemente se utiliza la estrategia que tiene mayor probabilidad de éxito. Por lo tanto, nuevamente la respuesta a las preguntas es SI.

- *¿Se puede utilizar un ambiente virtual de simulación para resolver el problema y posteriormente utilizar el resultado en un robot y situación reales?*

  Dado que el Selector fue entrenado y probado en un ambiente virtual y un simulador, se puede asumir que su funcionamiento es óptimo en ambientes virtuales, por lo que idealmente sirve para tomar decisiones sólo en dichos ambientes. Para hacer el paso del ambiente virtual a la realidad se propone que el robot utilice el Selector en un ambiente virtual mapeado al real y elija la estrategia de colocación adecuada. Una vez elegida, se debe poder ejecutar en el ambiente real mapeado y el objeto debe ser colocado exitosamente tal como en las simulaciones.

  Como aplicación y caso de prueba para validar que el Selector puede funcionar adecuadamente en situaciones reales con sistemas más complejos para realizar tareas más demandantes, imaginemos que se opera el robot usando un sistema de teleoperación con realidad mixta (detallado en la sección 5.1) para navegar y explorar el ambiente real. El robot puede llegar a una ubicación donde desea colocar un objeto que ya trae en la mano (puede ser algo que otro usuario en el ambiente real remoto le proporcionó) y el operador simplemente le manda la instrucción de "colocar" en la ubicación dada (alguna mesa, mueble o superficie). El robot coloca el objeto automáticamente, planeando dónde puede colocar el objeto sobre el destino sin dañar otros objetos que ya estén en el lugar. Para esto, el robot usa su cámara en el ambiente real para obtener imágenes del lugar destino y los objetos existentes. Haciendo uso de técnicas de reconocimiento de objetos, el robot identifica los objetos y su ubicación para luego representarlos en el ambiente virtual. Dentro del ambiente virtual, el robot utiliza el Selector de Estrategias (entrenado previamente en un ambiente virtual) para predecir la estrategia de colocación que tenga mayor probabilidad de éxito (sin dañar los objetos existentes ni al ambiente). Con esto se espera evitar cruzar o saltar la "Brecha de la realidad" (*reality gap*) para no tener la necesidad de reentrenar al Selector con imágenes de ambientes reales. Adicionalmente, con estos experimentos se pretende probar que se puede extender este tipo de entrenamientos en ambientes virtuales a otras funcionalidades y modelos. Las pruebas realizadas en el simulador y los sistemas acoplados sugieren que el funcionamiento del Selector es similar en el ambiente virtual que en el real. En resumen, la respuesta es que las pruebas sugieren que SI.

En general se puede concluir que el Selector de Estrategias de Colocación cumple con los objetivos planteados y funciona adecuadamente.

El Selector también funciona adecuadamente en colaboración con otros sistemas existentes, por lo que se puede extender su uso a más sistemas o como parte de tareas más complejas como por ejemplo aquellas de las competencias de robots. Con este enfoque se intenta de cierta manera cruzar la brecha de la realidad ya que se puede utilizar entrenamiento hecho en el mundo virtual para resolver problemas del mundo real.

7.2. Contribuciones

Las principales contribuciones de este proyecto son:

- *Utilización de un conjunto reducido de estrategias de colocación simples previamente probadas para evitar calcular trayectorias complicadas y/o controlar movimientos complejos del robot.*

  El uso del Selector evita la planeación, además de ser un proceso menos tardado que otros similares observados en pruebas durante competencias. Por ejemplo, usando MoveIt (*MoveIt Motion Planning Framework*) para planeación de movimientos, éste generalmente tarda mucho en planear y puede dar como respuesta que no es posible hacer movimientos adecuados, en contraste, el simulador tarda menos de 10 segundos en decidir y ejecutar la estrategia, lo cual es mucho más rápido. El Selector es un método simple y efectivo, no necesita demasiadas estrategias de colocación ya que la mayoría de los objetos se pueden colocar en pocas ubicaciones, otros sistemas fallan incluso cuando la colocación es muy sencilla (observado en competencias como RoboCup y TMR).

- *Uso de una nueva metodología para resolver el problema de la colocación de objetos utilizando imágenes RGB-D capturadas por el robot y el conjunto de estrategias de colocación.*

  Las metodologías para colocación de objetos más comúnmente utilizadas, como detección de áreas libres y/o detección de objetos en conjunto con técnicas de planeación de movimientos como cálculo de trayectorias usando cinemática inversa, consumen muchos más recursos computacionales que la propuesta en este trabajo. En el caso de la detección de áreas libres el robot llega a tardar bastante tiempo (incluso varios minutos), lo cual durante las competencias o incluso en una situación de la vida cotidiana es demasiado, además, dado que todo ese tiempo el robot está inmóvil mientras procesa la información, puede llegar a parecer que el sistema está "colgado" o "trabado" (como coloquialmente se le conoce), lo mismo aplica para el cálculo de trayectorias, que si bien suele ser un poco menos tardado que la detección de áreas libres, también llega a consumir una cantidad considerable de recursos. Con la metodología propuesta el tiempo de planeación de movimientos se elimina dado que los movimientos de una estrategia son previamente definidos y probados, con lo que además se minimiza el riesgo de fallos.

- *Modelo de red neuronal profunda de dos streams o flujos (RGB y profundidad) usando ramas de atención y un mecanismo de balanceo con atención temporal para predecir el éxito al colocar.*

  La mayoría de los modelos utilizados en estas tareas suelen usar solamente un flujo, generalmente el RGB, y procesar las imágenes con métodos convencionales. En dichos casos es muy común que se tengan problemas o errores ocasionados por las condiciones de iluminación, fondos no contrastantes con los objetos, superficies brillosas o con texturas complejas, etc. En dichas situaciones las imágenes RGB pueden arrojar resultados deficientes, por lo que se podría pensar que los datos de profundidad se desempeñarían mucho mejor en estos casos. Sin embargo, usar sólo la información de profundidad (*Depth*) sería útil excepto cuando hay objetos transparentes, con huecos o cóncavos ya que la cámara de profundidad no obtiene información correcta sobre estos objetos, es por esto que la tarea de colocar es difícil de predecir. Usando el enfoque propuesto se han obtenido mejoras importantes ya que utiliza los resultados obtenidos usando ambos enfoques y balanceando el resultado para obtener el mejor resultado posible dadas las características del ambiente observado en cada situación particular.

- *Creación de un conjunto de datos (dataset) etiquetado en ambientes virtuales con objetos de la vida cotidiana en un ambiente hogareño.*

  Los conjuntos de datos o datasets son muy apreciados en la comunidad dado que no es fácil ni económico generar dichos datos, la mayoría de los *datset* de este tipo requieren de mucho tiempo y esfuerzo para ser recolectados, ya que se requiere recrear las condiciones una y otra vez con variaciones manualmente y ejecutar repetidas veces todo el proceso de colocación. Adicionalmente, se requieren recursos humanos expertos para el etiquetado de los mismos, sin embargo, no se puede asegurar un criterio unificado ya que éste suele ser subjetivo. Por todas estas situaciones, el crear un conjunto de datos resulta muy útil para la comunidad y para otros proyectos desarrollados dentro de nuestro laboratorio. Además, usando la simulación creada para la recolección se puede agrandar el *dataset* o bien personalizar sus elementos como objetos, muebles, condiciones de luz, criterios de etiquetado, etc.

- *Integración de la metodología propuesta con sistemas de teleoperación en ambientes de realidad mixta y reconocimiento de objetos para mejorar el desempeño de los mismos.*

  El Selector propuesto se integró exitosamente con el sistema de teleoperación en ambientes de realidad mixta y el sistema de reconocimiento de objetos, ambos desarrollados en nuestro laboratorio. Con dicha integración se dotó al sistema de teleoperación con la capacidad de representar en el ambiente virtual objetos observados en el ambiente real usando el detector de objetos. También se le adicionó la nueva funcionalidad de colocación de objetos de manera autónoma usando el Selector. Esta funcionalidad es de

gran utilidad para un sistema teleoperado, ya que la tarea de colocación requiere de movimientos finos y precisos que son difíciles de controlar remotamente debido al problema de la latencia y el mapeo de movimientos del *joystick* al robot, los cuales al ejecutarse pueden llegar a ser burdos e imprecisos. Por otro lado, los experimentos mostraron que los objetos virtuales generados a partir de los objetos reales aportan al ambiente virtual mayor realismo y a su vez esta representación ayuda al Selector a realizar su tarea usando las imágenes del ambiente virtual sin tener que usar imágenes reales, por lo que el Selector se comporta exactamente igual que en los experimentos con el simulador y se obtienen los mismos resultados exitosos.

Derivados de este trabajo se publicaron hasta el momento algunos artículos, el primero titulado *"Teleoperated Service Robot with an Immersive Mixed Reality Interface"* [120] en cual se describe el sistema de teleoperación también desarrollado en el laboratorio de BioRobótica y que fue integrado con el Selector y el sistema de detección de objetos para mejorar sus funcionalidades. En este artículo se describen los detalles de su implementación, sus beneficios, utilidad y aplicaciones entre otros.

El segundo titulado *"Robotics, AI and Machine Vision"* [121] en el cual se detallan proyectos realizados en el laboratorio de BioRobótica así como también se describen los métodos y las tecnologías utilizados y las aportaciones al campo de la Robótica, la Inteligencia Artificial y la Visión Computacional.

7.3. Trabajo futuro

Una de las posibles aplicaciones del Selector en un futuro cercano sería en las nacientes competencias de robots en línea o con uso de simuladores. A pesar de que competencias como la RoboCup en su versión remota de 2021 utilizó otro simulador (gazebo) diferente a SIGVerse, el Selector podría utilizarse sin mayores modificaciones en otros simuladores.

Como trabajo futuro se podría considerar probar el Selector en conjunto con tareas más complejas, por ejemplo, las de buscar y traer objetos realizadas en las competencias internacionales como RoboCup. Otro proyecto a futuro sería migrar el Selector a otro robot de servicio distinto al HSR, por ejemplo, para el robot Justina (robot desarrollado totalmente en el laboratorio de BioRobótica) y adaptar las estrategias probadas a movimientos que este robot pueda ejecutar.

En la parte interna del Selector, podrían hacerse algunas modificaciones como sustituir la red neuronal CNN por otro tipo de red y probar los efectos que dicha red generaría. Otra idea sería utilizar la imagen completa (sin dividir

en parches) y modificar la estructura de la red, esto podría hacerse también utilizando la imagen final (después de colocar) para poder entrenar una red generativa (posiblemente adversarial o GAN) para generar datos artificiales en la que dada una imagen de entrada de la escena inicial generen la imagen final para cada estrategia y con esto predecir la probabilidad de éxito de cada una para entonces utilizar un optimizador y elegir la mejor.

La simulación por su parte podría modificarse para agregar objetos desconocidos (objetivos y obstáculos) y destinos desconocidos (mobiliario diferente). El número de obstáculos también se incrementaría para generar escenarios más desafiantes. Con estas modificaciones se probaría cómo se generaliza el Selector en circunstancias nunca antes vistas.

En resumen, el Selector tiene variadas aplicaciones además de poder usarse como base para desarrollos más complejos o para formar parte de sistemas más ambiciosos donde la colocación de objetos usando robots de servicio sea necesaria.

Apéndice A

Robot de Servicio HSR

El *Human Support Robot* (HSR) es un robot manipulador móvil compacto creado por *Toyota Motor Corporation* en 2012 que tiene ambas funciones, trabajo físico y comunicación (figura A.1). HSR se ha desarrollado con el objetivo de realizar tareas como manipular muebles (por ejemplo, abrir/cerrar cajones, usar microondas, etc.), recoger y transportar objetos de la vida diaria y ordenar las habitaciones. Tiene como objetivo apoyar a las personas que tienen mayores necesidades para la vida diaria.

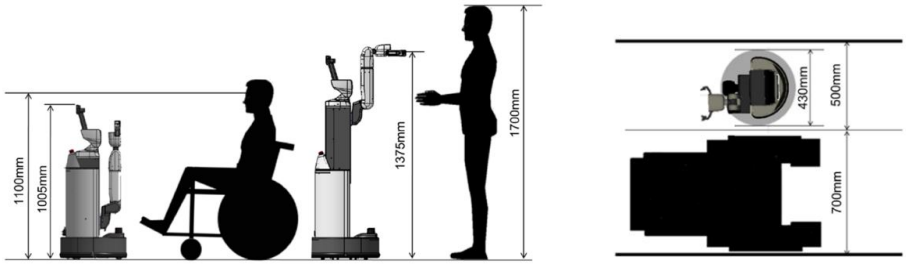
Figura A.1: Human Support Robot de Toyota (imagen tomada de [122])

Operable por comandos de voz o por aplicaciones, el HSR tiene un cuerpo cilíndrico altamente maniobrable, compacto y liviano con un brazo plegable que le permite levantar objetos del piso, succionar objetos delgados, recuperar objetos de lugares altos, abrir cortinas, y realizar otras tareas del hogar.

El objetivo de Toyota es establecer la comunidad de desarrolladores de HSR como una red sólida entre varios institutos de investigación que comparten la misma plataforma de robot para acelerar la investigación y el desarrollo de un manipulador móvil doméstico.

El HSR fue adoptado como plataforma estándar para RoboCup @Home de los seis candidatos que aprobaron la selección de documentos bajo la revisión del Comité Internacional de RoboCup en 2016. El HSR se ha utilizado en la Liga de plataforma doméstica estándar (DSPL) desde RoboCup 2017 Nagoya. Además, se ha adoptado como plataforma estándar para las competencias de robots de servicio de la *World Robot Summit* (WRS) en 2018 y 2021. Dados estos hechos, parece que HSR ha ganado popularidad en las competiciones de robots. El HSR ha sido proporcionado a 44 universidades y empresas en 12 países a través de ofertas públicas (al 30 de noviembre de 2018), y la investigación y el desarrollo están en curso en cada proyecto [123].

En cuanto al diseño, el robot en su conjunto tiene 8 grados de libertad (DoF, del inglés *degrees of freedom*) para manipulación, compuesto por 3 DoF de la base móvil, 4 DoF del brazo y 1 DoF del levantamiento del torso. Por lo tanto, es posible generar movimientos flexibles moviendo la base móvil y el brazo juntos. Cuenta también con un novedoso método de control de movimiento de todo el cuerpo que aprovecha mejor la configuración de este robot para la coordinación entre el transporte, movimiento y la operación de agarre. La carga útil máxima de peso en postura arbitraria es de 1.2 kg para poder tomar 43 clases de objetos del piso al escritorio (0 - 725 mm) en tres direcciones (superior, lateral, frontal) de la mano. La altura objetivo es 1.35m considerando la accesibilidad a los muebles a la altura del hombro (1331 mm), a la que normalmente pueden acceder personas de pie. La máxima velocidad es 0.8 km/h considerando pruebas de campo con personas mayores o con discapacidades, para darles un sentido de seguridad. Los detalles de las dimensiones del HSR se muestran en la figura A.2 y sus especificaciones básicas en la figura A.3. Más detalles se pueden consultar en [123]



(a) Vista Lateral (b) Vista superior

Figura A.2: Dimensiones del HSR (imágenes tomadas de [123])

Height	φ430 × 1005 (~ 1350)mm
Weight	37 kg
Arm length	600 mm
Shoulder height	340 ~ 1030 mm
Grasped object	~ 1.2 kg weight
	~130 mm width
Maximum velocity	0.8 km/h
Mobility performance	~ 5 mm difference in level
	~ 5 deg slope

Figura A.3: Especificaciones básicas HSR (imágenes tomadas de [123])

En la figura A.4 se muestran los sensores y equipamiento con que cuenta el HSR. Se han colocado varios sensores para facilitar el uso del robot como plataforma de investigación. Las medidas de seguridad incluyen entre otros: la reducción del riesgo de caídas a través de la compensación por gravedad del brazo y el mecanismo de autobloqueo de la mano, la reducción del peligro de contacto al reducir el empuje y la función de parada de la cinta magnética para reducir el riesgo caer de escaleras y escalones, etc.

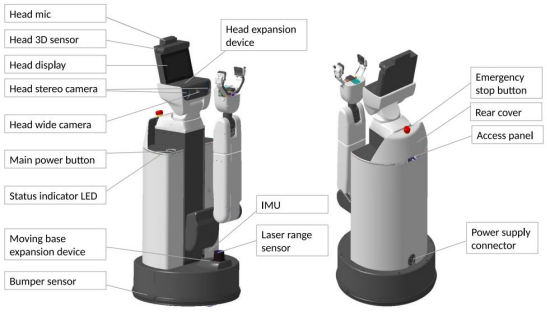


Figura A.4: Sensores y equipamiento del HSR (imágenes tomadas de [123])

La arquitectura de software de HSR se basa en ROS (*Robot Operating System*) [124]. El sistema de software se divide principalmente en cuatro subsistemas: el subsistema de control del dispositivo, que está estructurado por un grupo de servoamplificadores; el subsistema de control de movimiento, que opera en tiempo real en la computadora del robot; el subsistema funcional de nivel superior, que consta de grupos de nodos ROS, también opera en la computadora robot; y el subsistema de interfaz de usuario. Si bien la adopción de ROS ha contribuido en gran medida al desarrollo de software de robots, ha existido la preocupación de que haya aumentado el costo de aprendizaje para los usuarios novatos, con conceptos avanzados como el modelo de comunicación de *publicación-suscripción* o la programación de *callbacks*. Para reducir el costo de aprendizaje, HSR proporciona una interfaz de programación de alto nivel escrita en *Python*. Con lo que se abstrae la interfaz ROS del subsistema funcional y permite una programación de robots altamente abstracta en forma de operaciones imperativas en objetos.

Apéndice B

Simulador SIGVerse

SIGVerse [125] es una plataforma de simulación para robótica que cuenta con características específicamente diseñadas para cubrir las necesidades en el desarrollo de sistemas de interacción *humano-robot* (HRI, del inglés *Human-Robot Interaction*), por lo que ofrece la mayoría de los elementos fundamentales para el modelado y simulación de robots al integrar realidad virtual y tecnologías en la nube. En las simulaciones SIGVerse cuenta con cálculos realistas de física y dinámica para poder simular las propiedades de los objetos, interacciones como movimientos, colisiones, tomar y manipular los objetos, etc. Dado que estas características son de vital importancia en la simulación de interacción *humano-robot*.

Por otro lado, también se incorporan cuestiones de percepción para los agentes (sensores de visión, de audio y táctil, por ejemplo) para hacer más realistas las simulaciones. Además, cuenta con un modelado de robots y agentes humanos (avatares) con los cuales además se puede interaccionar de manera verbal o bien por medio de gestos y movimientos, así como también se pueden enviar mensajes escritos por medio de su interfaz gráfica de usuario (GUI, del inglés *Graphic User Interface*).

SIGVerse es una plataforma de software abierta con alta reutilización para recopilar y aprovechar los datos de experiencias de interacción multimodal que se recopilaron en entornos de la vida diaria y que requieren una interacción social incorporada. Aborda cuatro desafíos importantes: reducción de costos de la construcción del entorno experimental, provisión de las mismas condiciones experimentales a los participantes, reproducción de experiencias pasadas y desarrollo de un sistema base para la interfaz natural teleoperación *avatar/robot*.

B.1. Arquitectura

La arquitectura de SIGVeres es *cliente-servidor* (figura B.1). El cliente trabaja sobre Windows y utiliza Unity para simular el ambiente con gráficos y física de gran calidad lo que lo hace fácil personalizar y editar. El servidor trabaja sobre Linux y opera la conexión con ROS con lo que se permite la operación compleja de robots y su interacción.

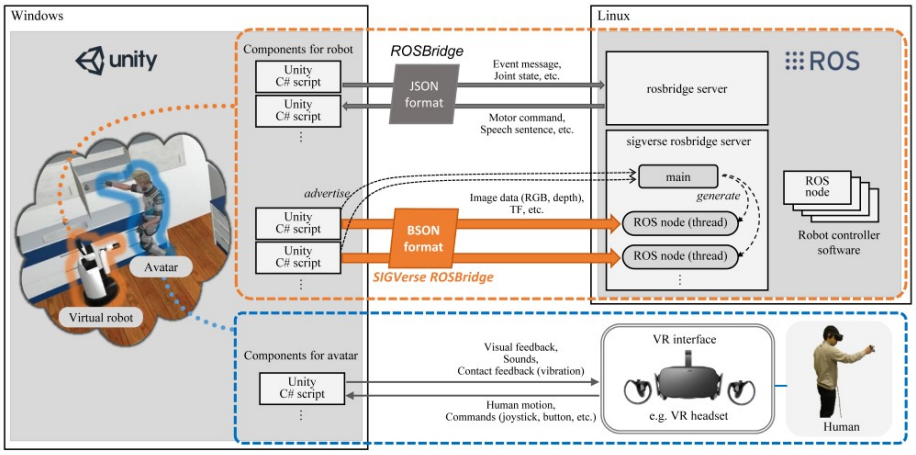


Figura B.1: Arquitectura General de SIGVerse (imagen tomada de [30])

El cliente Unity (figura B.2) contiene las partes principales que hacen fuerte al sistema SIGVerse. Unity permite que varios dispositivos interactivos como visores de realidad virtual como los HTC, Oculus Riff, etc, puedan interactuar con los ambientes virtuales, lo que ayuda a reflejar el comportamiento humano del mundo real en el mundo virtual. Un ser humano real (sujeto de prueba) puede iniciar sesión en el mundo virtual a través de dispositivos de realidad virtual generales para interactuar con el robot virtual en el mundo virtual.

Oculus Rift está disponible como dispositivo de realidad virtual predeterminado cuando un humano inicia sesión en un avatar. El usuario usa el HMD en la cabeza y controla el avatar en realidad virtual agarrando los controladores manuales, llamados Oculus Touch, con ambas manos. La posición y la postura del HMD y los controladores manuales se miden en tiempo real y se reflejan en la cabeza y las manos del avatar.

El servidor basado en ROS hace posible operar los robots con ROS, provee al SIGVerse una gran flexibilidad y permite múltiples interacciones *cliente-servidor*. El software para el control de robots virtuales se puede reutilizar en robots reales sin modificaciones y viceversa.

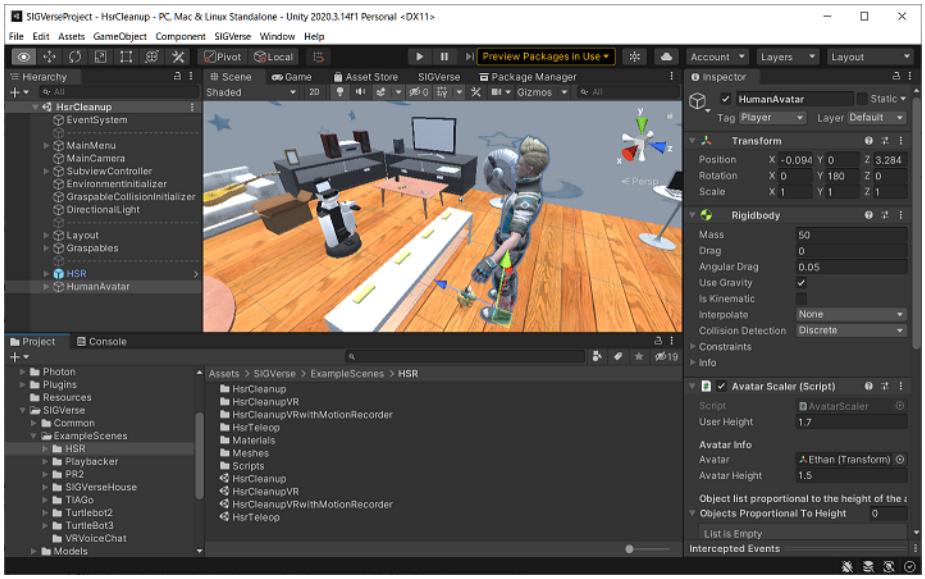


Figura B.2: Cliente en Unity (imagen tomada de [30])

El factor más importante para realizar la integración de ROS y Unity es el protocolo de comunicación entre ellos, por lo que el sistema tiene un mecanismo de puente entre ROS y Unity usando ROSBridge y SIGVerseROSBridge.

El servidor usa dos componentes principales que permiten al cliente comunicarse con ROS:

- *SIGVerseROSBridge* usa BSON para los mensajes y crea nodos ROS de acuerdo con especificaciones estáticas. Este módulo usa BSON para manejar archivos de mensajes de gran tamaño como las imágenes.
- *ROSBridge* utiliza JSON para los mensajes y provee una interfaz de mensajes dinámica. Este módulo se usa para manejar mensajes de tamaño pequeño.

JSON es un formato de intercambio de datos basado en texto que representa pares de palabras clave y valores. Por otro lado, BSON es una serialización codificada en binario con un formato similar a JSON. Aunque el protocolo ROSBridge garantiza el envío y la recepción de mensajes ROS, su rendimiento en la transferencia de datos JSON grandes, como imágenes, es insuficiente y no puede satisfacer la retroalimentación del sensor en tiempo real. Por lo tanto, se implementa un servidor específico (*sigverse_rosbridge_server*) para comunicar grandes volúmenes de datos. Para acelerar la comunicación, se emplea el formato BSON en lugar de JSON.

Cuando los scripts de Unity anuncian mensajes ROS, el hilo principal de *sigverse_rosbridge_server* genera un hilo nuevo para cada tópico como un nodo ROS. Cada hilo recibe mensajes ROS de los scripts de Unity y los publica en los nodos ROS del controlador del robot como mensajes de tópico de ROS (figura B.3).

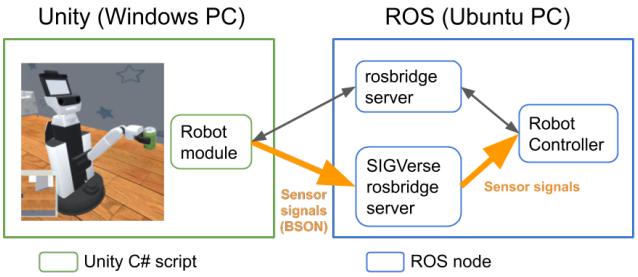


Figura B.3: Operación del robot en ROS (imagen tomada de [30])

Adicionalmente, SIGVerse puede generar los datos binarios sin comprimir de datos de imagen (es decir, datos sin procesar con la estructura de memoria exacta como dispositivos de cámara reales instalados en robots reales) a altas velocidades utilizando las funcionalidades de Unity.

B.2. Demos

SIGVerse cuenta con algunos "demos" para comenzar a usar y probar la comunicación entre el cliente y el servidor, así como para probar algunas de sus características más importantes. En versiones anteriores del sistema se contaba con el demo *"Follower"* (figura B.4), donde se tiene un robot y un avatar (humano) y el robot debe seguir al avatar a donde quiera que se dirija.

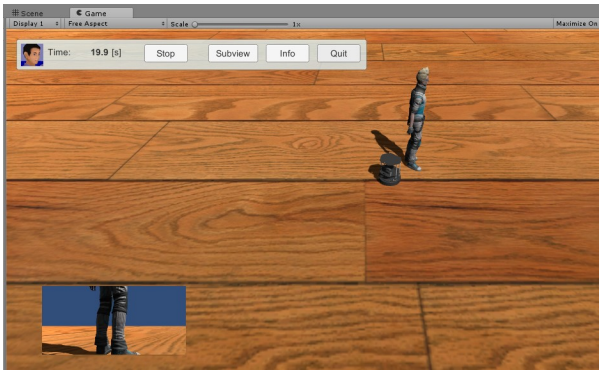


Figura B.4: Demo *"Follower"* en Unity

En el servidor (con ROS) se ve la información del cliente conectado y los datos que enviaría un robot real equivalente al simulado (figura B.5).

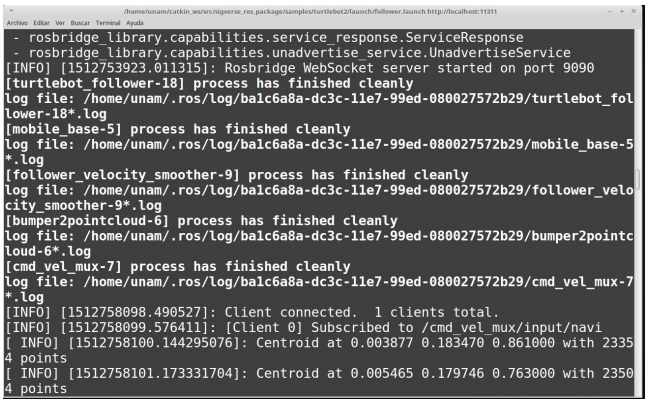


Figura B.5: Demo*"Follower"* en ROS

Además, se puede ver lo que la cámara del robot virtual está visualizando (figura B.6).

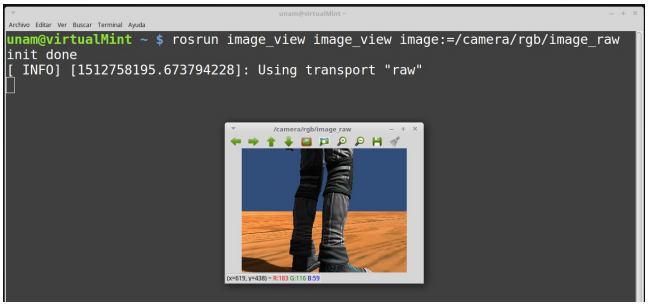
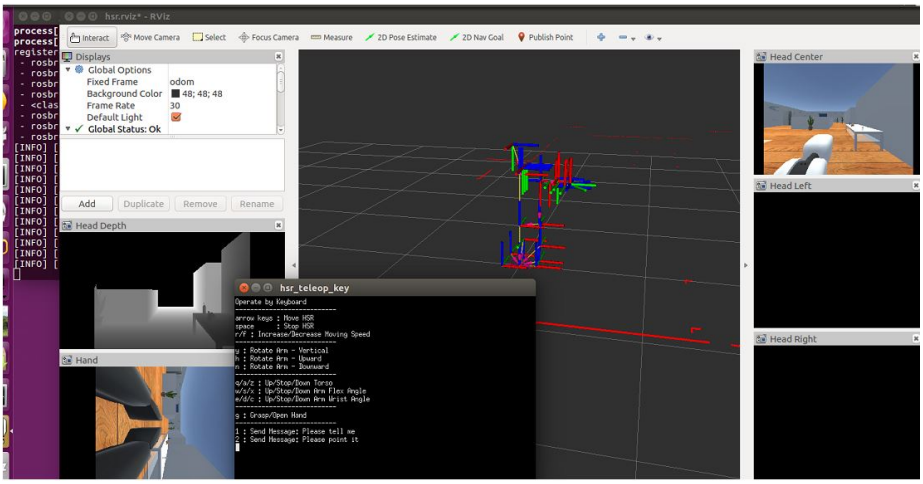


Figura B.6: Imagen de la cámara del robot virtual

En versiones más recientes de SIGVerse, se añadieron más demos para mostrar y probar las nuevas funcionalidades que se han ido incorporando al sistema, algunas incluso hacen uso de los HMD y/o datos en la nube (figura B.7). Cabe mencionar que estos demos y en general el proyecto sigue en constante actualización a través de GitHub [126].



(a) Demo ejecutándose en ROS

(b) Demo ejecutándose en Unity

Figura B.7: Demo de HSR operado con teclado (imágenes tomadas de [30])

Referencias

[1] International Federation of Robotics. https://ifr.org/service-robots/.

[2] L. Sucar and Y. Hernandez, *Robótica de Servicio*. Academia Mexicana de Computación, Octubre 2019.

[3] R. Aracil, C. Balaguer, and M. Armada, "Robots de servicio," *Revista Iberoamericana de Automática e Informática Industrial RIAI*, vol. 5, no. 2, pp. 6–13, 2008, https://doi.org/10.1016/S1697-7912(08)70140-7.

[4] S. Park, "Multifaceted trust in tourism service robots," *Annals of Tourism Research*, vol. 81, p. 102888, 2020, https://doi.org/10.1016/j.annals.2020.102888.

[5] S. Pieska, M. Luimula, J. Jauhiainen, and V. Spiz, "Social service robots in wellness and restaurant applications," *Journal of Communication and Computer*, vol. 10, no. 1, pp. 116–123, 2013.

[6] A. Billard, "Robota: Clever toy and educational tool," *Robotics and Autonomous Systems*, vol. 42, no. 3, pp. 259–269, 2003, https://doi.org/10.1016/S0921-8890(02)00380-9.

[7] V. N. Lu, J. Wirtz, W. H. Kunz, S. Paluch, T. Gruber, A. Martins, and P. G. Patterson, "Service robots, customers and service employees: what can we learn from the academic literature and where are the gaps?" *Journal of Service Theory and Practice*, 2020, https://doi.org/10.1108/JSTP-04-2019-0088.

[8] J. Forlizzi and C. DiSalvo, "Service robots in the domestic environment: A study of the roomba vacuum in the home," in *Proceedings of the 1st ACM SIGCHI/SIGART Conference on Human-Robot Interaction*. New York, NY, USA: Association for Computing Machinery, 2006, pp. 258–265, https://doi.org/10.1145/1121241.1121286.

[9] C. Torras, "Service robots for citizens of the future," *European Review*, vol. 24, no. 1, pp. 17–30, 2016, https://doi.org/10.1017/S1062798715000393.

[10] M. Sprenger and T. Mettler, "Service robots," *Business & Information Systems Engineering*, vol. 57, no. 4, pp. 271–274, 2015, https://doi.org/10.1007/s12599-015-0389-x.

[11] K. Severinson-Eklundh, A. Green, and H. Hüttenrauch, "Social and collaborative aspects of interaction with a service robot," *Robotics and Autonomous Systems*, vol. 42, no. 3, pp. 223–234, 2003, https://doi.org/10.1016/S0921-8890(02)00377-9.

[12] A. Bauer, D. Wollherr, and M. Buss, "Human–robot collaboration: a survey," *International Journal of Humanoid Robotics*, vol. 5, no. 01, pp. 47–66, 2008, https://doi.org/10.1142/S0219843608001303.

[13] T. Fong, I. Nourbakhsh, and K. Dautenhahn, "A survey of socially interactive robots: Concepts, design and applications," Carnegie Mellon University, Tech. Rep., 2002. [Online]. Available: http://hdl.handle.net/2299/3821

[14] Y. Jiang, M. Lim, C. Zheng, and A. Saxena, "Learning to place new objects in a scene," *The International Journal of Robotics Research*, vol. 31, no. 9, pp. 1021–1043, 2012, https://doi.org/10.1177/0278364912438781.

[15] M. J. Schuster, J. Okerman, H. Nguyen, J. M. Rehg, and C. C. Kemp, "Perceiving clutter and surfaces for object placement in indoor environments," in *2010 10th IEEE-RAS International Conference on Humanoid Robots*, 2010, pp. 152–159, https://doi.org/10.1109/ICHR.2010.5686328.

[16] Robocup 2019 rulebook. https://athome.robocup.org/wp-content/uploads/2019_rulebook.pdf.

[17] A. Holladay, J. Barry, L. P. Kaelbling, and T. Lozano-Pérez, "Object placement as inverse motion planning," in *2013 IEEE International Conference on Robotics and Automation*, 2013, pp. 3715–3721, https://doi.org/10.1109/ICRA.2013.6631099.

[18] Toyota Shifts Home Helper Robot R&D into High Gear with New Developer Community and Upgraded Prototype. https://newsroom.toyota.co.jp/en/detail/8709541.

[19] Robocup. https://www.robocup.org/.

[20] H. Kitano, M. Asada, Y. Kuniyoshi, I. Noda, and E. Osawa, "Robocup: The robot world cup initiative," in *Proceedings of the first international conference on Autonomous agents*, 1997, pp. 340–347, https://doi.org/10.1145/267658.267738.

[21] G. Steinbauer and A. Ferrein, "20 years of RoboCup," *KI - Künstliche Intelligenz*, vol. 30, pp. 221–224, 2016, https://doi.org/10.1007/s13218-016-0442-z.

[22] Torneo Mexicano de Robótica TMR 2021. https://www.femexrobotica.org/tmr2021/.

[23] RoboCup 2021. https://2021.robocup.org/.

[24] F. Kanehiro, K. Fujiwara, S. Kajita, K. Yokoi, K. Kaneko, H. Hirukawa, Y. Nakamura, and K. Yamane, "Open architecture humanoid robotics platform," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 1, 2002, pp. 24–30, http://doi.org/10.1109/ROBOT.2002.1013334.

[25] O. Michel, "Cyberbotics Ltd. Webots: Professional Mobile Robot Simulation," *International Journal of Advanced Robotic Systems*, vol. 1, no. 1, p. 5, 2004, https://doi.org/10.5772/5618.

[26] B. Gerkey, R. T. Vaughan, and A. Howard, "The player/stage project: Tools for multi-robot and distributed sensor systems," in *Proceedings of the 11th International Conference on Advanced Robotics (ICAR 2003)*, vol. 1, 2003, pp. 317–323.

[27] P. E. Johnson, "Agent-Based Modeling: What I Learned From the Artificial Stock Market," *Social Science Computer Review*, vol. 20, no. 2, pp. 174–186, 2002, https://doi.org/10.1177/089443930202000207.

[28] T. Inamura, T. Shibata, H. Sena, T. Hashimoto, N. Kawai, T. Miyashita, Y. Sakurai, M. Shimizu, M. Otake, K. Hosoda, S. Umeda, K. Inui, and Y. Yoshikawa, "Simulator platform that enables social interaction simulation - SIGVerse: SocioIntelliGenesis simulator," in *2010 IEEE/SICE International Symposium on System Integration*, 2010, pp. 212–217, https://doi.org/10.1109/SII.2010.5708327.

[29] A. Koubâa *et al.*, *Robot Operating System (ROS)*. Springer, 2019, vol. 1.

[30] SIGVerse. http://www.sigverse.org/wiki/en/.

[31] World Robot Summit. http://worldrobotsummit.org/en/.

[32] G. Niemeyer, C. Preusche, S. Stramigioli, and D. Lee, "Telerobotics," in *Springer Handbook of Robotics*, B. Siciliano and O. Khatib, Eds. Cham: Springer International Publishing, 2016, pp. 1085–1108, https://doi.org/10.1007/978-3-319-32552-1_43.

[33] A. Toet, I. A. Kuling, B. N. Krom, and J. B. F. van Erp, "Toward enhanced teleoperation through embodiment," *Frontiers in Robotics and AI*, vol. 7, p. 14, 2020, https://doi.org/10.3389/frobt.2020.00014.

[34] L. Almeida, P. Menezes, and J. Dias, "Improving robot teleoperation experience via immersive interfaces," in *2017 4th Experiment@International Conference (exp.at'17)*, 2017, pp. 87–92, https://doi.org/10.1109/EXPAT.2017.7984414.

[35] J. Chen, M. Glover, C. Li, and C. Yang, "Development of a user experience enhanced teleoperation approach," in *2016 International Conference on Advanced Robotics and Mechatronics (ICARM)*, 2016, pp. 171–177, https://doi.org/10.1109/ICARM.2016.7606914.

[36] A. Schäfer, G. Reis, and D. Stricker, "Investigating the sense of presence between handcrafted and panorama based virtual environments," in *Mensch Und Computer 2021*, ser. MuC '21. New York, NY, USA: Association for Computing Machinery, 2021, pp. 402–405, https://doi.org/10.1145/3473856.3474024.

[37] S. Lichiardopol, *A survey on teleoperation*, ser. DCT rapporten. Technische Universiteit Eindhoven, 2007, DCT 2007.155. [Online]. Available: https://research.tue.nl/en/publications/a-survey-on-teleoperation

[38] P. Stotko, S. Krumpen, M. Schwarz, C. Lenz, S. Behnke, R. Klein, and M. Weinmann, "A VR system for immersive teleoperation and live exploration with a mobile robot," in *2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, Nov 2019, pp. 3630–3637, http://doi.org/10.1109/IROS40897.2019.8968598.

[39] J. Gradecki, *The Virtual Reality Programmer's Kit*. Wiley, 1994.

[40] R. Hetrick, N. Amerson, B. Kim, E. Rosen, E. J. d. Visser, and E. Phillips, "Comparing virtual reality interfaces for the teleoperation of robots," in *2020 Systems and Information Engineering Design Symposium (SIEDS)*, 2020, pp. 1–7, http://doi.org/10.1109/SIEDS49339.2020.9106630.

[41] D. Whitney, E. Rosen, D. Ullman, E. Phillips, and S. Tellex, "ROS Reality: A Virtual Reality Framework Using Consumer-Grade Hardware for ROS-Enabled Robots," in *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2018, pp. 1–9, http://doi.org/10.1109/IROS.2018.8593513.

[42] J. Lee, M. Kim, and J. Kim, "A Study on Immersion and VR Sickness in Walking Interaction for Immersive Virtual Reality Applications," *Symmetry*, vol. 9, no. 5, 2017, http://doi.org/10.3390/sym9050078.

[43] P. M. Kebria, H. Abdi, M. M. Dalvand, A. Khosravi, and S. Nahavandi, "Control methods for internet-based teleoperation systems: A review," *IEEE Transactions on Human-Machine Systems*, vol. 49, no. 1, pp. 32–46, 2019, http://doi.org/10.1109/THMS.2018.2878815.

[44] P. M. Kebria, A. Khosravi, S. Nahavandi, P. Shi, and R. Alizadehsani, "Robust adaptive control scheme for teleoperation systems with delay and uncertainties," *IEEE Transactions on Cybernetics*, vol. 50, no. 7, pp. 3243–3253, 2020, http://doi.org/10.1109/TCYB.2019.2891656.

[45] H. Rogers, A. Khasawneh, J. Bertrand, and K. C. Madathil, "An investigation of the effect of latency on the operator's trust and performance for manual multi-robot teleoperated tasks," *Proceedings of the Human Factors and Ergonomics Society Annual Meeting*, vol. 61, no. 1, pp. 390–394, 2017, https://doi.org/10.1177/1541931213601579.

[46] T. Sheridan and W. Ferrell, "Remote manipulative control with transmission delay," *IEEE Transactions on Human Factors in Electronics*, vol. HFE-4, no. 1, pp. 25–29, 1963, https://doi.org/10.1109/THFE.1963.231283.

[47] J. Corde Lane, C. Carignan, B. Sullivan, D. Akin, T. Hunt, and R. Cohen, "Effects of time delay on telerobotic control of neutral buoyancy vehicles," in *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, vol. 3, 2002, pp. 2874–2879, https://doi.org/10.1109/ROBOT.2002.1013668.

[48] D. Ruelas, "Agentes inteligentes tele-operados utilizando ambientes virtuales," Master's thesis, Universidad Nacional Autónoma de México, 2020.

[49] J. Young, E. Sharlin, and T. Igarashi, "What is mixed reality, anyway? considering the boundaries of mixed reality in the context of robots," in *Mixed Reality and Human-Robot Interaction*, X. Wang, Ed. Springer Netherlands, 2011, pp. 1–11, https://doi.org/10.1007/978-94-007-0582-1_1.

[50] P. Milgram and F. Kishino, "A taxonomy of mixed reality visual displays," *IEICE TRANSACTIONS on Information and Systems*, vol. 77, no. 12, pp. 1321–1329, 1994.

[51] T. Lozano-Perez, J. Jones, E. Mazer, P. O'Donnell, W. Grimson, P. Tournassoud, and A. Lanusse, "Handey: A robot system that recognizes, plans, and manipulates," in *Proceedings. 1987 IEEE International Conference on Robotics and Automation*, vol. 4, 1987, pp. 843–849, https://doi.org/10.1109/ROBOT.1987.1087847.

[52] C. Borst, M. Fischer, and G. Hirzinger, "A fast and robust grasp planner for arbitrary 3d objects," in *Proceedings 1999 IEEE International Conference on Robotics and Automation (Cat. No.99CH36288C)*, vol. 3, 1999, pp. 1890–1896, https://doi.org/10.1109/ROBOT.1999.770384.

[53] A. Miller, S. Knoop, H. Christensen, and P. Allen, "Automatic grasp planning using shape primitives," in *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, vol. 2, 2003, pp. 1824–1829, https://doi.org/10.1109/ROBOT.2003.1241860.

[54] D. Berenson, J. Kuffner, and H. Choset, "An optimization approach to planning for mobile manipulation," in *2008 IEEE International Conference on Robotics and Automation*, 2008, pp. 1187–1192, https://doi.org/10.1109/ROBOT.2008.4543365.

[55] R. Diankov, N. Ratliff, D. Ferguson, S. Srinivasa, and J. Kuffner, "Bispace planning: Concurrent multi-space exploration," *Proceedings of robotics: Science and systems IV*, vol. 63, 2008.

[56] K. Harada, K. Kaneko, and F. Kanehiro, "Fast grasp planning for hand/arm systems based on convex model," in *2008 IEEE International Conference on Robotics and Automation*, 2008, pp. 1162–1168, https://doi.org/10.1109/ROBOT.2008.4543361.

[57] T. Tsuji, K. Harada, K. Kaneko, F. Kanehiro, and K. Maruyama, "Grasp planning for a multifingered hand with a humanoid robot," *Journal of Robotics and Mechatronics*, vol. 22, no. 2, pp. 230–238, 2010.

[58] K. Harada, T. Tsuji, K. Nagata, N. Yamanobe, K. Maruyama, A. Nakamura, and Y. Kawai, "Grasp planning for parallel grippers with flexibility on its grasping surface," in *2011 IEEE International Conference on Robotics and Biomimetics*, 2011, pp. 1540–1546, https://doi.org/10.1109/ROBIO.2011.6181508.

[59] D. Katz, J. Kenney, and O. Brock, "How can robots succeed in unstructured environments," in *Workshop on Robot Manipulation: Intelligence in Human Environments at Robotics: Science and Systems*, 2008.

[60] U. Thomas, B. Finkemeyer, T. Kroger, and F. Wahl, "Error-tolerant execution of complex robot tasks based on skill primitives," in *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, vol. 3, 2003, pp. 3069–3075, https://doi.org/10.1109/ROBOT.2003.1242062.

[61] H. Liu, N. Stoll, S. Junginger, and K. Thurow, "A fast approach to arm blind grasping and placing for mobile robot transportation in laboratories," *International Journal of Advanced Robotic Systems*, vol. 11, no. 3, p. 43, 2014, https://doi.org/10.5772/58253.

[62] Y. S. Choi, T. Chen, A. Jain, C. Anderson, J. D. Glass, and C. C. Kemp, "Hand it over or set it down: A user study of object delivery with an assistive mobile manipulator," in *RO-MAN 2009 - The 18th IEEE International Symposium on Robot and Human Interactive Communication*, 2009, pp. 736–743, https://doi.org/10.1109/ROMAN.2009.5326254.

[63] K. Harada, T. Tsuji, K. Nagata, N. Yamanobe, and H. Onda, "Validating an object placement planner for robotic pick-and-place tasks," *Robotics and Autonomous Systems*, vol. 62, no. 10, pp. 1463–1477, 2014, https://doi.org/10.1016/j.robot.2014.05.014.

[64] A. Cosgun, T. Hermans, V. Emeli, and M. Stilman, "Push planning for object placement on cluttered table surfaces," in *2011 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2011, pp. 4627–4632, https://doi.org/10.1109/IROS.2011.6094737.

[65] Y. Jiang and A. Saxena, "Hallucinating humans for learning robotic placement of objects," in *Experimental Robotics: The 13th International Symposium on Experimental Robotics*, J. P. Desai, G. Dudek, O. Khatib, and V. Kumar, Eds. Heidelberg: Springer International Publishing, 2013, pp. 921–937, https://doi.org/10.1007/978-3-319-00065-7_61.

[66] J. Baumgartl, T. Werner, P. Kaminsky, and D. Henrich, "A fast, GPU-based geometrical placement planner for unknown sensor-modelled objects and placement areas," in *2014 IEEE International Conference on Robotics and Automation (ICRA)*, 2014, pp. 1552–1559, https://doi.org/10.1109/ICRA.2014.6907058.

[67] K. Gillespie, K. M. Gupta, and M. Drinkwater, "Case-based object placement planning," in *Case-Based Reasoning Research and Development*, L. Lamontagne and E. Plaza, Eds. Cham: Springer International Publishing, 2014, pp. 170–184, https://doi.org/10.1007/978-3-319-11209-1_13.

[68] A. Shrivastava, T. Pfister, O. Tuzel, J. Susskind, W. Wang, and R. Webb, "Learning from simulated and unsupervised images through adversarial training," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[69] K. Bousmalis, N. Silberman, D. Dohan, D. Erhan, and D. Krishnan, "Unsupervised pixel-level domain adaptation with generative adversarial networks," in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, July 2017.

[70] Y. Ganin, E. Ustinova, H. Ajakan, P. Germain, H. Larochelle, F. Laviolette, M. Marchand, and V. Lempitsky, "Domain-adversarial training of neural networks," *Journal of machine learning research*, vol. 17, no. 1, pp. 2096–2030, 2016.

[71] K. Bousmalis, G. Trigeorgis, N. Silberman, D. Krishnan, and D. Erhan, "Domain separation networks," in *Advances in Neural Information Processing Systems*, D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett, Eds., vol. 29. Curran Associates, Inc., 2016.

[72] E. Tzeng, C. Devin, J. Hoffman, C. Finn, P. Abbeel, S. Levine, K. Saenko, and T. Darrell, "Adapting deep visuomotor representations with weak pairwise constraints," in *Algorithmic Foundations of Robotics XII: Proceedings of the Twelfth Workshop on the Algorithmic Foundations of Robotics*, K. Goldberg, P. Abbeel, K. Bekris, and L. Miller, Eds. Cham: Springer International Publishing, 2020, pp. 688–703, https://doi.org/10.1007/978-3-030-43089-4_44.

[73] F. Sadeghi and S. Levine, "CAD2RL: Real Single-Image Flight Without a Single Real Image," in *Proceedings of Robotics: Science and Systems*, Cambridge, Massachusetts, July 2017, https://doi.org/10.15607/RSS.2017.XIII.034.

[74] J. Tobin, R. Fong, A. Ray, J. Schneider, W. Zaremba, and P. Abbeel, "Domain randomization for transferring deep neural networks from simulation to the real world," in *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2017, pp. 23–30, https://doi.org/10.1109/IROS.2017.8202133.

[75] S. James, A. J. Davison, and E. Johns, "Transferring end-to-end visuomotor control from simulation to real world for a multi-stage task," in *Proceedings of the 1st Annual Conference on Robot Learning*, ser. Proceedings of Machine Learning Research, S. Levine, V. Vanhoucke, and K. Goldberg, Eds., vol. 78. PMLR, 13–15 Nov 2017, pp. 334–343.

[76] P. Christiano, Z. Shah, I. Mordatch, J. Schneider, T. Blackwell, J. Tobin, P. Abbeel, and W. Zaremba, "Transfer from simulation to real world through learning deep inverse dynamics model," *arXiv preprint arXiv:1610.03518*, 2016.

[77] J. Esteves, A. Carvalho, and C. Couto, "Generalized geometric triangulation algorithm for mobile robot absolute self-localization," in *2003 IEEE International Symposium on Industrial Electronics (Cat. No.03TH8692)*, vol. 1, 2003, pp. 346–351, https://doi.org/10.1109/ISIE.2003.1267272.

[78] A. Vedaldi and K. Lenc, "MatConvNet: Convolutional Neural Networks for MATLAB," in *Proceedings of the 23rd ACM International Conference on Multimedia*, ser. MM '15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 689–692, https://doi.org/10.1145/2733373.2807412.

[79] A. Eitel, J. T. Springenberg, L. Spinello, M. Riedmiller, and W. Burgard, "Multimodal deep learning for robust RGB-D object recognition," in *2015 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2015, pp. 681–687, https://doi.org/10.1109/IROS.2015.7353446.

[80] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[81] Y. Yang, Y. Li, C. Fermuller, and Y. Aloimonos, "Robot learning manipulation action plans by watching unconstrained videos from the world wide web," *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 29, no. 1, Mar. 2015.

[82] D. C. Ciresan, U. Meier, L. M. Gambardella, and J. Schmidhuber, "Deep, big, simple neural nets for handwritten digit recognition," *Neural Computation*, vol. 22, no. 12, pp. 3207–3220, Dec 2010, https://doi.org/10.1162/NECO_a_00052.

[83] M. Turan, J. Shabbir, H. Araujo, E. Konukoglu, and M. Sitti, "A deep learning based fusion of RGB camera information and magnetic localization information for endoscopic capsule robots," *International Journal of Intelligent Robotics and Applications*, vol. 1, pp. 442–450, 2017, https://doi.org/10.1007/s41315-017-0039-1.

[84] M. Jordan and T. Mitchell, "Machine learning: Trends, perspectives, and prospects," *Science*, vol. 349, no. 6245, pp. 255–260, July 2015, https://doi.org/10.1126/science.aaa8415.

[85] A comprehensive guide to convolutional neural networks, the ELI5 way. https://towardsdatascience.com/a-comprehensive-guide-to-convolutional-neural-networks-the-eli5-way-3bd2b1164a53.

[86] M. Turan, Y. Almalioglu, H. Araujo, E. Konukoglu, and M. Sitti, "Deep EndoVO: A recurrent convolutional neural network (RCNN) based visual odometry approach for endoscopic capsule robots," *Neurocomputing*, vol. 275, pp. 1861–1870, 2018, https://doi.org/10.1016/j.neucom.2017.10.014.

[87] B. Lake, R. Salakhutdinov, and J. Tenenbaum, "Human-level concept learning through probabilistic program induction," *Science*, vol. 350, no. 6266, pp. 1332–1338, 12 2015, https://doi.org/10.1126/science.aab3050.

[88] S. Levine, P. Pastor, A. Krizhevsky, J. Ibarz, and D. Quillen, "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection," *The International Journal of Robotics Research*, vol. 37, no. 4-5, pp. 421–436, March 2016, https://doi.org/10.1177/0278364917710318.

[89] Y. Yang, C. Fermüller, Y. Li, and Y. Aloimonos, "Grasp type revisited: A modern perspective on a classical feature for vision," in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015, pp. 400–408, https://doi.org/10.1109/CVPR.2015.7298637.

[90] C. Dong, C. C. Loy, K. He, and X. Tang, "Image super-resolution using deep convolutional networks," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 2, pp. 295–307, Feb 2016, https://doi.org/10.1109/TPAMI.2015.2439281.

[91] A. M. Nguyen, J. Yosinski, and J. Clune, "Innovation engines: Automated creativity and improved stochastic optimization via deep learning," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '15. New York, NY, USA: Association for Computing Machinery, 2015, pp. 959–966, https://doi.org/10.1145/2739480.2754703.

[92] N. Sünderhauf, S. Shirazi, F. Dayoub, B. Upcroft, and M. Milford, "On the performance of ConvNet features for place recognition," in *2015 IEEE/RSJ international conference on intelli-*

gent robots and systems (IROS). IEEE, 2015, pp. 4297–4304, https://doi.org/10.1109/IROS.2015.7353986.

[93] J. Schmidhuber, "Deep learning in neural networks: An overview," *Neural Networks*, vol. 61, pp. 85–117, 01 2015, https://doi.org/10.1016/j.neunet.2014.09.003.

[94] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "ImageNet: A large-scale hierarchical image database," in *2009 IEEE Conference on Computer Vision and Pattern Recognition*, 2009, pp. 248–255, https://doi.org/10.1109/CVPR.2009.5206848.

[95] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft COCO: Common Objects in Context," in *Computer Vision – ECCV 2014*, D. Fleet, T. Pajdla, B. Schiele, and T. Tuytelaars, Eds. Cham: Springer International Publishing, 2014, pp. 740–755, https://doi.org/https://doi.org/10.1007/978-3-319-10602-1_48.

[96] E. Silva, "Deteccion de objetos con redes neuronales profundas para un robot de servicio," Master's thesis, Universidad Nacional Autónoma de México, 2020.

[97] YOLO Object Detection. https://appsilon.com/object-detection-yolo-algorithm/.

[98] D. H. Hubel and T. N. Wiesel, "Receptive fields, binocular interaction and functional architecture in the cat's visual cortex," *The Journal of physiology*, vol. 160, no. 1, pp. 106–154, 1962, https://doi.org/10.1113/jphysiol.1962.sp006837.

[99] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*. MIT Press, 2016.

[100] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[101] Deteccion de objetos. https://www.aprendemachinelearning.com/modelos-de-deteccion-de-objetos/.

[102] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *arXiv preprint arXiv:1804.02767*, 2018.

[103] H. Fukui, T. Hirakawa, T. Yamashita, and H. Fujiyoshi, "Attention branch network: Learning of attention mechanism for visual explanation," in *2019 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, 2019, pp. 10 697–10 706, https://doi.org/10.1109/CVPR.2019.01096.

[104] A. Magassouba, K. Sugiura, and H. Kawai, "Multimodal attention branch network for perspective-free sentence generation," in *Proceedings of the Conference on Robot Learning (CoRL)*, ser. Proceedings of Machine Learning Research, L. P. Kaelbling, D. Kragic, and K. Sugiura, Eds., vol. 100. PMLR, 30 Oct–01 Nov 2020, pp. 76–85.

[105] ——, "A multimodal target-source classifier with attention branches to understand ambiguous instructions for fetching daily objects," *IEEE Robotics and Automation Letters*, vol. 5, no. 2, pp. 532–539, 2020, https://doi.org/10.1109/LRA.2019.2963649.

[106] A. Aakerberg, K. Nasrollahi, C. B. Rasmussen, and T. B. Moeslund, "Depth value pre-processing for accurate transfer learning based RGB-D object recognition," in *Proceedings of the 9th International Joint Conference on Computational Intelligence, IJCCI 2017*, C. Sabourin, J. J. M. Guervós, U. O'Reilly, K. Madani, and K. Warwick, Eds. SciTePress, 2017, pp. 121–128, https://doi.org/10.5220/0006511501210128.

[107] C. Hori, T. Hori, T. K. Marks, and J. R. Hershey, "Early and late integration of audio features for automatic video description," in *2017 IEEE Automatic Speech Recognition and Understanding Workshop (ASRU)*, Dec 2017, pp. 430–436, https://doi.org/10.1109/ASRU.2017.8268968.

[108] A. Magassouba, K. Sugiura, A. Nakayama, T. Hirakawa, T. Yamashita, H. Fujiyoshi, and H. Kawai, "Predicting and attending to damaging collisions for placing everyday objects in photo-realistic simulations," *Advanced Robotics*, vol. 35, no. 12, pp. 787–799, 2021, https://doi.org/10.1080/01691864.2021.1913446.

[109] T. Shibata, "Head mounted display," *Displays*, vol. 23, no. 1, pp. 57–64, 2002, https://doi.org/10.1016/S0141-9382(02)00010-0.

[110] D. Parisi, "A counterrevolution in the hands: The console controller as an ergonomic branding mechanism," *Journal of Games Criticism*, vol. 2, no. 1, pp. 1–23, 2015.

[111] P. R. Desai, P. N. Desai, K. D. Ajmera, and K. Mehta, "A Review Paper on Oculus Rift-A Virtual Reality Headset," *arXiv preprint arXiv:1408.1173*, 2014.

[112] R. Webster and J. F. D. Jr., "System Usability Scale (SUS): Oculus Rift®DK2 and Samsung Gear VR®," in *2017 ASEE Annual Conference & Exposition*, no. 10.18260/1-2–28899. Columbus, Ohio: ASEE Conferences, June 2017, https://peer.asee.org/28899.

[113] J. Savage, A. LLarena, G. Carrera, S. Cuellar, D. Esparza, Y. Minami, and U. Peñuelas, "ViRbot: A system for the operation of mobile robots," in *RoboCup 2007: Robot Soccer World Cup XI*, U. Visser, F. Ribeiro, T. Ohashi, and F. Dellaert, Eds. Berlin, Heidelberg: Springer Berlin

Heidelberg, 2008, pp. 512–519, https://doi.org/10.1007/978-3-540-68847-1_55.

[114] R. H. Creighton, *Unity 3D game development by example: A Seat-of-your-pants manual for building fun, groovy little games quickly.* Packt Publishing Ltd, 2010.

[115] ROS web video server package. http://wiki.ros.org/web_video_server.

[116] M. Bjelonic. (2016–2018) YOLO ROS: Real-time object detection for ROS. https://github.com/leggedrobotics/darknet_ros.

[117] Libfreenect2 driver. https://github.com/OpenKinect/libfreenect2.

[118] T. Wiedemeyer. (2014 – 2015) IAI Kinect2. https://github.com/code-iai/iai_kinect2. Institute for Artificial Intelligence. University Bremen.

[119] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein *et al.*, "Imagenet large scale visual recognition challenge," *International journal of computer vision*, vol. 115, no. 3, pp. 211–252, 2015, https://doi.org/10.1007/s11263-015-0816-y.

[120] A. Nakayama, D. Ruelas, J. Savage, and E. Bribiesca, "Teleoperated Service Robot with an Immersive Mixed Reality Interface," *Informatics and Automation*, vol. 20, no. 6, pp. 1187–1223, Dec. 2021. [Online]. Available: http://www.proceedings.spiiras.nw.ru/index.php/sp/article/view/15069

[121] J. Savage, A. Nakayama, and C. Sarmiento, "Robotics, AI and Machine Vision," in *Proceedings of Artificial Intelligence for Science, Industry and Society — PoS(AISIS2019)*, vol. 372, October 2020, p. 043, http://doi.org/10.22323/1.372.0043.

[122] Human support robot. https://blog.siggraph.org/tag/human-support-robot/.

[123] T. Yamamoto, K. Terada, A. Ochiai, F. Saito, Y. Asahara, and K. Murase, "Development of human support robot as the research platform of a domestic mobile manipulator," *ROBOMECH Journal*, vol. 6, no. 1, p. 4, Apr 2019, https://doi.org/10.1186/s40648-019-0132-3.

[124] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA Workshop on Open Source Software*, vol. 3. Kobe, Japan, 2009, p. 5.

[125] T. Inamura and Y. Mizuchi, "SIGVerse: A Cloud-Based VR Platform for Research on Multimodal Human-Robot Interaction," *Frontiers in Robotics and AI*, vol. 8, p. 158, 2021, https://doi.org/10.3389/frobt.2021.549360.

[126] Sigverse github. https://github.com/SIGVerse.