PLATAFORMA ROBÓTICA MÓVIL CON RETROALIMENTACIÓN HÁPTICA

RUBÉN DARÍO MORENO GUERRA

UNIVERSIDAD DE PAMPLONA
FACULTAD DE INGENIERÍAS Y ARQUITECTURA
DEPARTAMENTO DE INGENIERIA MECÁNICA, INDUSTRIAL Y MECATRÓNICA
PAMPLONA, COLOMBIA
2020

PLATAFORMA ROBÓTICA MÓVIL CON RETROALIMENTACIÓN HÁPTICA

RUBÉN DARÍO MORENO GUERRA

TRABAJO DE GRADO PARA OPTAR POR EL TÍTULO DE INGENIERO EN MECATRÓNICA

CÉSAR AUGUSTO PEÑA CORTÉS
DOCTOR EN AUTOMÁTICA Y ROBÓTICA

UNIVERSIDAD DE PAMPLONA
FACULTAD DE INGENIERÍAS Y ARQUITECTURA
DEPARTAMENTO DE INGENIERIA MECÁNICA, INDUSTRIAL Y MECATRÓNICA
PAMPLONA, COLOMBIA
2020

Nota de Aceptación

_____

_____

_____

_____

Presidente del Jurado

_____

Jurado

_____

Jurado

Pamplona Norte de Santander, 18 junio 2020

Dedicatoria

Esta tesis está dedicada especial a mi querida madre Ruby Guerra Torres, porque a pesar de que fue difícil levantar a tres hijos sin la ayuda de una figura paterna en casa, nos ayudó a salir a delante a todos con la ayuda de Dios. Gracias por apoyarme en este deseo loco de graduarme algún día como ingeniero en Mecatrónica y por ser una madre esforzada y valiente. Gracias por enseñarnos a amar a papá.

Por último y de corazón, quiero dedicar esta tesis a todos aquellos lifelong learners apasionados por el conocimiento puesto en marcha, en pro del mejoramiento de la calidad de vida de nuestra sociedad.

AGRADECIMIENTOS

A mi director de tesis PhD. Cesar Augusto Peña Cortes por su acompañamiento, paciencia y horas de accesoria durante todo este proceso. También por auspiciar en gran parte los materiales y equipos que fueron necesarios para el desarrollo de esta tesis. De él aprendí el análisis y criterio lógico para la interpretación y desarrollo de códigos de programación, así como a estar seguro de los conocimientos y fundamentos adquiridos durante el proceso de investigación.

A mi amigo MSc. Parmâan Mohammad Alizadeh que desde la distancia, estuvo allí brindándome de su tiempo, conocimiento y apoyo incondicional, en aquellos momentos donde más lo necesité. Gracias a la cooperación mutua y debido a su experiencia en sistemas informáticos, el hacerme a los conocimientos en redes y comunicación entre dispositivos robóticos necesarios para el desarrollo de esta tesis fue verdaderamente gratificante.

A mí querida esposa Claudia Peralta Beltrán, por siempre estar a mi lado, inicialmente como novia en los primeros semestres en la Universidad, y como esposa durante todo el transcurso de la carrera. Gracias te doy por ser mi apoyo emocional y porque al fin logramos juntos nuestro sueño de ser ingenieros. Finalmente, el saber que estábamos esperamos una bebe, fue un motorcito extra que me impulsó a concluir esta tesis con mucho más esmero.

A mi cuñada Diana Hellen Rueda Beltrán y a mi suegra María Helena Beltrán por su calor familiar, y por brindarnos apoyo económico a mi esposa y a mí justo cuando nuestros prestamos de estudio emperezaban a agotarse. De igual manera agradecer a mi suegro Hildebrando Peralta y a mi querida madre Ruby Guerra Torres por sus aportes económicos que con tanto esfuerzo nos hicieron llegar.

Finalmente agradecer a la Universidad de Pamplona por ser el lugar donde se formaron muchas de las habilidades con las que ahora cuento, esto de la mano de todo un cuerpo docente que con mucho sacrificio y amor hicieron participes a sus estudiantes de su conocimiento y experiencias necesarias para forjar a un profesional más. A todos, mil gracias.

Contenido

Índice de figuras:

Índice de tablas:

CAPITULO I

1. RESUMEN DEL PROYECTO

Este trabajo hace parte de un proyecto macro denominado "**teleoperación mediante control compartido humano-robot vía señales hápticas y neuroseñales"** el cual se está llevado a cabo en la Universidad de Pamplona por parte de algunos docentes del programa de Ingeniería Mecatrónica. La idea general de este proyecto es recrear un experimento de forma física, el cual se había planteado previamente en una simulación realizada en la plataforma virtual V-REP. La simulación previa consistía en teleoperar un robot Pioneer en un entorno estructurado, utilizando el dispositivo Novint Falcon de 3 grados de libertad, por medio del cual se establecen las asistencias de retroalimentación háptica.

Debido a los grandes costos del robot Pioneer, en este proyecto se pretende la construcción de una plataforma robótica de bajo costo, que permita la detección de obstáculos a su alrededor. Esta plataforma será controlada por medio del dispositivo háptico. La plataforma requerirá de un módulo de comunicación para recibir las referencias del teleoperador y a su vez enviar la información de los sensores encargados de detectar los obstáculos al controlador para general la asistencia háptica correspondiente.

Adicionalmente, la localización del robot dentro del entorno estructurado, se realizara por medio de un sistema de visión artificial.

Palabras claves: Plataforma Robótica, Dispositivo háptico, Sensores, evasión de obstáculos.

1.1 INTRODUCCIÓN

El presente trabajo tiene como objetivo principal la construcción de una plataforma robótica móvil que será teleoperada a través del dispositivo háptico Novint Falcon de 3GDL.

El trabajo se estructurara de la siguiente manera: En el capítulo I, en el cual nos encontramos, se hace una introducción a este trabajo explicando por qué se ha elegido la temática, qué pretendemos conseguir con él, sus delimitaciones y en qué dirección están avanzando las investigaciones sobre esta temática en la actualidad a través del estado del arte.

En el capítulo II, se habla del marco teórico, incluyendo todos aquellos conceptos elementales sobre los sistemas móviles autónomos, tipos de estructura, cinemática y control. Finalizando con una breve introducción a la tecnología háptica y los protocolos de comunicación en sistemas embebidos.

El capítulo III, denominado como descripción de los sensores y actuadores, trata a detalle la descripción técnica de cada uno de los componentes y piezas electrónicas utilizadas en el proyecto. También, se analiza alguna las placas de desarrollo disponibles en el mercado y se describe el por qué la elección de una de ellas de acuerdo a las necesidades del proyecto.

El capítulo IV, es el desarrollo y puesta en marcha de la teoría expuesta en los capítulos anteriores. En esta parte se habla del diseño electromecánico de la plataforma robótica, así como del sistema del control implementado, con una explicación de los entornos de programación y un paso a paso del código tanto en la parte del servidor y cliente.

El capítulo V, destaca los resultados obtenidos y algunas pruebas para observar el desempeño del robot y aspectos a mejorar.

Y por último, el capítulo VI, se consolidan las conclusiones, puntos a favor y en contra del trabajo realizado. Terminado con la bibliografía consultada y la sección de anexos.

1.2 PLANTEAMIENTO DEL PROBLEMA Y JUSTIFICACIÓN

Últimamente, las industrias de tecnología, a través de los años han desarrollado he incorporado un gran número de aplicaciones, relacionadas con la teleoperación háptica, esto gracias a importantes investigaciones realizadas en universidades y centro de desarrollo e investigación fuertemente reconocidos a nivel nacional e internacional en este campo; la Universidad de Pamplona recientemente ha ampliado su repertorio investigativo con el fin de profundizar y estar a la vanguardia, aprovechado el personal docente con un reconocido desempeño en el campo.

No tan solo en la parte de la teleoperación háptica, sino también, las estructuras de plataformas robóticas móviles que hoy en día lo que se busca es que exista el menor número de limitantes en el entorno de trabajo, para que el robot pueda desempeñarse de la mejor manera posible, esto con una plataforma robótica que pueda adaptarse a condiciones cambiantes del entorno, como por ejemplo, en ambientes industriales dinámicos la evasión de obstáculos en el camino es un factor muy importante.

Con base en esto, se busca seguir trabajando e investigando en estas tecnologías, teniendo en cuenta el factor económico promedio de los países subdesarrollados, que a la larga es muy limitado, así, para que los estudiantes no solo de la Universidad de Pamplona, sino también en cualquier otra universidad o lugar, puedan realizar proyectos de este calibre a un precio asequible, esto con la utilización de recursos como impresoras 3D, parte del diseño enfocado en este punto y la ingeniera inversa realizable a proyectos existentes de código libre; de esta manera promover e incentivar a que cada vez más estudiantes no se sientan cohibidos económicamente para trabajar en estos temas.

Por otra parte, se ponen a prueba conocimientos existentes en el área para el desarrollo y puesta en marcha del proyecto; a su vez dejar un referente bibliográfico local al que se pueda recurrir con el fin de ser una ayuda en futuros trabajos e investigaciones tomando como base el proyecto físico que se piensa será parte del laboratorio de robótica de la universidad de Pamplona.

1.3 ESTADO DEL ARTE

Desde la aparición de los primeros conceptos de robótica, se ha visto la evolución a través de los años y como se ha ido compaginado en la vida cotidiana con un gran abanico de aplicaciones y campos de acción, partiendo desde robots manipuladores, robots de entretenimiento y de servicio, hasta robots móviles (Ortigoza, 2007). Este estado del arte pretende ser una introducción a los robots móviles con ruedas, donde se aborda un poco de historia de la robótica en general así como los desarrollos más relevantes que han ido aportando las bases del conocimiento en esta rama de la robótica.

La robótica nace como ciencia a partir del siglo XX, hacía la década de los cuarenta. Los orígenes de la misma, datan del siglo XVIII con la construcción de autómatas humanoides por Vaucanson y los Jaquet-Droz. La palabra autómata se puede encontrar también en la antigua era judía, china y en las legendas Indias. La idea de que un autómata mecánico se pareciera a los humanos o a los animales fue cobrando vida en la literatura a través de la historia, volviéndose popular en el siglo XIV, especialmente en el siglo XX, fue donde encontró el medio de representar y traer a la vida real a los autómatas a través de películas de ciencia ficción. Si bien con el paso del tiempo se desarrolló un gran número de figuras dotadas de partes móviles, fue gracias a la revolución industrial en los albores del siglo XIX (históricamente el hito más importante en el desarrollo de nuevas tecnologías) que se impulsó el crecimiento de las diferentes áreas científicas y la creación de nuevos dispositivos aplicables a distintos rubros (i.e. construcción, milicia, aeronáutica, transporte, etc.), que posteriormente convergieron en la concepción de la robótica.(Klančar, Zdešar, Blažič, & Škrjanc, 2017)

Un factor muy importante donde está centrada gran parte de las investigaciones en los robots con ruedas, es en la integración de mecanismos híbridos que permitan incorporar distintos estilos de locomoción. La universidad de defensa y tecnología, China, y la universidad de Salford en compañía de la universidad de Manchester, Reino Unido, en una de sus publicaciones más recientes, denominada “reconfiguración hibrida de un robot móvil con oruga basado en el mecanismo Watt II de 6 barras” desarrolla un prototipo robótico, RHMbot (Luo, Shang, Wei, & Ren, 2018) capaz de cambiar entre sus tres modos de locomoción, entre esos: modo de locomoción con ruedas, donde su desempeño se basa en terrenos llanos; modo de oruga triangular para tracción o al encontrar pequeños obstáculos y modo de ascenso utilizando su marco en forma de cola a ambos costados que le brinda soporte para dar vuelcos en caso de que se encuentren obstáculos mucho más grandes. La fig. 1 y 2 brindan una perspectiva de la estructura de este prototipo.

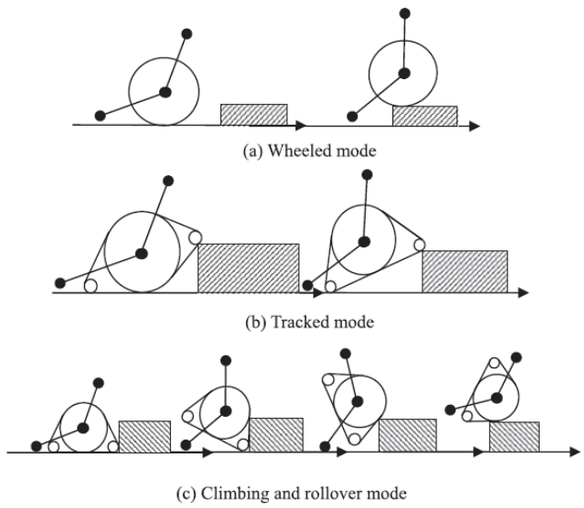


Figura 1. Modos de locomoción del robot móvil RHMbot (Luo et al., 2018)

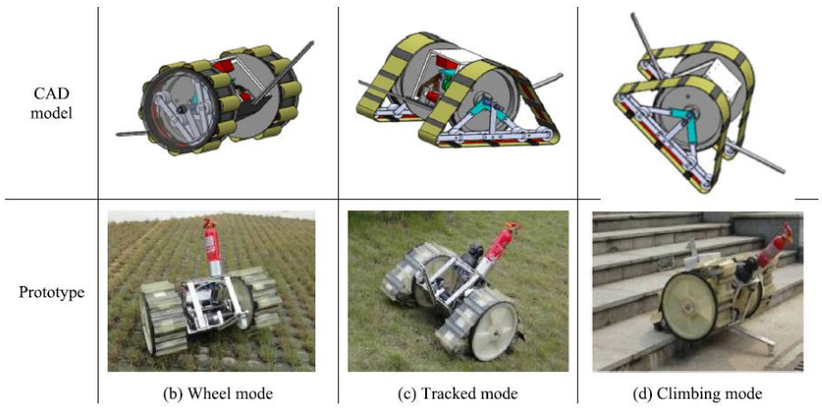


Figura 2. Modelo en CAD y prototipo del robot móvil RHMbot (Luo et al., 2018)

.

Por otra parte el instituto de tecnología y la HIT, China. En su artículo titulado “desarrollo de un robot móvil de oruga transformable y su modo de selección de sobrepasar obstáculos” publicado el año 2014 en una conferencia internacional de robótica y automatización realizada en la ciudad de Tianjin, China. Propone el desarrollo de un prototipo robótico basado en un mecanismo de 4 barras con una especie de palanca retraible en uno de sus costados que sirve como apoyo en caso de la necesidad de sobrepasar obstáculos y/o pequeñas escaleras (Guo, Jiang, Zong, & Gao, 2014). En su momento resaltaron que tenían falencias en cuanto a la fuerza de agarre en modo oruga triangular y al momento de mantener la redondez cuando se cambiaba al modo rueda. En la fig. 3 da una idea de la estructura del robot.

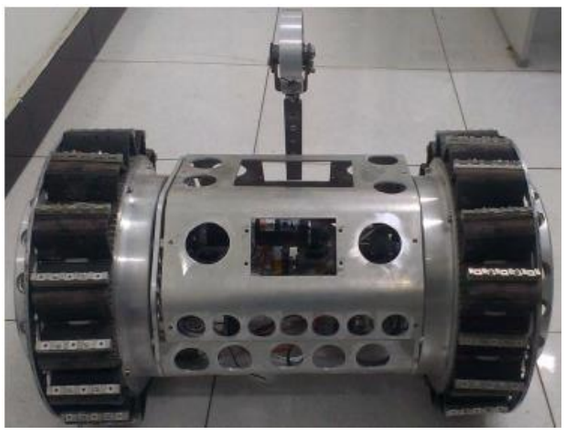


Figura 3. Robot móvil con oruga en modo rueda (Guo et al., 2014)

En un trabajo liderado por la academia de agricultura y ciencias forestales, Beijing, China. Llamado “Extensión del principio de dirección Ackerman para el control de movimientos coordinados de un robot móvil agrícola de tracción en las cuatro ruedas” (Qiu et al., 2018). Detalla los primeros y últimos estudios relacionados con los robots móviles en la industria agrícola, profundizando en una estrategia de control para el movimiento coordinado de una estructura Ackerman operada remotamente en cultivos abiertos. Los alcances de este trabajo se basan en un decremento sustancial de posibles deslizamientos de la plataforma móvil al momento de hacer giros rápidos en forma de zigzag o en círculos.

Otras investigaciones en el campo de locomoción diferencial en robots con ruedas, basa sus ideales en la mejora del deslizamiento que frecuentemente se presenta y que afecta el desempeño y coordinación de robots múltiples. Un estudio realizado en la universidad del estado de Oklahoma, USA, titulado “Efectos del deslizamiento en la coordinación de robots móviles con ruedas” (Konduri, Cobos Torres, & Pagilla, 2014). Señala que las llantas del robot suelen estar sometidas frecuentemente a aceleraciones y desaceleraciones necesarias para moverse de acuerdo a las exigencias del entorno, siendo muchas veces inevitable el deslizamiento y por consecuente la descoordinación. La metodología empleada fue el modelo de fuerza de tracción basada en la fricción de Coulomb, que ayuda a distinguir los momentos de no deslizamiento y deslizamiento del robot, así, de acuerdo al modelo, automáticamente se controla o limita el torque de las llantas en un posible deslizamiento.

Por su parte la Universidad de tecnología Gdansk, Polinia, realizo sus estudios en la corrección las vibraciones indeseadas de una plataforma móvil de un robot diferencial de bajo nivel de control, de la misma manera reduciendo el índice de rendimiento energético. El método propuesto fue la corrección de los errores de posición y orientación a través de la cinemática inversa con la ayuda de las ecuaciones de Appell–Gibbs (Kaliński & Mazur, 2016). Según una comparación realizada con un modelo PID para el mismo propósito, el método propuesto estaba muy por encima con mejores resultados.

En el campo de los sensores utilizados y métodos para copilar la mayor información del ambiente y lograr un desempeño cada vez mucho más eficiente y preciso de la labor requerida, se trabaja en algoritmos y sistemas de navegación para robots móviles. La universidad Nueva Gales del Sur, Sídney, Australia, en uno de sus más

recientes trabajos, le apuntan al desarrollo de un sensor de red basado en sistemas de navegación en ambientes dinámicos industriales cerrados, donde la detección de obstáculos se lleva a cabo gracias a un rastreador de ruta de bajo nivel y algoritmos de computo incorporados en el robot (Li & Savkin, 2018). La fig. 4 muestra el trabajo que hasta el momento se encuentra en su fase de prototipo pero con grandes ideales en entornos dinámicos y complejos.

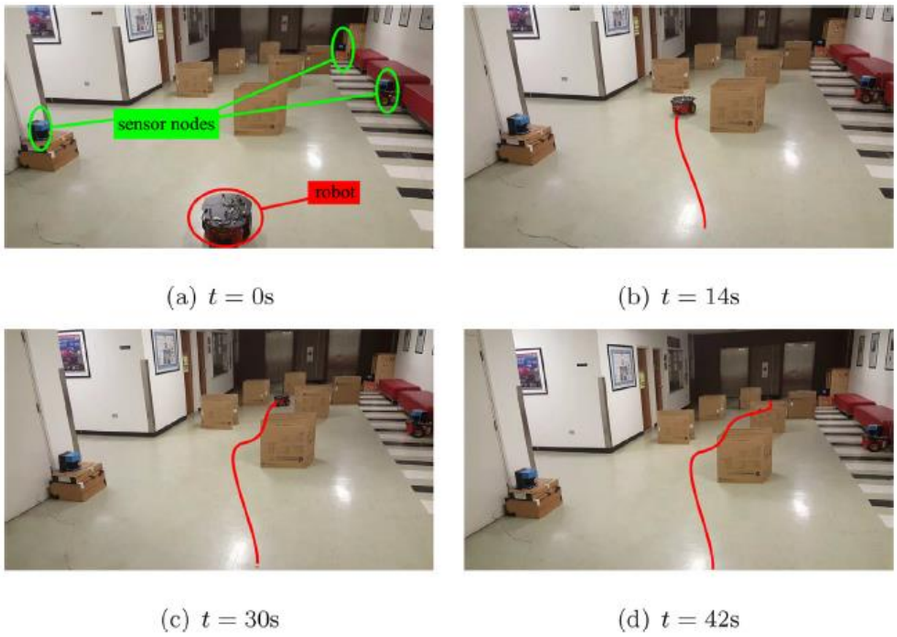


Figura 4. Experimento de la red se sensores, simulación de ambiente con obstáculos estáticos (Li & Savkin, 2018)

En la rama de la háptica, la robótica también ha estado haciendo presencia a través de los años. Tras sus múltiples investigaciones, la comunidad científica, estudiantes y expertos interesados en el tema, han podido desvelar diferentes conceptos y avances no tan visibles años atrás. Diferentes estudios han dejado significantes aportes a esta rama. Por citar; una nueva interfaz háptica para la teleoperación de robots móviles, propuesta en el 2010 por la universidad de tecnología y educación en Cheonan, Corea (Nguyen & Ryu, 2010). Basada en un diseño de volante para dirigir el móvil esclavo en sus sentidos de giro, pero también, para controlar su velocidad de marcha. El experimento se llevó a cabo a través de una simulación computarizada. La fig. 5 se puede ver la apariencia de la interface háptica propuesta.

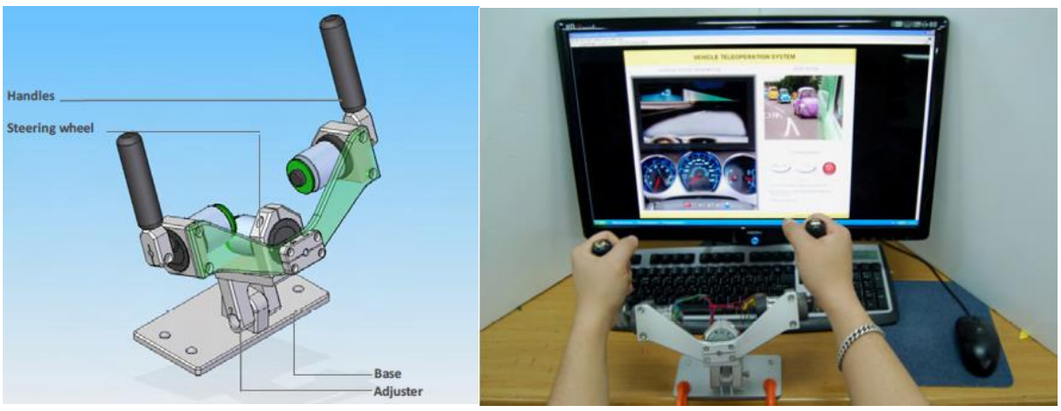


Figura 5. Izquierda: CAD de la interface háptica. DERECHA: Simulación de mando realizada de forma computarizada (Nguyen & Ryu, 2010).

Otro estudio realizado en la Universidad de Massachusetts Lowell. Denominado "Métodos para la evaluación y comparación del uso la realimentación háptica en la interacción humano – robot con un robot móvil terrestre" (Brooks, Tsui, Lunderville, & Yanco, 2015). Dio a conocer cómo las personas, entre estudiantes y trabajadores del común interpretan y/o reaccionan a los efectos de la retroalimentación háptica mientras se está teleoperando un robot móvil. Abriendo paso al desarrollo de un kit de herramientas de evaluación háptica para marcar pautas investigativas en la creación de nuevos dispositivos hápticos para la teleoperación de robots móviles terrestres.

La háptica también ha sido ampliamente utilizada en los vehículos operados remotamente o ROV. Esto mediante el uso de sensores de presencia que transmiten la información al joystick háptico de control, el cual a su vez genera una fuerza directamente proporcional a la distancia del obstáculo con el fin de alertar al operador de la proximidad del obstáculo y prevenir colisiones. Algunos experimentos se han llevaron a cabo en entornos de realidad virtual bajo el agua, aplicando algoritmos FCM-KF y la formula APF para el control de evasor de obstáculos (Le, Nguyen, Ranmuthugala, & Forrest, 2016). Estos sistemas de manejo háptico pueden mejorar la seguridad y la maniobrabilidad de los vehículos teleoperados remotamente en ambientes cerrados. La fig. 6 muestra un esquema del sistema de conducción grafica propuesto.

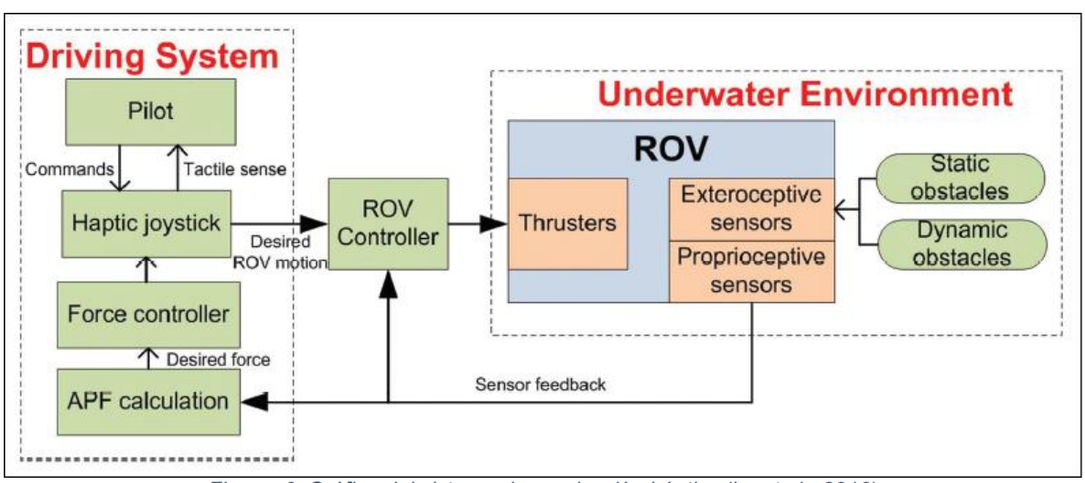


Figura 6. Gráfica del sistema de conducción háptica (Le et al., 2016)

Pasando al campo de las plataformas robóticas móviles más utilizadas hasta nuestros tiempos, debemos tener en cuenta que la mayoría de las investigaciones y proyectos realizados apuntan a prototipos de bajo costo (Sempere, Serna-Leon, Gil, Puente, & Torres, 2015) y (Gunawan et al., 2017), para que puedan ser replicados en otros escenarios investigativos sin mayor complicación. Hoy en día se cuenta con muchas plataformas comerciales que son bastante utilizadas tanto en proyectos universitarios como en colegios y escuelas, y muchas veces tomados como base en proyectos investigativos de mayor complejidad ("Kit Robot MiniQ 2WD v2.0 - VISTRONICA SAS," 2018), ("Kit Chasis con Motores 2WD miniQ - VISTRONICA SAS," 2018) y ("3WD Triangular 100mm omni wheel mobile robotics car," 2018). Por otro lado existe en la web diversos tipos de plataformas robóticas de código libre ya desarrolladas por: desde amateurs a profesionales en el campo que pueden ser fácilmente adaptadas a la necesidad de un nuevo proyecto ("SMARS modular robot by tristomietitoredeituit - Thingiverse," 2017) y (MakeBlock Robotics, n.d.). Muchas veces también se recurre a este factor para agilizar contratiempos mediante la compra directa en una tienda local de robótica.

Por último las plataformas mucho más sofisticadas, desarrolladas por empresas reconocidas ("Robot móvil SUMMIT XL HL | Robotnik," 2017), (Kuka Robotics, 2014) y (Adept, 2011), las podemos obtener a un coste mucho más considerable pero que a su vez integran un mayor número de tecnologías y por ende se deja entrever un mejor desempeño en sus diferentes campos de aplicación, tales como la educación, vigilancia, militares, monitorización remota y acceso a entornos adversos.

1.4 OBJETIVO GENERAL

Construir una plataforma robótica para teleoperar un robot móvil a través de un dispositivo háptico.

1.5 OBJETIVOS ESPECÍFICOS

- Diseñar la estructura electromecánica del robot.
- Implementar el sistema de detección de obstáculos.
- Seleccionar el módulo de comunicaciones.
- Generar el algoritmo de control para las asistencias hápticas.

1.6 DELIMITACIONES

- En la selección de los componentes se tendrá en cuanta la accesibilidad al mercado local.
- Plataforma robótica de bajo costo.
- Superficies planas, no apto para exteriores.
- No será un robot velocista.

1.7 RESULTADOS/PRODUCTOS ESPERADOS Y POTENCIALES BENEFICIARIOS

El factor que motiva el desarrollo de este trabajo es la investigación, en primer lugar, de los avances realizados hasta el momento en el campo de la robótica móvil, con base en ello, generar ideas que contribuyan en cierta medida a alcanzar los objetivos propuestos por el equipo de trabajo, y como segundo, la implementación real del robot teleoperado a través del dispositivo Novint Falcon.

Por otra parte, dar a conocer los aspectos necesarios que permitan al lector neófito, tener un acercamiento claro acerca del proyecto planteado, con el fin de que se pueda tomar o mejorar aspectos del mismo en futuros trabajos relacionados.

CAPITULO II

2. MARCO TEÓRICO

En este apartado se hará una breve introducción de conceptos generales necesarios para brindar una perspectiva sobre el tema y se tomarán en cuenta los conocimientos previos fundamentales que el lector debe reconocer y asimilar, para así abordar el campo de la robótica móvil y la retroalimentación háptica de una manera más abierta y tener un criterio investigativo para que pueda continuar con futuras investigaciones en esta rama.

2.1 SISTEMAS MÓVILES AUTÓNOMOS

Un sistema móvil puede ser definido como un sistema que no está fijo en un lugar, sino por el contrario, puede moverse a ciertos espacios del entorno. Se clasifican de acuerdo al entorno donde se mueva (Klančar et al., 2017):

- Sistemas terrestres
- Sistemas aéreos
- Sistemas acuáticos y subacuáticos

En este caso, debido a que nuestro enfoque trata sobre robots móviles con ruedas no detallaremos los otros dos sistemas.

Un sistema móvil es considerado autónomo cuando realiza movimientos dentro de su entorno de forma autónoma sin intervención humana. Su nivel de autonomía se define desde los siguientes puntos de vista (Klančar et al., 2017):

- Energía: el robot lleva su propia fuente de energía. No es dependiente a un cable de alimentación permanentemente.
- Toma decisiones: El robot es capaz de tomar ciertas decisiones y tomar las acciones apropiadas.

En realidad esto significa que el sistema móvil toma las instrucciones proporcionadas por el operador humano y de acuerdo a las capacidades del sistema, este las ejecuta en cierto intervalo de tiempo en caso de no existir circunstancias imprevistas. Basado en el nivel de autonomía del robot, los siguientes son los comandos que pueden ser tomados del operador (Klančar et al., 2017):

2.1.1 Velocidad deseada de las ruedas

El robot toma los comandos recibidos por el operario para la velocidad de sus ruedas, apoyándose de su básico algoritmo de control y a través de los sensores.

Generalmente con el uso de encoders rotatorios para asegurar un buen control de movimiento.

2.1.2 Velocidades angular y longitudinal deseadas

El algoritmo inmerso en el programación del robot contiene el modelo cinemático, encargado de calcular la velocidad apropiada de las ruedas para lograr la velocidad optima a la que se debe mover el robot.

2.1.3 Trayectoria deseada

El robot es capaz de determinar y controlar su ubicación dentro del entorno; la ubicación se define usualmente como la información que integra la posición y la orientación relativa para algunas coordenadas del sistema. Con el fin, de obtener la mejor información posible de la ubicación del robot; se utilizan diferentes sensores montados en su estructura o en el entorno de trabajo (Li & Savkin, 2018), sin dejar de lado los problemas de control que surgen debido a la no-linealidad del sistema o a la información errónea por parte de los sensores, como el deslizamiento (Konduri et al., 2014), modelos matemáticos insuficientes, retrasos en la transmisión y recepción de la información.

2.1.4 Operaciones deseadas dentro de entornos conocidos pero con posible obstáculos

En este punto el robot planifica la ruta de acuerdo a su objetivo, en caso de presentarse obstáculos estáticos o en movimiento que interfieran con la ruta planificada, el robot tendrá que re-planificar una nueva ruta para evadir los obstáculos y cumplir su objetivo.

2.1.5 Operaciones deseadas dentro de entornos desconocidos

El robot desconoce completamente su entorno, teniendo que llevar a cabo acciones simultaneas de localización y mapeado del entorno para conocer su ubicación; esta técnica se conoce como *mapeado y localización simultanea "*SLAM".

2.1.6 Misión deseada

El robot inicia la misión que tiene que cumplir, operando en un entorno donde posiblemente debe colaborar con otros robots o agentes (Gunawan et al., 2017). El robot necesita tener ciertas prioridades y un cierto entendimiento de su misión, para poder eludir ciertas misiones y tomar otras con mayor prioridad. Un buen ejemplo es que el robot debe inspeccionar su nivel de batería y cargarla en caso de ser necesario.

2.2 SISTEMA MÓVIL TERRESTRE

Aquí se encuentran varios robots móviles con ruedas u orugas, robot con patas (humanoides o imitación de animal). Estos sistemas a menudo no transportan al operario y se les conoce como vehículos terrestres no tripulados.

2.2.1 Robots móviles con ruedas

Existen muchas actividades que hacen atractivo el uso de robots móviles en ambientes cotidianos he industriales, desde el acceso remoto a lugares inaccesibles, hasta la realización de tareas, que físicamente afectarían la integridad del ser humano, mencionando además, que suelen ser más productivos en tareas repetitivas y mejor calidad de servicio. Así, se deja entrever que hoy en día los robots móviles son utilizados en un extenso número de aplicaciones. Por citar algunas:

- Agricultura, largas jornadas de trabajo (Harvest Automation, 2008).
- Servicio al cliente, almacenes de cadenas, logística. (Ackerman, 2013)
- Inspección y vigilancia, bombas antipersona, reactores nucleares. (SMP Robotics, 2009)
- Misiones espaciales. (Greicius, 2015)
- Carga y descarga de materiales pesados, en aviones, barcos, trenes. (Ackerman, 2013)
- Campos militares, provisión de explosivos, alimentos, emboscadas. (iRobot, 2017)
- Rehabilitación, sillas de ruedas autónomas. (Gross et al., 2017)
- Educación, robots dedicados a investigaciones y desarrollo de algoritmos. (Adept, 2011)

Esto a grandes rasgos, son los campos donde los robots móviles han estado haciendo presencia. Cada vez más, existe la tendencia de potenciar estructuras robóticas ya conocidas que ofrezcan mejores desempeños que las ya conocidas (Transcend Robotics, 2015).

2.2.2 Tipos de estructuras de robots móviles con ruedas

Existen cuatro tipos principales de locomoción, las cuales se seleccionan de acuerdo a la función objetivo que realizará el robot, ya que cada una de estas ofrece características diferentes. Aunque todas permiten la movilidad en suelos planos pavimentados o sobre superficies duras, este no es el único factor importante en una plataforma móvil. Una de las limitaciones más típicas es el deslizamiento en la impulsión, que va relacionado con de las características del terreno, por otra parte, excepto en configuraciones muy especiales, no es posible alterar internamente los puntos de estabilidad para adaptarse al tipo del terreno, lo que limita de forma importante los caminos aceptables del soporte (Ollero, 2001). Por lo cual, es recomendable precisar de manera previa las condiciones del terreno, eficiencia energética, dimensiones, cargas útiles y maniobrabilidad, ya que cada configuración confiere estas propiedades en mayor o menor medida.

2.2.2.1 Estructura Ackerman

Esta configuración es la más utilizada en los vehículos de cuatro ruedas convencionales. Está compuesto por cuatro ruedas; dos ruedas traseras con el mismo eje y dos ruedas delanteras que pueden girar en sincronía para cambiar de dirección.

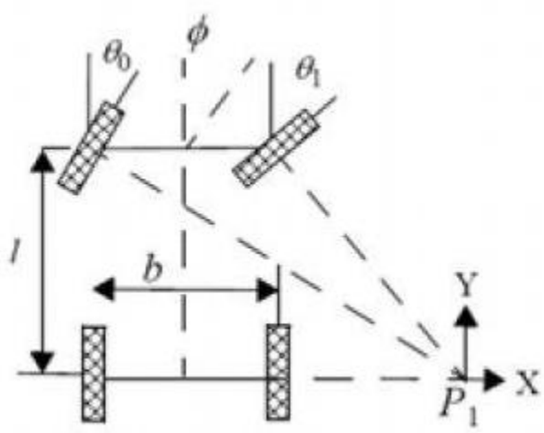


Figura 7. Sistema Ackerman (Ollero, 2001)

La fig. 7 muestra en detalle el sistema de locomoción. La rueda delantera interior por lo general se diseña con un ángulo ligeramente superior a la exterior $(\theta 1 > \theta 0)$ para prescindir el deslizamiento. Por otro lado se debe tener en cuenta las fuerzas centrifugas al momento del diseño de la estructura, en virajes a velocidades relativamente altas, ya que los vectores de velocidad instantánea son tangentes a dichas curvas, lo que puede concurrir en un volcamiento (Ollero, 2001). Dentro de sus ventajas podemos señalar una excelente estabilidad, útil para realizar giros sobre la marcha a cierta velocidad, y sus puntos débiles radican en la maniobrabilidad limitada, difícil posicionamiento y complicado modelamiento cinemático (Microautomacion, 2018).

2.2.2.2 Estructura manejo diferencial

El manejo o direccionamiento diferencial es uno de los diseños más populares, compuesto por dos ruedas fijas situadas una a cada lado de la plataforma, y una rueda castor para soporte, *fig. 8*. Debido a la diferencia de velocidades se obtiene el direccionamiento del robot, he igualado las velocidades se da el torque máximo, siendo a su vez posible incorporar dos o más ruedas tipo castor para mayor estabilidad a la plataforma robótica (Dudek & Jenkin, 2016).

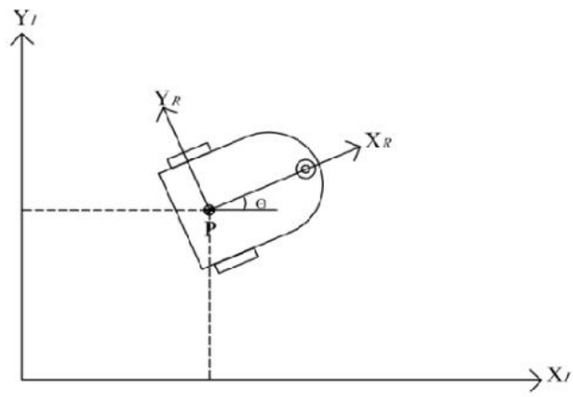


Figura 8. Direccionamiento diferencial (Malu & Majumdar, 2014)

Las ventajas más destacadas de esta estructura son: Estructura mecánica y modelo cinemático simple, bajos costes de fabricación y por otra parte sus desventajas son: dificultad al moverse en superficies irregulares, donde pueden presentarse movimientos abruptos si una llanta actuador pierde el contacto con el suelo (Dudek & Jenkin, 2016).

2.2.2.3 Estructura triciclo clásico

En esta estructura la rueda delantera sirve tanto para la tracción como para el direccionamiento. *Fig. 9.*

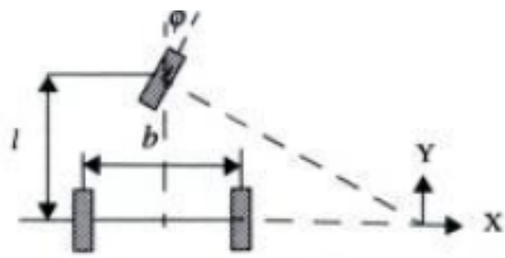


Figura 9. Triciclo clásico (Ollero, 2001)

Las ruedas del eje trasero son pasivas, la maniobrabilidad es mayor que en la estructura Ackerman, pero en terrenos fluctuantes suele presentar sus falencias de estabilidad y perdida de tracción, ya que el centro de gravedad tiende a desplazarse cuando el robot se somete a ligeras inclinaciones (Ollero, 2001). Debido a su simplicidad, es bastante frecuente en vehículos robóticos para interiores y exteriores pavimentados.

2.2.2.4 Estructura Skid Steer

Se disponen varias ruedas en cada lado del vehículo que actúan simultáneamente, siendo capaz de girar su plataforma 180 grados sin ningún radio de giro (Ollero, 2001). Este tipo de plataformas presenta un buen desempeño en diferentes terrenos ya que tiene tracción en todas sus ruedas (“Robot móvil SUMMIT XL HL | Robotnik,” 2017)

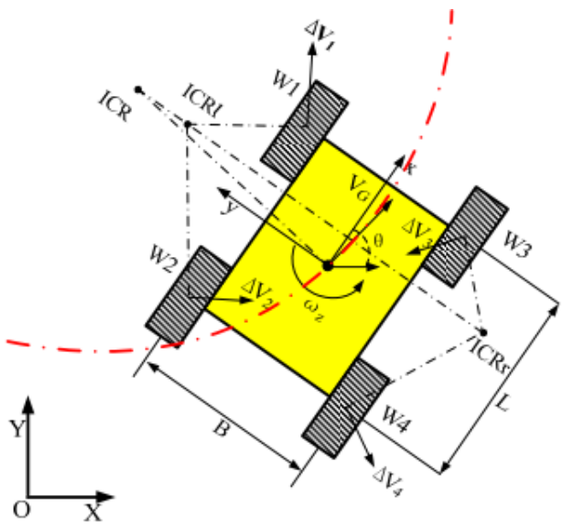


Figura 10. Estructura Skid Steer (Wang et al., 2015)

Sin embargo, es un poco complejo el desarrollo de los modelos cinemáticos que describen el movimiento, ya que se necesita un alto desarrollo de ecuaciones y calculo computacional para el control de movimiento, de igual forma al tener tracción en todas sus ruedas el gasto energético es un factor ineludible (Wang et al., 2015). También este tipo de estructura suele considerarse análoga al direccionamiento con orugas, en este caso la propulsión esta menos limitada por el deslizamiento y la resistencia al desgaste es mucho mayor (Ollero, 2001).

2.2.2.5 Estructura asíncrona

Consiste en la actuación simultanea de todas sus ruedas que giran de forma síncrona. La transmisión se obtiene usualmente mediante coronas de engranajes o con correas concéntricas (Ollero, 2001).

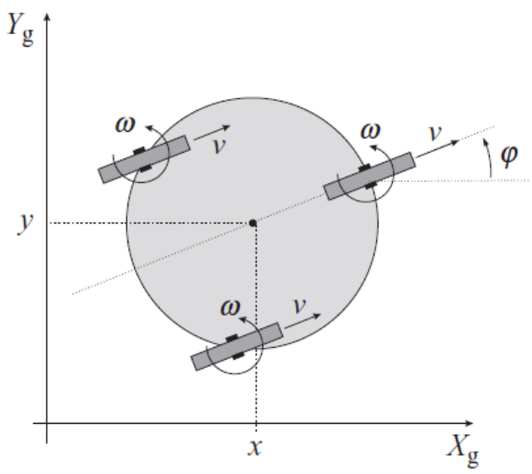


Figura 11. Direccionamiento asíncrono (Klančar et al., 2017)

A través del direccionamiento de las ruedas esta plataforma puede desplazarse en cualquier dirección no cambiando la orientación del chasis, y esto lo puede lograr con dos actuadores, ya que en todos los casos el tercer actuador actúa como soporte. Sus ventajas entonces radican en el movimiento omnidireccional y por

consiguiente, requiere menos esfuerzo para lograr el control de movimiento, también la información odometríca es relativamente precisa y eventualmente ofrece tracción en todas sus ruedas. Por lo que resta, sus grandes desventajas radican en que su estructura mecánica es complicada, que pueden producirse diferencias de velocidad entre las ruedas en caso de que la transmisión presente holguras por desgaste (Dudek & Jenkin, 2016).

2.2.2.6 Estructura omnidireccional con ruedas especiales

En la estructura de tres ruedas se emplean ruedas universales u omni-wheel *fig. 12*, y en estructuras de más de cuatro ruedas, se emplean las ruedas suecas, también conocidas como ruedas mecanum *fig. 13*, en honor a la compañía sueca Mecanum AB donde trabajó su creador Bengt Erland Ilon (*Wheels for a course stable selfpropelling vehicle movable in any desired direction on the ground or some other base*, 1972). Esto con el fin de tener el movimiento omnidireccional en la base del robot, siendo una de sus principales ventajas, sin embargo, el diseño mecánico y su construcción suele ser un poco complejo (Dudek & Jenkin, 2016). Existen plataformas de cuatro ruedas con nuevos diseños, basados en el principio de las ruedas mecanum (Isogawa, 2017).

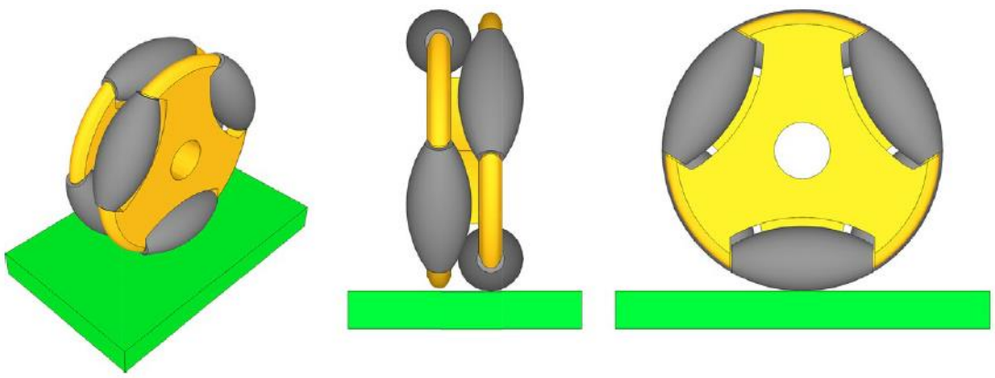


Figura 12. Rueda Universal desde diferentes vistas (Klančar et al., 2017)

Figura 13. Rueda Mecanum (Nexus, 2016)

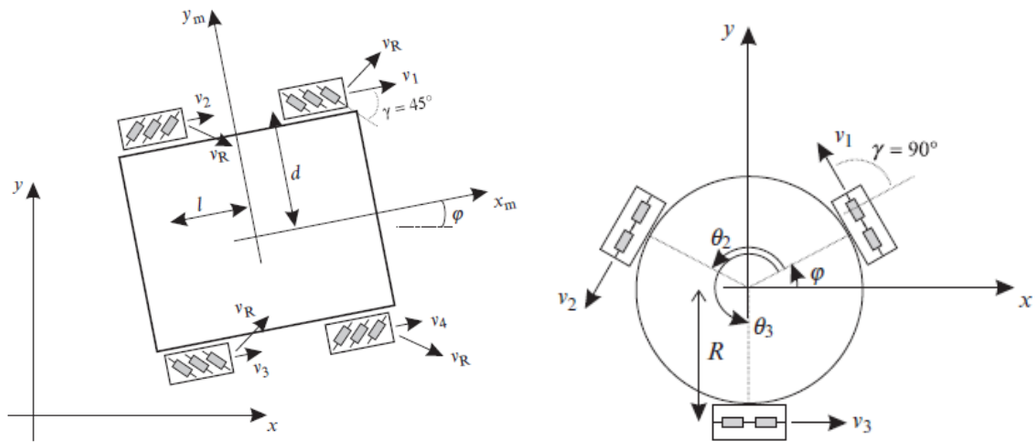


Figura 14. Izquierda: Plataforma de cuatro ruedas, Derecha: plataforma de tres ruedas (Klančar et al., 2017)

Por otra parte, sus desventajas surgen por las vibraciones en los movimientos verticales, debido a los contactos discontinuos de sus ruedas, lo que también se traduce en más ruido en cada desplazamiento, baja durabilidad de los rodillos en comparación con las ruedas convencionales (Dudek & Jenkin, 2016). Las ruedas mecanum tienen pequeños rodillos dispuestos alrededor del borde de la rueda generalmente suelen tener un ángulo de 45 grados y con la combinación de los movimientos (hacia atrás, hacia adelante) se genera la rotación de los pequeños rodillos, proporcionando diferentes direcciones de movimiento; ya en la plataforma de tres ruedas se emplean las ruedas especiales usualmente dispuestas a 120 grados una respecto a la otra, y a diferencia de las mecanum, sus rodillos tiene un ángulo de 90 grados con respecto eje perpendicular de la rueda (Klančar et al., 2017).

2.3 CONTROL DE ROBOTS MOVILES

Siguiendo con el desarrollo de los objetivos del proyecto, se plantea trabajar con una estructura omnidireccional con cuatro ruedas mecanum, debido a que este tipo de estructura reduce la complejidad del sistema de control de movimiento, ya que las aplicaciones de software pueden simplificarse gracias a que solo se controla la dirección del robot en general y no la dirección de ruedas en sí, lo que se traduciría en la eliminación de las restricciones de vehículos no-holonomicos.

El modelado de movimiento puede describir la cinemática del robot, en este caso, se consideran solo los movimientos y no las causas que lo ocasionan, como sería el torque y la fuerza. El modelo cinemático describe (Klančar et al., 2017):

- La relación de la geometría presente en el sistema, así como la relación entre los parámetros de control del comportamiento del sistema dado por la representación tiempo-estado.
- El sistema de velocidades representadas por un conjunto de ecuaciones diferenciales de primer orden.

Por su parte el modelo dinámico describe el sistema de movimiento cuando se aplican fuerzas al sistema. Este modelo incluye la física de los movimientos y se

usan parámetros de fuerzas, energías, masas e inercias del sistema. Este tipo de modelo se representa por medio de ecuaciones diferenciales de segundo orden.

2.3.1 Cinemática de robots móviles

Existen varios tipos de modelos cinemáticos (Klančar et al., 2017);

2.3.1.1 La cinemática interna

Explica la relación entre las variables del sistema interno, como son: rotación de las ruedas y movimiento del robot.

2.3.1.2 La cinemática externa

Describe la posición del robot y la orientación de acuerdo con algunas referencias de coordenadas.

2.3.1.3 Cinemática directa e inversa

La cinemática directa describe el estado del robot en función de sus entradas, tales como velocidad de las ruedas, movimiento de sus articulaciones, dirección de las ruedas, etc. Por otro lado, con la cinemática inversa se puede diseñar o planear el movimiento, lo que quiere decir, que las entradas del robot se calculan para obtener una secuencia de movimiento deseada.

2.3.2 Restricción de movimiento

Surge cuando un sistema tiene menos variables de entrada que grados de libertad. Las restricciones holonomicas prohíben ciertas posiciones del robot mientras que las restricciones no-holonomicas prohíben las direcciones laterales del robot, es decir, que el robot solo se puede mover en las direcciones de la rotación sus ruedas.

El estudio de la cinemática está basado en el uso de ruedas holonomicas, ubicadas en los cuatro extremos del robot y que suelen ser accionadas por servomotores o motores paso a paso (Klančar et al., 2017).

2.3.3 Cinemática del manejo omnidireccional con cuatro ruedas

Una de las configuraciones más populares es el movimiento proporcionado por las cuatro llantas mecanum, ya que permite en cierta medida, tener un rango de estabilidad mucho más elevado en comparación a las estructura de tres llantas universales. La disposición de las ruedas es mostrada en la *fig. 15*.

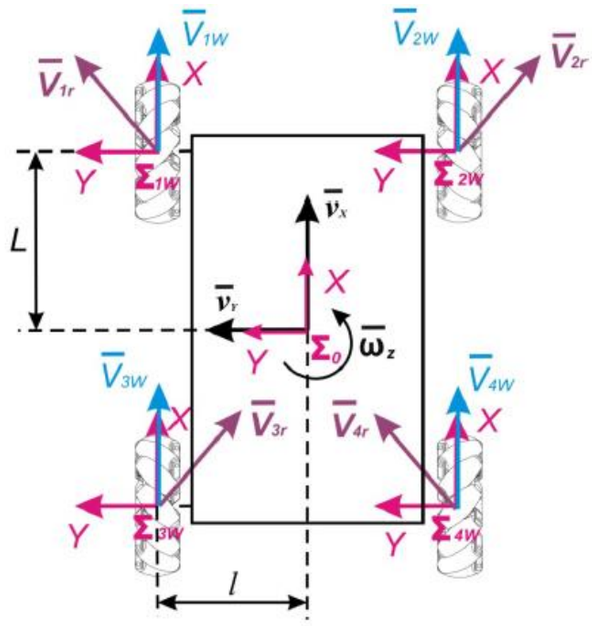


Figura 15. Disposición del variables para el análisis cinemático (Olimpiu, Mândru, Ardelean, & Ple, 2014).

A continuación se describen las variables para las llantas:

$V_{iw}(i = 1,2,3,4)$ - es el vector velocidad correspondiente a la rotación de la rueda.

$V_{ir}(i = 1,2,3,4)$ – es el vector de velocidad tangencial del rodillo que toca la superficie en movimiento.

Donde $w_{iw} = w_{iw}(R_w)$, R_w – es el radio de la rueda omni-direccional, y W_{iw} – es la velocidad angular de la misma (i).

Las siguientes son las variables para el robot:

V_x, V_y – son la velocidad linear de los componentes del robot en los ejes X y Y.

W_z – es la velocidad angular del robot

L – es la distancia de cada eje de la llanta al centro de gravedad del robot en el eje X

l – es la distancia de cada eje de la llanta al centro de gravedad del robot en el eje Y

Habiendo identificado cada uno de las variables anteriores, la cinemática inversa puede ser descrita de la siguiente manera:

$$V_w = J_o \,.\, V_o \qquad (1)$$

Donde $V_w = [V_{1w}\ V_{2w}\ V_{3w}\ V_{4w}]^T \in R^{4x1}$ es el vector de velocidad correspondiente a la velocidad angular y $V_0 = [V_x\ V_y\ W_z]^T \in R^{3x1}$ la velocidad del vector en coordenadas cartesianas.

Entonces la matriz de transformación J_o puede expresarse de la de la siguiente manera.

$$J_o = \begin{bmatrix} 1 & -1 & -(l+L) \\ 1 & & 1+(l+L) \\ 1 & & 1-(l+L) \\ 1 & -1 & +(l+L) \end{bmatrix} \epsilon\, R^{4x1} \quad (2)$$

La velocidad angular de las ruedas es dada por la relación de (1):

$$\begin{pmatrix} w_{1w}(t) \\ w_{2w}(t) \\ w_{3w}(t) \\ w_{4w}(t) \end{pmatrix} = \frac{1}{R_W} \cdot \begin{bmatrix} 1 & -1 & -(L+l) \\ 1 & & 1+(L+l) \\ 1 & & 1-(L+l) \\ 1 & -1 & +(L+l) \end{bmatrix} \cdot \begin{pmatrix} V_x(t) \\ V_y(t) \\ w(t) \end{pmatrix} \quad (3)$$

Para determinar la velocidad del robot, la matriz pseudo inversa J_o está determinada por la ecuación:

$$J_0^+ = (J_0^T . J_0)^{-1} J_0^T = \frac{1}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ -\frac{1}{L+1} & \frac{1}{L+1} & -\frac{1}{L+1} & \frac{1}{L+1} \end{bmatrix} \epsilon\, R^{4x1} \quad (4)$$

Y

$$J_0^+ \cdot J_0 = I_3 \quad (5)$$

Usando la matriz pseudo inversa J_0^+, tenemos:

$$V_0 = J_0^+ \cdot V_w \quad (6)$$

Por lo tanto las velocidades V_x V_y y w_z del robot son:

$$\begin{bmatrix} V_x \\ V_y \\ w_z \end{bmatrix} = J_0^+ \begin{bmatrix} V_{1w} \\ V_{2w} \\ V_{3w} \\ V_{4w} \end{bmatrix} \quad (7)$$

O

$$\begin{bmatrix} V_x \\ V_y \\ w_z \end{bmatrix} = \frac{R_w}{4} \begin{bmatrix} 1 & 1 & 1 & 1 \\ -1 & 1 & 1 & -1 \\ -\frac{1}{L+1} & \frac{1}{L+1} & -\frac{1}{L+1} & \frac{1}{L+1} \end{bmatrix} \cdot \begin{bmatrix} w_{1w}(t) \\ w_{2w}(t) \\ w_{3w}(t) \\ w_{4w}(t) \end{bmatrix} \quad (8)$$

De la ecuación (7) tenemos:

$$V_x = \frac{R_w}{4} (\dot{\theta}_{1w} + \dot{\theta}_{2w} + \dot{\theta}_{3w} + \dot{\theta}_{4w}) \quad (9)$$

$$V_y = \frac{R_w}{4} (-\dot{\theta}_{1w} + \dot{\theta}_{2w} + \dot{\theta}_{3w} - \dot{\theta}_{4w}) \quad (10)$$

$$w_z = \frac{R_w}{4(L+l)}(-\dot{\theta}_{1w} + \dot{\theta}_{2w} - \dot{\theta}_{3w} + \dot{\theta}_{4w}) \quad (11)$$

Donde

$$\dot{\theta}_{iw} = w_{iw} \quad (i = 1,2,3,4) \quad (12)$$

El robot tiene 3GDL, solo tres velocidades $(V_{x,}\, V_y, w_z)$ de las cuatro ruedas pueden ser asignadas independientemente $(w_{1w,}\, w_{2w,}\, w_{3w,}\, w_{4w}\,)$.

La dirección del movimiento resultante α, es definido por:

$$\alpha = \operatorname{atan}\left(\frac{V_y}{V_x}\right) \quad (13)$$

Y la velocidad resultante con:

$$V_r = \sqrt{(V_x^2 + V_y^2)} \quad (14)$$

Usando las ecuaciones dadas anteriormente podemos encontrar la posición, dirección de movimiento (13) y la velocidad resultante (14) del robot omni direccional con llantas mecanum (Olimpiu et al., 2014).

2.3.4 Teleoperación y telerobótica

Antes que nada, se citaran los siguientes términos que usualmente suelen generar ciertas confusiones (Trubin, 2013):

Telepresencia: Se refiere al conjunto de tecnologías que permiten a una persona experimentar un efecto o apariencia como si estuviera presente en un lugar que no es su verdadera ubicación.

Teleoperación: Es la manipulación de una máquina a distancia.

Telerobótica: Es el área de la robótica encargada del control de robots a distancia, principalmente usando conexiones inalámbricas. Es la combinación de dos subcampos principales: Teleoperación y telepresencia.

Lo que quiere decir, que cuando se hace referencia a la teleoperación, se esta hablando de un objeto teleoperado que no es necesariamente un robot, ni tampoco

transmite al operador la sensación que se vive en el entorno remoto de forma tangible.

Otros autores describen la telerobótica con el control remoto de un humano a un robot y la teleoperación más enfocada con las operaciones a nivel de tarea (Dudek & Jenkin, 2016)

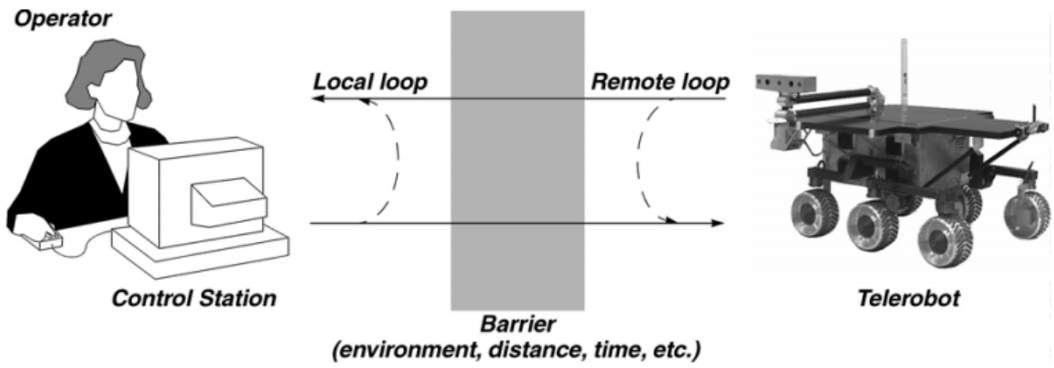


Figura 16. Teleoperación de un robot (Fong & Thorpe, 2001)

A principios de los 1900 se dice que surgió el término teleoperación siendo de muy poco renombre en ese entonces, pero ya en los años 70 su conocimiento era bastante global. Hoy por hoy, y a través de múltiples trabajos y desarrollos realizados en este campo, la teleoperación se ha extendido a diferentes entornos y ambientes como los ROV, RPV, UAV, UGV por sus siglas en inglés (Fong & Thorpe, 2001).

Vehículos operados remotamente (ROV)
Vehículos pilotados remotamente (RPV)
Vehículos aéreos no tripulados (UAV)
Vehículos terrestres no tripulados (UGV)

En términos generales, un sistema teleoperado se basa es un operador desde una estación de control que genera comandos y recibe algún tipo de retroalimentación de un entorno distante; el vehículo remoto ejecuta los comandos, a menudo usando cierto nivel de autonomía (Fong & Thorpe, 2001). Por lo que se dice que existe una separación física entre el operador y el operado, pero no necesariamente en todos los casos esta separación es grande ya que incluso suele darse en un mismo recinto.

Estos sistemas se dividen en dos partes: El sitio local (maestro), que engloba al operador y todos los elementos necesarios para implementar el sistema de conexión con el usuario, que puede ser un joystick, pantalla, teclado u otro tipo de dispositivo de entrada y salida. El sitio remoto (esclavo) que engloba al robot, sensores, elementos de control y en entorno que será manipulado

Un aspecto a tener en cuenta, son los diferentes tipos de arquitectura de control usados en la telerobótica (Dudek & Jenkin, 2016), fig. 17:

2.3.5 Control directo o control manual:

Podemos decir que está en un extremo del control, ya que aquí el usuario está controlando el movimiento del robot sin ningún tipo de ayuda automatizada. Con este tipo de control se puede lograr que el operador sienta que está en el lugar remoto, por lo que es imprescindible contar con un gran ancho de banda. La primacía radica en un retardo casi nulo, pero desafortunadamente contar con un gran ancho de banda a un costo accesible es algo apremiante y es en lo que se trabaja en diferentes investigaciones (Nu, 2014).

2.3.6 Control supervisado:

Involucra el otro extremo del control e implica que los comandos y retroalimentación del usuario acurren en un nivel muy alto y que el robot requiere un mayor grado de autonomía o inteligencia para cumplir su función.

Las interfaces de control también están diseñadas para altos niveles de generación de comandos, monitoreo y diagnóstico, siendo adecuadas para aplicaciones que involucran bajo ancho de banda o grandes retrasos en la comunicación (Fong & Thorpe, 2001). *Fig. 16.*

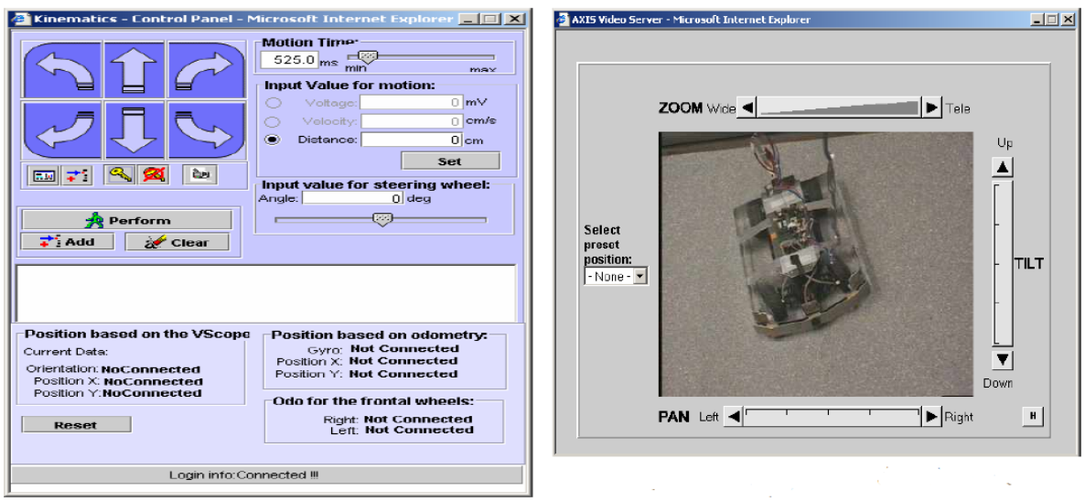


Figura 17. Interfaz de un sistema de control supervisado (Nu, 2014)

2.3.7 Control compartido

En este tipo de estructura de control se necesita algún grado de autonomía o ayuda automatizada disponible que asista al usuario.

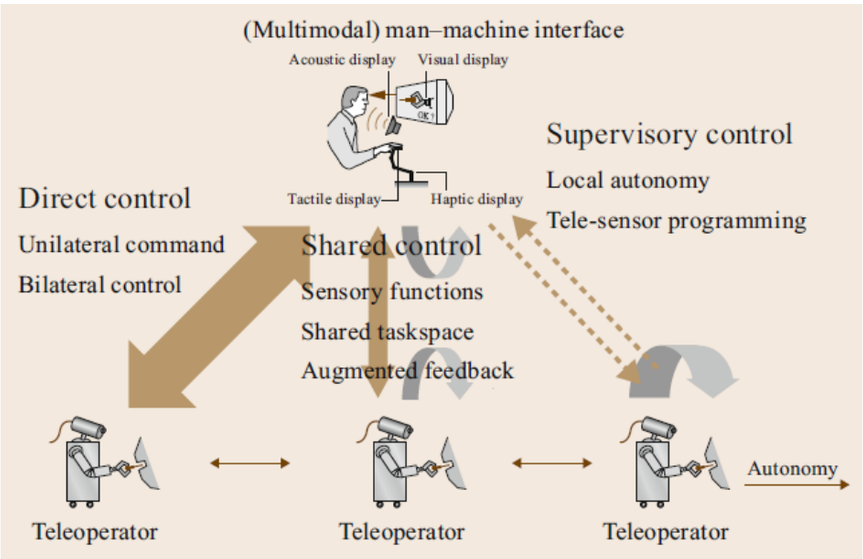


Figura 18. Arquitecturas de control teleoperado (Dudek & Jenkin, 2016)

En la práctica, muchos sistemas involucran al menos algún nivel de control directo, y aceptan los comandos de movimiento vía joystick o de algún otro dispositivo similar en la interfaz del usuario. El joystick es un instrumento mecánico y puede ser visto de cierta forma como el robot en sí. El entorno local y el remoto son llamados como maestro - esclavo respectivamente o sistema maestro - esclavo. Para proveer el control directo, el robot esclavo es programado para seguir el movimiento del robot maestro, el cual es posicionado por el usuario, siendo común el joystick como maestro para ser una réplica cinemática del esclavo, proveyendo una interfaz intuitiva (Dudek & Jenkin, 2016).

Algunos sistemas maestro-esclavo proveen retroalimentación de fuerzas, es decir, el robot maestro no solo da órdenes, sino que también, percibe movimientos del entorno remoto que pueden ser sentidos de la misma manera por el usuario. De esta forma la interfaz usuario se vuelve completamente bidireccional y se le conoce como sistema telerobótico bilateral (Dudek & Jenkin, 2016)

2.3.8 Esquemas de control bilateral

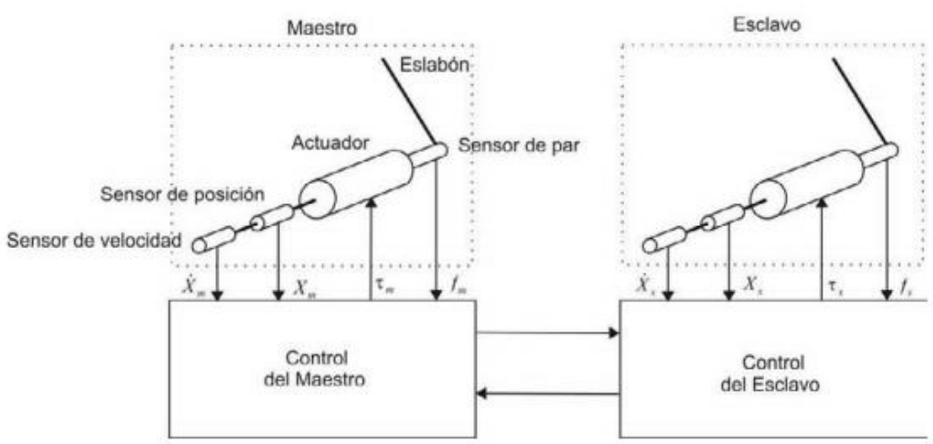


Figura 19. Esquema general de control bilateral, con 1 GDL (Bogado Torres, 2007)

2.3.8.1 Arquitectura posición – posición

El robot maestro y esclavo están instruidos para seguirse mutuamente, implementando a cada lado, generalmente un controlador que los ayude a cumplir su tarea (Dudek & Jenkin, 2016). Es el esquema más sencillo de implementar dado que la acción de fuerzas se basa en las diferencias entre las posiciones del maestro y del esclavo por lo que no es necesario hacer uso de sensores de fuerzas (Bogado Torres, 2007).

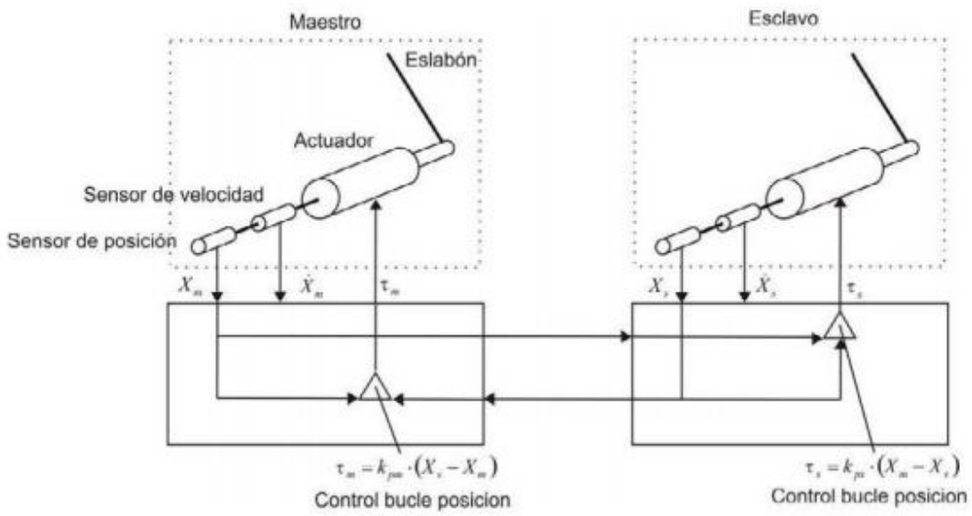


Figura 20. Esquema de posición - posición (Bogado Torres, 2007)

2.3.8.2 Arquitectura posición – fuerza

En esta arquitectura al igual que en el de posición-posición, existe un control de posición en el esclavo, que toma como referencia la lectura de posición del maestro. Pero su diferencia reside en la implementación de un sensor que sirva para obtener la fuerza/par que aplica el esclavo sobre su entorno o viceversa.

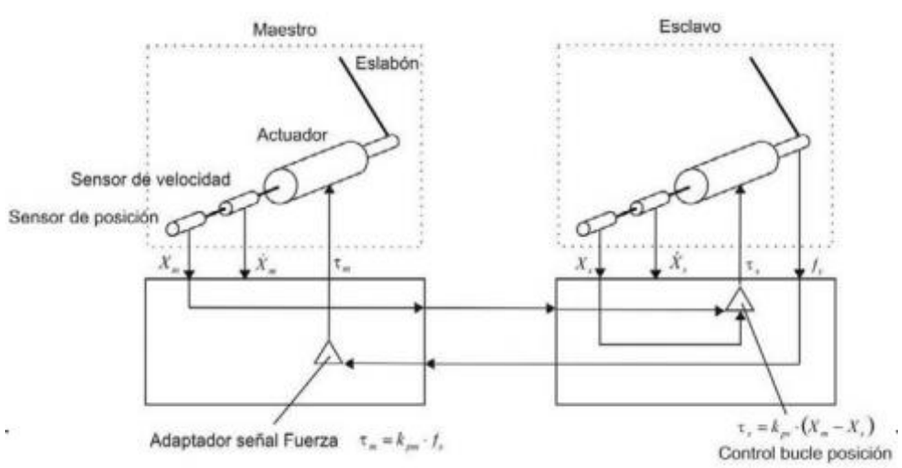


Figura 21. Esquema fuerza - posición (Bogado Torres, 2007)

La diferencia entre las dos arquitecturas es que la primera es más estable, pero no se puede ser consciente de la fuerza que se le aplica al esclavo, aspectos que son inversos en la segunda arquitectura.

2.4 HÁPTICA

La háptica es la ciencia encargada de experimentar y crear sensaciones táctiles en operadores humanos, para mejorar su desempeño en ambientes simulados o teleoperados. Las interfaces hápticas se enfocan en replicar o mejorar la experiencia de manipulación o percepción real de un entorno remoto, a través de dispositivos mecatrónicos y control computarizado (Dudek & Jenkin, 2016). Se dice que la háptica fue inicialmente motivada por la telerobótica, ya que los usuarios querían sentir interacciones con objetos remotos (Kuchenbecker, Fiene, & Niemeyer, 2006).

Básicamente, el dispositivo háptico lee la entrada del operador, la cual puede ser posición, fuerza, actividad muscular, etc. Dichas entradas son aplicadas en un ambiente teleoperado o virtual; en este último, las fuerzas aplicadas al objeto virtual se muestran al operador a través de modelos y algoritmos de renderizado háptico, lo que hace experimentar como si estuviera tocando un objeto real. En ambientes teleoperados el manipulador actúa sobre su ambiente real y la información háptica que se retransmitirá al operador se registra o se estima para que finalmente los actuadores del dispositivo háptico transmitan la sensación táctil al operador humano.

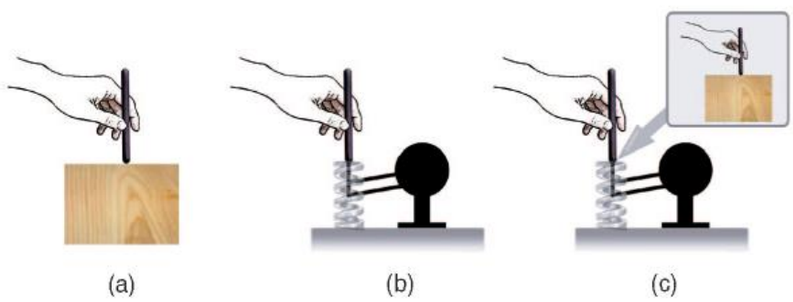


Figura 22. Representación gráfica en (a) del entorno virtual. (b) y (c) la fuerza ejercida por el dispositivo háptico (Kuchenbecker et al., 2006)

Por simple que esto parezca, uno de los grandes desafíos en la generación de sensaciones hápticas artificiales, es que los movimientos del operador no deben restringirse cuando no haya contacto con objetos virtuales o remotos, ya que el dispositivo háptico debe permitir que el operador humano realice movimientos deseados en cualquier dirección, por lo que en teoría se requiere que el dispositivo háptico tenga varios grados de libertad (Dudek & Jenkin, 2016)

Gracias al incesante incremento de la tecnología en el campo de la háptica de la mano con la robótica, así como se describe en apartado del estado del arte, cada vez más se abren nuevos caminos de investigación como en la medicina, entretenimiento, rehabilitación y hasta en actividades militares (Marcano Gamero, 2008).

2.4.1 Dispositivos hápticos

Dentro de este campo, existen dos tipos, los cuales se describirán a groso modo a continuación (Dudek & Jenkin, 2016):

2.4.1.1 Dispositivos de admitancia

Censan la fuerza aplicada por el operador y limitan su movimiento de acuerdo al objeto simulado en el entorno virtual. Lo que quiere decir que incorpora sensores activados por velocidad y sus movimientos tienden a ser relativamente más rígidos y con alta inercia. Los dispositivos o robots con admitancia son más utilizados en ambientes industriales.

2.4.1.2 Dispositivos de impedancia

Estos por el contrario censan la posición del operador y posteriormente aplican una fuerza al operador de acuerdo con la superficie simulada. Por lo general los dispositivos o robots de impedancia tienen baja fricción e inercia en sus movimientos e implementan actuadores activados por fuerza.

Estos dos tipos de arquitecturas enmarcan de forma drástica el diseño del software y hardware de los sistemas robóticos con realimentación háptica. Aunque hoy en día la mayoría de dispositivos hápticos implementados son de tipo impedancia, de la misma forma en que hoy en día los espacios de trabajo limitados son muy usuales. (Dudek & Jenkin, 2016).

2.4.2 Renderizado háptico y gráfico

El renderizado gráfico es el responsable de que el usuario pueda mover una cámara dentro de mundo virtual para que las simulaciones virtuales sean más realistas (Andreu, Torronteras, & Mora, 2015). En caso de que el entorno remoto sea real, este parte pasa a un segundo plano.

El renderizado háptico se define como el proceso de cómputo de fuerzas necesarias para generar una interacción lo más realista posible con el objeto virtual, de acuerdo con la medida de los movimientos del operador (Dudek & Jenkin, 2016).

Esto se lleva a cabo gracias a la integración de tres algoritmos o módulos de control entrelazados entre sí internamente fig.22 para dar como resultado un efecto lo más real posible. A continuación se describen a groso modo: (Andreu et al., 2015):

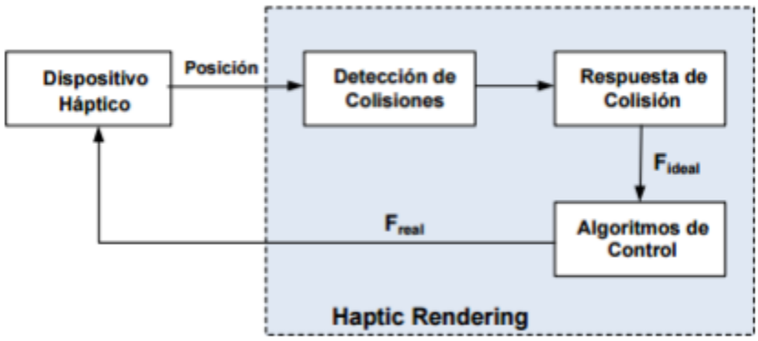


Figura 23. Algoritmos del renderizado háptico (Andreu et al., 2015)

2.4.2.1 Detección de colisiones

Esta principalmente encargado de la recopilación de toda la información del entorno, para que de acuerdo a la posición del dispositivo háptico se pueda determinar si existe o no colisión.

2.4.2.2 Cálculo de la respuesta de colisión

Se encarga de tomar la información del algoritmo de colisiones, y calcular la fuerza ideal de interacción entre el objeto virtual y su entorno.

2.4.2.3 Módulo de control

Por último esta parte devuelve al usuario, a través del dispositivo háptico y basado en sus características motrices, una respuesta real lo más parecida posible a la fuerza ideal calculada en la respuesta de colisión.

2.5 PROTOCOLOS DE COMUNICACIÓN EN SISTEMAS EMBEBIDOS

Los protocolos de comunicación, son un conjunto de reglas que permite la comunicación de datos entre dos o más sistemas electrónicos a través de algún medio físico. Tanto las reglas, sincronización, sintaxis como la semántica empleadas en la transmisión están definidas por el protocolo en cuestión ("Electronic Communication Protocols Basics and Types with Functionality," 2017). Existe gran variedad de protocolos de comunicación y cada uno tiene su propia área de aplicación.

Por otra parte, un sistema embebido, es un sistema electrónico compuesto por hardware y software, donde un controlador toma variables físicas del mundo real a través de sensores/actuadores, y proporciona una o varias acciones de control. En este caso, el sistema puede estar conformado por otros dispositivos electrónicos donde será necesario algún protocolo de comunicación para intercambiar información entre ellos.

En general, los protocolos de comunicación están asociados a una capa física ("Category:Physical layer protocols - Wikipedia," 2018), que describe: El tipo de señal incorporada, fuerza de la señal, establecimiento de comunicación (handshaking mechanism), configuración bus (bus arbitration), direccionamiento del dispositivo, línea de datos, comunicación alámbrica o inalámbrica, selección de la tasa de baud para transmitir y recibir datos, solo por mencionar algunas categorías de las capas de aplicación.

2.5.1 Tipos de protocolos de comunicación en sistemas embebidos

Los protocolos de comunicación están ampliamente clasificados en dos tipos:

- Protocolo de comunicación entre sistemas - *Inter System Communication Protocol.*
- Protocolos de comunicación dentro del sistema - *Intra System Communication Protocol.*

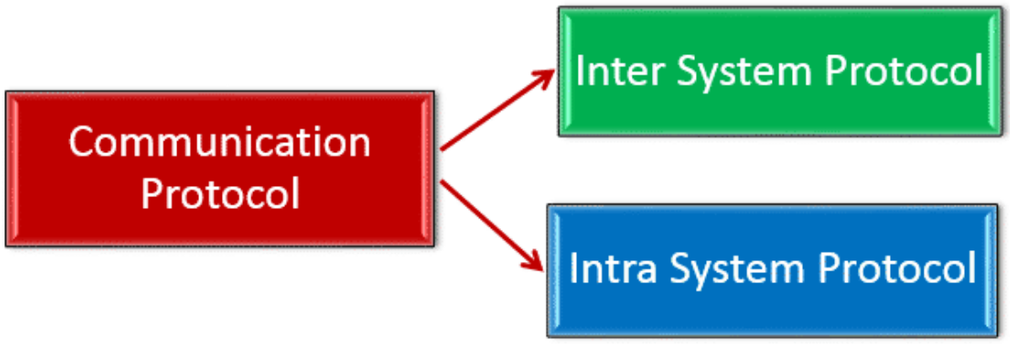


Figura 24. Clasificación de los protocolos de comunicación ("Communication Protocols in Embedded Systems - Types, Advantages & Disadvantages," 2018)

2.5.1.1 Protocolo de comunicación entre sistemas

Los protocolos entre sistemas, establecen la comunicación entre dos dispositivos de comunicación, por ejemplo: entre un PC y un microprocesador o placa de desarrollo, en este caso se logra la comunicación por medio de sistemas de buses seriales o interbus.

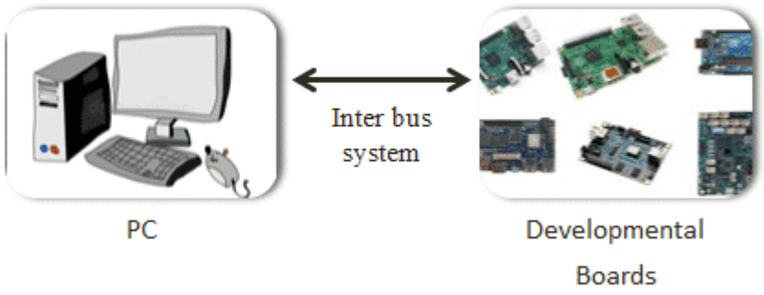


Figura 25. Protocolo de comunicación entre sistemas ("Communication Protocols in Embedded Systems - Types, Advantages & Disadvantages," 2018)

Dentro de esta categoría se encuentran:

- Protocolo de comunicación USB

El protocolo de bus serial universal "Universal Serial Bus – USB", es un protocolo serial de dos cables, el cual permite la conexión en cualquier momento de 127 dispositivos. Este protocolo, envía y recibe datos seriales entre el Host y un periférico externo por medio de dos líneas de datos **D+** y **D-**. Aparte de esas dos líneas tiene el **Vcc** y **GND.**

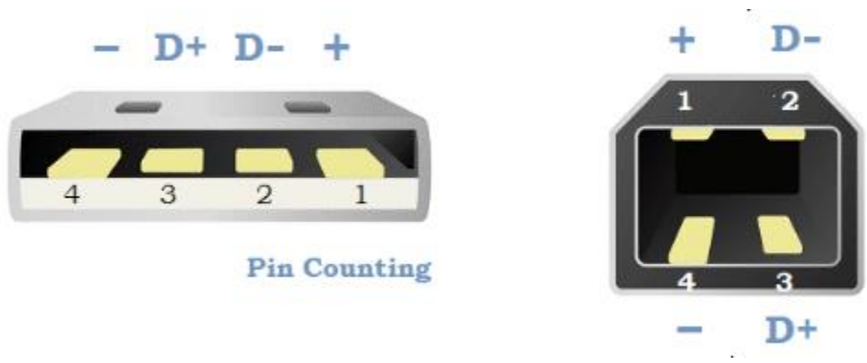


Figura 26. Las 4 líneas y tipos de USB ("Communication Protocols in Embedded Systems - Types, Advantages & Disadvantages," 2018)

Los datos son transmitidos en forma de paquetes, donde dos dispositivos se comunican entre ellos. Los paquetes se componen de 8 bits (byte) siempre transmitiendo primero el bit menos significativo.

Tabla 1. Pros y contras del protocolo USB, ("Communication Protocols in Embedded Systems - Types, Advantages & Disadvantages," 2018)

VENTAJAS	DESVENTAJAS
RÁPIDO Y SIMPLE	Necesita un master potente
BAJO COSTO	Es necesario especificar Driver
HARDWARE DE ENCHUFE Y DESENCHUFE	

- Protocolo de comunicación UART

El transceptor asíncrono universal (Universal Asynchronous Receiver/Transmitter - UART) no se considera como tal un protocolo de comunicación sino una pieza de hardware que convierte datos paralelos a datos seriales. Su función principal es transmitir y recibir datos serialmente.

Esta configuración de envío y recepción de datos se hace por medio de dos líneas o cables – RX y TX, que operan de forma asíncrona, por lo que no asocia ninguna señal de reloj, sino que se tiene un bits de inicio y otro de parada, lo cual define el inicio y fin del paquete de datos.

El UART se comunica de forma tal que solo transmite o recibe datos a la vez, lo que se conoce como comunicación semidúplex. Cuando el receptor final detecta el bit de inicio, empieza a leer los bits de datos a con una tasa especificada de baudios, lo que significa que los periféricos envío y recepción deben operar bajo la misma tasa de baudios.

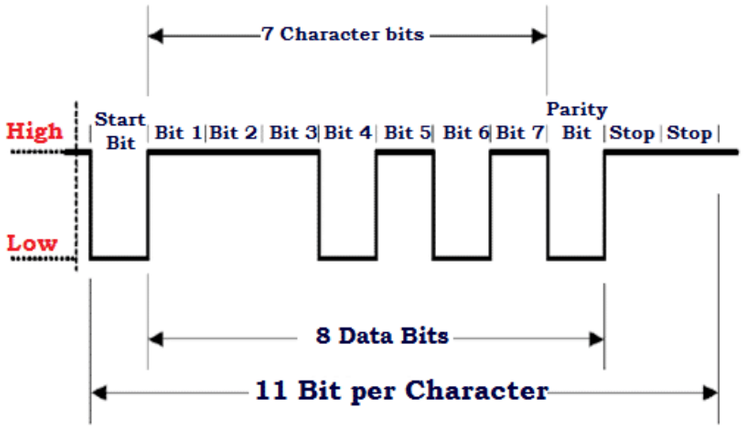


Figura 27. Trama de datos UART, ("Communication Protocols in Embedded Systems - Types, Advantages & Disadvantages," 2018)

- Protocolo de comunicación USART

El transceptor síncrono-asíncrono universal (Universal Synchronous Asynchronous Receiver/Transmitter - USART) es idéntico a la configuración UART, solo que agrega la funcionalidad síncrona. Lo que quiere decir que el transmisor genera una señal de reloj, la cual es captada por el receptor final a través el flujo de datos transmitido, sin la necesidad de sincronización mutua de baud rate o tasa de baudios.

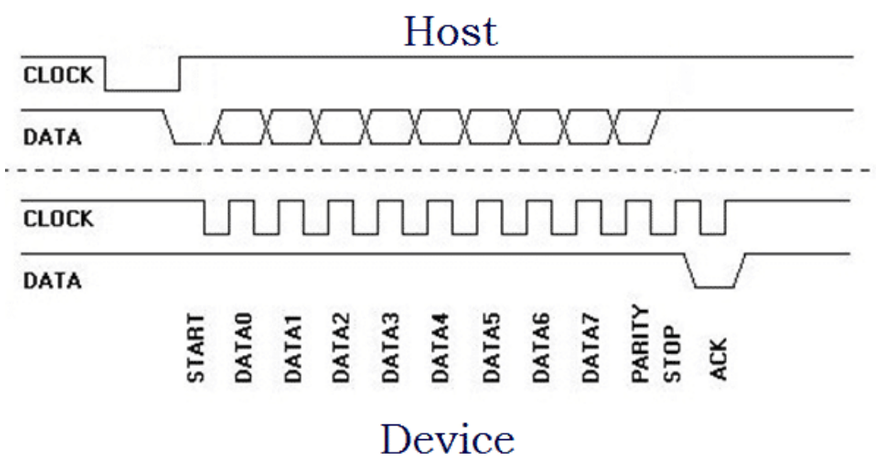


Figura 28. Trama de datos USART, ("Communication Protocols in Embedded Systems - Types, Advantages & Disadvantages," 2018)

De cierta manera, la configuración USART sopesa la habilidad del UART, ya que opera bajo la comunicación full dúplex, lo que permite recibir y transmitir datos al mismo tiempo dependiendo del área de aplicación.

Tabla 2. Pros y contras del UART y USART, (“Communication Protocols in Embedded Systems - Types, Advantages & Disadvantages,” 2018)

VENTAJAS	DESVENTAJAS
NO REQUIERE SEÑAL DE RELOJ	No soporta la funcionalidad de múltiple maestro-esclavo
COSTO – EFECTIVIDAD	La tasa de baudios del UART debe ser el 10% entre transmisor y receptor
UTILIZA BIT DE PARIDAD PARA DETECTAR ERRORES	
REQUIERE 2 LÍNEAS PARA LA COMUNICACIÓN	

2.5.1.2 Protocolos de comunicación dentro del sistema

Los protocolos de comunicación dentro del sistema, establecen la comunicación dentro de una placa de circuito o sistema embebido, incrementado así, el número de componentes electrónicos que se pueden conectan al controlador.

Dentro de esta categoría están:

- Protocolo I2C

El protocolo de comunicación entre circuitos integrados (Inter Integrated Circuit - I2C) desarrollado por Philips Semiconductor, provee facilidad al conectar dispositivos electrónicos con micro controladores. En sistemas embebidos, todos los dispositivos periféricos están conectados al micro controlador como dispositivos mapeados de memoria.

El I2C, precisa de dos líneas para transportar los datos entre dispositivos. La línea para datos seriales (Serial Data Line – SDA) y la línea para la señal de reloj (Serial Clock Line – SCL).

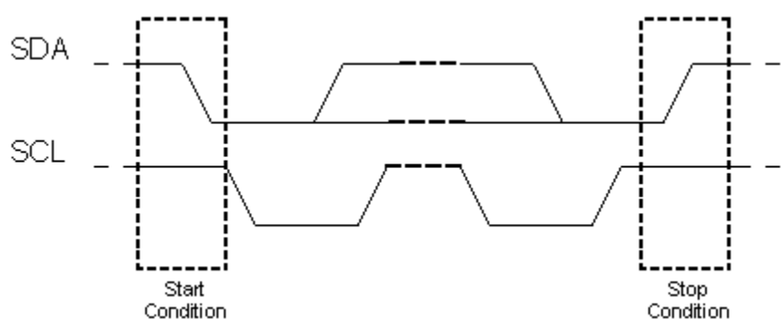


Figura 29. Condición de inicio y parada I2C, (“Electronic Communication Protocols Basics and Types with Functionality,” 2017)

En este protocolo de comunicación maestro-esclavo, se hace necesario proveer una dirección de acceso única a cada esclavo para que se pueda establecer una comunicación. El dispositivo maestro envía la dirección del esclavo al cual se quiere

comunicar, seguido de la opción escribir o leer datos. El esclavo correspondiente pasa a estado activo mientras los demás esclavos permanecen desactivados. Una vez se recibe la señal del maestro – 1 byte de datos, el dispositivo esclavo responderá con un bit de ACK “acknowledge” - que indica que ha sido reconocido. La condición de parada es emitida al final de la comunicación entre los dispositivos.

Tabla 3. Pros y contras del protocolo I2C, (“Communication Protocols in Embedded Systems - Types, Advantages & Disadvantages,” 2018)

VENTAJAS	DESVENTAJAS
PROVEE BUENA COMUNICACIÓN ENTRE DISPOSITIVOS INTEGRADOS A LOS QUE NO SE ACCEDE FRECUENTEMENTE.	Velocidad de transmisión y recepción limitada.
MECANISMO DE DIRECCIONAMIENTO FACILITA LA COMUNICACIÓN MAESTRO-ESCLAVO.	
EL COSTO Y LA COMPLEJIDAD DEL CIRCUITO NO TERMINAN EN LA CANTIDAD DE DISPOSITIVOS	

Tabla 4. Características del protocolo I2C

LÍNEAS	2
VELOCIDAD DE TRANSMISIÓN	
ESTÁNDAR	100 kbps
RÁPIDA	400 kbps
SÚPER RÁPIDA	3.4 Mbps
ULTRA RÁPIDA	5 Mbps
SÍNCRONA O ASÍNCRONA	Síncrona
SERIAL O PARALELA	Serial
NÚMERO MÁX. DE MAESTROS	Ilimitado
NÚMERO MÁX. DE ESCLAVOS	1008

- Protocolo SPI

La interfaz periférica serial (Serial Peripheral Interface - SPI), es uno de los protocolos de comunicación desarrollados por Motorola, el cual opera por medio de 4 líneas: **MOSI** (Master Output Slave Input), **MISO** (Master Input Slave Output), **SS** (Slave Select), y **SCL** (Serial Clock).

El dispositivo maestro configura primero la frecuencia del reloj a una frecuencia determinada, seguidamente, la línea SS es usada para seleccionar el esclavo apropiado, e iniciar la respectiva comunicación de datos de forma bidireccional (full dúplex). Este tipo de comunicación no se limita a trabajar con palabras de ocho bits.

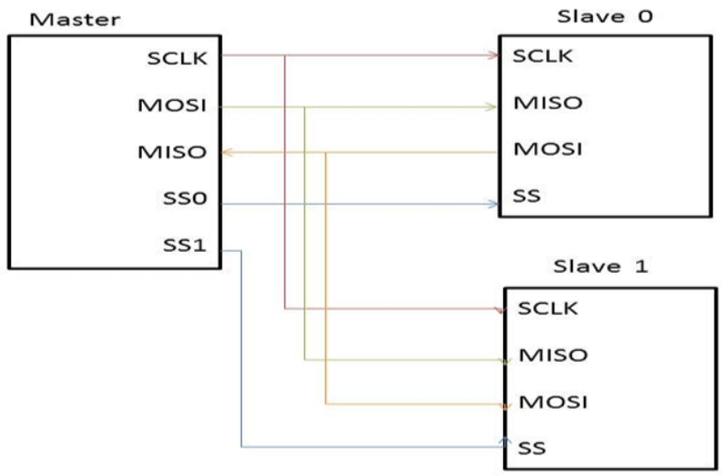


Figura 30. Representación cableada SPI para 1 maestro 2 esclavos, (“Communication Protocols in Embedded Systems - Types, Advantages & Disadvantages,” 2018)

VENTAJAS	DESVENTAJAS
VELOCIDAD DE TRANSMISIÓN RÁPIDA	Requieres más líneas de transmisión
PUEDE CONECTAR MÚLTIPLES ESCLAVOS	No se puede comunicar esclavo-esclavo
BAJO COSTO	

Figura 31. Pros y contras del protocolo SPI, (“Communication Protocols in Embedded Systems - Types, Advantages & Disadvantages,” 2018)

Por otra parte existen los sistemas de comunicación de sistemas embebidos a través de la red, que utilizan como base el modelo OSI (de las siglas en inglés: Open Systems Interconnection, o sea, “Interconexión de Sistemas Abiertos”) el cual estandariza todo aquello relacionado con la comunicación y transmisión de datos en internet. En otras palabras, el modelo OSI es la representación de la comunicación entre dispositivos informáticos.

Por tal motivo se hace necesario explicar cómo funciona el protocolo UDP que fue el utilizado para la transmisión de los datos de los sensores de la plataforma robótica hasta VREP y viceversa.

2.2 Protocolo UDP

Modelo OSI resumido:

7. Capa aplicación: es el medio por el cual se interactúa con el modelo OSI – YT, whatsapp, buscador, etc.

6. Capa presentación: identifica que tipo de formato son los datos: texto, video, imagen, audio, etc. y los estampa con 0 y 1

5. Capa de sesión: Inicia la conversación entre servidor y cliente (da el visto bueno)

4. Capa de transporte: segmenta el mensaje y etiqueta si es TCP o UDP

3. Capa de red: se encarga del direccionamiento y mejor ruta del envío a través de internet (conjunto de *routers* que están interconectados a nivel mundial)

2. Capa de enlace de datos: capa de acceso a los medios (tarjeta de red)

1. Capa física: Transmisión binaria.

CAPITULO III

3. DESCRIPCION DE LOS SENSORES Y ACTUADORES

Partiendo del objetivo principal del proyecto que es lograr que la plataforma robótica móvil sea capaz de evadir obstáculos dentro de un entorno previamente estructurado, de la misma forma que el joystick de mando retroalimente al usuario con fuerzas repulsivas inversamente proporcionales a la distancia de los obstáculos, esto con el propósito de mejorar la teleoperabilidad, desempeño y confort del sistema. Es por ello que la elección de la sensórica juega un papel determinante tanto en aspectos de diseño como de programación.

Generalmente para elegir el sensor de proximidad se tienen en cuenta aspectos y características como rangos de medición, entorno de trabajo, robustez, presupuesto, entre otros. Por su parte, se pueden aprovechar estudios de desempeño realizados previamente en proyectos relacionados, este último fue un factor determinante para decidir por un sensor en especial.

Dentro de las muchas posibilidades en el mercado a la hora de elegir el sensor, inicialmente se pensó en el LIDAR (“RPLIDAR-A1 360°Laser Range Scanner _ Domestic Laser Range Scanner|SLAMTEC,” 2018), el cual, es un sensor bastante robusto e ideal para aplicaciones de detección de movimiento y mapeado, solo por mencionar algunas características, pero se descartó inicialmente por su elevado costo y debido a su gran resolución de toma de datos precisaría de una placa de control mucho más potente. El segundo fue un arreglo de 8 sensores ultrasónicos HC-SR04 dispuestos estratégicamente a los extremos del robot, de tal forma que se pudiera abarcar el rango de medida de 360°, ésta, a pesar de ser una opción mucho más asequible, se descartó debido a que el cableado de los sensores es propenso a inducir ruido en medida, terminando en complicaciones en los datos capturados; además del uso considerado de pines digitales. Finalmente se revisó el VL53L0X que es un sensor bastante novedoso e ideal en diversas aplicaciones de detección de obstáculos por su tamaño y por su tecnología de tiempo de vuelo (TOF) para la toma de datos.

3.1 SENSOR VL53L0X

Este sensor aparte de ser uno de los más pequeños y versátiles en el mercado hoy por hoy, se puede conseguir por menos de 30.000 COP dependiendo del fabricante. Lo que lo hace especial, es que incorpora tecnología de medición por *tiempo de vuelo,* o (TOF) por sus siglas en inglés, su rango de precisión es del $\pm 3\%$, el cual no se ve afectado por objetos reflectantes a distancias enmarcadas entre 5cm y 120cm. El chip VCSEL que incorpora produce una luz láser de 940nm de longitud de onda y utiliza el protocolo de comunicación I2C para enviar datos al dispositivo de control. A diferencia del sensor ultrasónico que basa su principio de funcionamiento en la velocidad de las ondas de sonidos, el sensor VL53L0X opera de acuerdo a la velocidad de la luz, lo que lo hace más eficaz.

Su configuración hardware posee una API que permite configurar el sensor en 3 modos diferentes de operación: *medida única*, donde la medida se hace solo una vez, *medida continua*, donde la medida se hace una consecutivamente después de la otra sin parar, *medida con tiempo*, en el cual el sensor funciona tomando medidas consecutivamente solo con la diferencia que espera un delay o retraso antes de hacer otra medida. (Microelectronics, 2018).

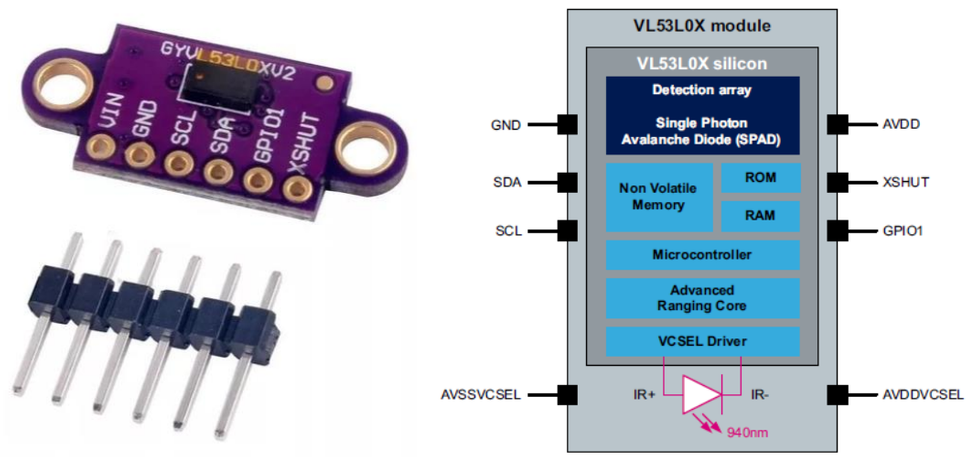


Figura 32. SENSOR VL53L0X y disposición de los pines

Tabla 5. Descripción de pines del VL53L0X

Nombre de la señal	Tipo de señal	Descripción de la señal
AVDDVESEL	Alimentación	Alimentación del VCSEL, se conecta a la alimentación principal
AVSSVCSEL	Tierra	Tierra del VCSEL, se conecta a la tierra principal
GND	Tierra	Se conecta a la tierra principal
XSHUT	Entrada digital	Pin XSHUT, activo en bajo
GPIO 1	Salida digital	Salida de interrupción. Salida de drenaje abierto
SDA	Salida/entrada digital	Datos en serie I2C
SCL	Entrada digital	Entrada reloj serie I2C
AVDD	Alimentación	se conecta a la alimentación principal

De acuerdo a las especificaciones del proyecto, la medición de distancia de los obstáculos se hará en un rango de 360 grados aproximadamente, con la disposición de dos sensores laser VL53L0X ubicados a 180 grados uno con respecto al otro, para posteriormente hacer un barrido de 180 grados con la ayuda de un servomotor de recorrido rápido. La distancia de detección de obstáculos máxima será acotada a solo 400mm. A partir de allí el dispositivo inicia la retroalimentación de fuerzas al operador, con el fin de trabajar en un rango mucho más cercano al momento de colisión y ofrecer así una retroalimentación háptica mucho más holgada, sin sentir que se está a punto de colisionar aun cuando el obstáculo está relativamente lejos.

Tabla 6. Especificación técnica del VL53L0X

Características	Detalle
Paquete	LGA 12 Óptico
Tamaño	4,40 x 2,40 x 1.00 mm
Voltaje de operación	2,6 a 3,5 V
Temperatura	-20 a 70°C
Emisor infrarrojo	940nm
I2C	Hasta 400 KHz (modo rápido) bus serial, dirección: 0x52

La comunicación entre los sensores y el dispositivo de control o maestro, se lleva a cabo mediante los pines SCL y SDA. El pin XSHUT de cada sensor se encarga de ponerlos en modo standby o en interrupción una vez que no se está realizando toma de datos. Para nuestro caso, no se utilizará el pin GPIO del sensor para reset, por lo que el fabricante recomienda dejarlo desconectado.

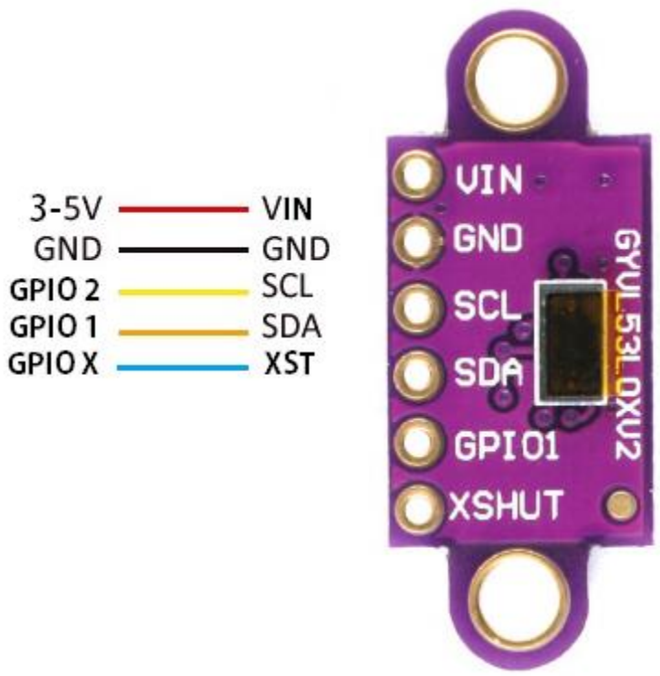


Figura 33. Pines del GYVL53L0X

Por otro lado, el fabricante también recomienda conectar la líneas de datos y de reloj a la fuente de alimentación por medio de resistencias *pull ups* con valores entre 1.5 a 2kohms. Esto para definir el voltaje AVDD (voltaje interno del chip VCSEL) a 2.8V y una velocidad de reloj óptima para I2C de 400khz, (Microelectronics, 2018). Esto también para que cuando no se esté transmitiendo datos, ambas señales estén en alto.

3.1.1 Descripción funcional del sistema

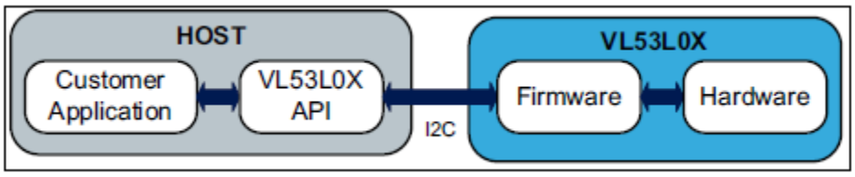


Figura 34. Descripción funcional del sistema del VL53L0X

La API está a disposición del usuario, a través de un conjunto de funciones de alto nivel que permite el control del firmware del VL53L0X, tal como la inicialización/calibración, inicio/paro de medida, elección de precisión, elección del modo de medida; dicho de otro modo, la API permite al usuario beneficiarse por completo de todas las capacidades del VL53L0X sin las complicaciones del acceso directo a múltiples registros.

El protocolo I2C permitirá acceder al firmware del sensor, *figura 38*, aprovechando las opciones que brinda la API. Para usar el bus I2C en la ESP8266, el IDE Standard proporciona la librería "Wire.h", que contiene las funciones necesarias para controlar el hardware integrado. ("I2C | Aprendiendo Arduino," 2019). Dicha librería actualmente soporta el modo maestro hasta aproximadamente 450 KHz - Lo que es justo el rango de velocidad máxima a la que transmite datos el sensor. Antes de utilizar I2C, es necesario establecer los pines SDA, SCL, llamando la función *Wire.begin(int sda, int scl), p.ej. Wire.begin(0, 2)* en ESP-01, de lo contrario, los pines por defecto son 4(SDA) y 5(SCL). ("Librerías — documentación de ESP8266 Arduino Core - 2.4.0," 2019)

Este sensor cuenta con 4 perfiles de medición, configurables a través de código. Cada uno de estos perfiles tiene ciertas características de trabajo, como el tiempo óptimo para validación de toma de datos, el rango máximo típico de operación y aplicaciones.

	Timing budget	Typical max range	Typical application
Default mode	30ms	1.2m (white target)	standard
High accuracy	200ms	1.2m (white target)	precise measurement
Long range	33ms	2m (white target)	long ranging, only for dark conditions
High Speed	20ms	1.2m (white target)	high speed where accuracy is not priority

Figura 35. Perfiles de los rangos de medición de la API (Microelectronics, 2018)

Para cada perfil de medición, se recomienda predefinir los rangos de operación antes de la medición. Esto con el fin de calibrar y mejorar el desempeño del perfil elegido (STMicroelectronics, 2016), en nuestro caso, se utilizará el modo default, ya que la distancia a medir es corta y el error de la medida en el tiempo es aceptable.

Por otra parte, se quiso revisar que tanto influye el color y el tipo de luz a la hora de tomar las distancias con respecto a un objeto. Debido al gran alcance de medición del sensor, el porcentaje de precisión puede variar. En las tablas 3 – 5 se visualiza el porcentaje de error en la medida con objetivos reflectantes a diferentes rangos de alcance del sensor, usando además 3 variedades de colores (Nicolau, 2018), por lo que se sabe que el color del objeto incide significativamente por encima o por debajo al error promedio de $\pm 3\%$ según las especificaciones del fabricante.

Tabla 7. Precisión del sensor VL53L0X usando Luz artificial (Nicolau, 2018).

Distancia	Rosado	Negro	Blanco
100mm	±3%	±3%	±3%
500mm	±3%	±3.6%	±1.8%
800mm	±0.6%	±1.25%	±0.8%
1000mm	±2.6%	±3.7%	±2.2%
1200mm	±2.8%	±5.6%	±3.4%
1400mm	±4.5%	±12.1%	±3.1%
1500mm	±3.8%	±11.5%	±4.1%

Tabla 8. Precisión del sensor VL53L0X en la oscuridad (Nicolau, 2018).

Distancia	Rosado	Negro	Blanco
100mm	±2%	±3%	±2%
500mm	±1.4%	±2.2%	±2%
800mm	±1.1%	±1.5%	±1.5%
1000mm	±1.4%	±1.4%	±1.5%
1200mm	±2.6%	±4.6%	±4%
1400mm	±5.6%	±10.6%	±9.6%
1500mm	±8.9%	±10.1%	±9.4%

Tabla 9. Precisión del sensor VL53L0X en luz ambiente (Nicolau, 2018).

Distancia	Rosado	Negro	Blanco
100mm	±3%	±4%	±5%
500mm	±1.8%	±1.8%	±1.8%
800mm	±3.3%	±2.6%	±2.6%
1000mm	±3.5%	±2.9%	±3.3%
1200mm	±7.2%	±8%	±7.8%
1400mm	±13.6%	±17.4%	±13.2%

Por lo que en los obstáculos de color blanco se destaca una medida con mayor desempeño y menor margen de error. El entorno de trabajo del robot estará principalmente iluminado con lámparas alógenas y la toma de datos con respecto a los obstáculos inicia a partir de los primeros 400mm, de esta manera se conoce que el margen error promedio para ese rango es de +-1.8% y se puede confiar que los datos son bastante precisos.

3.2 SERVO MOTOR DYNAMIXEL AX-12A

Este es un servo motor de la serie Dynamixel, con una precisión bastante confiable, y una circuitería de control que le permite operar en una red de hasta 254 servomotores Dynamixel. A pesar de sus dimensiones, puede producir un gran torque ya que está fabricado con materiales de alta calidad para proveer la fuerza necesaria y resiliencia estructural para resistir grandes fuerzas externas. Por otro lado, tiene la habilidad de detectar y actuar de acuerdo a condiciones internas como cambios de temperatura, voltaje, entre otras variables, debido a que incorpora un microcontrolador ATmega8 el cual se comunica a través del protocolo UART-TTL semidúplex.

Tabla 10. Principales características Dynamixel AX12A. (“a X-12,” 2006)

RESOLUCIÓN	0.35°	
ANGULO DE OPERACIÓN	300° o giro continuo	
VOLTAJE	7V a 10V (se recomienda 9.6V)	
CORRIENTE MÁXIMA	0.9ª	
TEMPERATURA DE OPERACIÓN	-5°C a +85°C	
SEÑAL DE MANDO	Digital	
TIPO DE PROTOCOLO	Comunicación serial asíncrona semidúplex (8 bit, 1 paro, sin paridad)	
UNIÓN FÍSICA	Nivel TTL multi-drop (conector tipo Daisy chain)	
POSIBLES ID’S	0 a 254	
VELOCIDAD DE COMUNICACIÓN	7343bps a 1Mbps	
RETROALIMENTACIÓN	Posición, Temperatura, carga, voltaje de entrada	
MATERIAL	Plástico de ingeniería	
PESO (G)	55	
DIMENSIONES		
VOLTAJE DE ENTRADA	a 7V	a 12V
TORQUE (KGF.CM)	12	16.5
S/60GRADOS	0.269	0.196

3.2.1 Terminales del servomotor Dynamixel AX12A

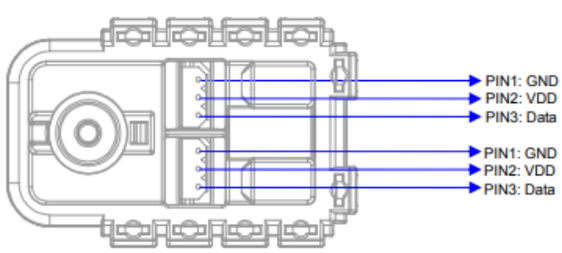


Figura 36. Pines de cada conector, Dynamixel AX-12A. (“a X-12,” 2006)

Los dos terminales están conectados internamente pin a pin, por lo que el AX-12A puede ser operado conectando solo un terminal (Data, VCC, GND) al controlador principal, dejando el otro terminal dispuesto para operar un segundo o múltiples

servos Dynamixel conectados a un mismo nodo. Este método de conexión es conocido como multi-drop. Ver *figura 27*.

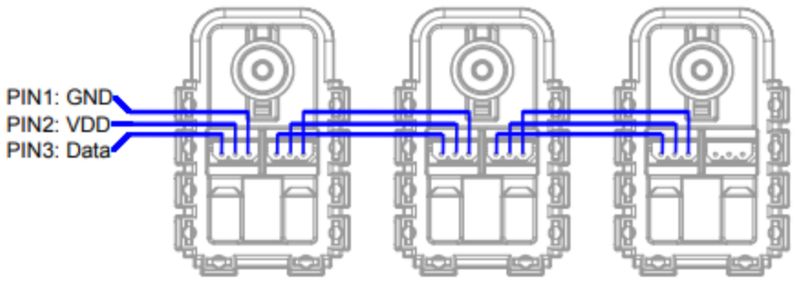


Figura 37. Conexión multi-drop para múltiples servos. (“a X-12,” 2006)

3.2.2 Comunicación entre Dynamixel y Controlador principal:

Para que el controlador principal pueda operar este actuador, debe soportar niveles TTL semidúplex, por lo que es necesario convertir las señales UART del controlador principal a semidúplex. En otras palabras, el pin TX o RX del controlador principal (placa de desarrollo) puede solo enviar o recibir datos a la vez; el Pin DATA del Dynamixel debe, por lo tanto, recibir o enviar datos por ese mismo pin. Para esto, la hoja del fabricante nos muestra un circuito dispuesto internamente en el controlador CM-5 de Robotics para hacer este arreglo de señales.

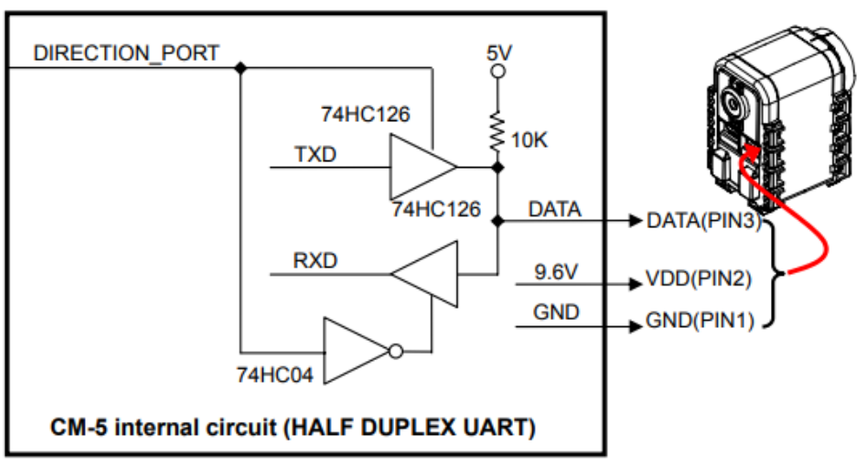


Figura 38. Buffer Tri-estado del CM-5

Este circuito, es esencialmente un Buffer Tri-estado, que permite el envío o recepción de datos a través de un solo cable y asegura que cuando el controlador principal este transmitiendo, el bus no esté conectado al pin RX o que cuando está esperando una recepción de datos no esté siendo interferido por el pin TX (AJNA & HERSAN, 2019).

Existen dos posibilidades para comunicar los servos Dynamixel. Adquirir el controlador CM-5 u algún otro controlador especial para el control de este tipo de servos, (“CM-5 - ROBOTIS,” 2015), (“DYNAMIXEL Shield | SANDOROBOTICS,” 2018), (“Shield - Dynamixel AX,” 2017), o montar directamente el circuito, aunado al

uso de librerías externas para comunicar los Dynamixel con cualquier otro controlador (Savage, 2019), (“Tarjeta interfaz Dynamixel,” 2019).

3.2.3 Paquetes de comunicación

El controlador principal se comunica con el Dynamixel por medio de un envío y recepción de paquetes. Hay dos tipos de paquetes; los paquetes de instrucción que son enviados desde el controlador principal a los Dynamixel y los paquetes de estado que son enviados desde los Dynamixel al controlador principal.

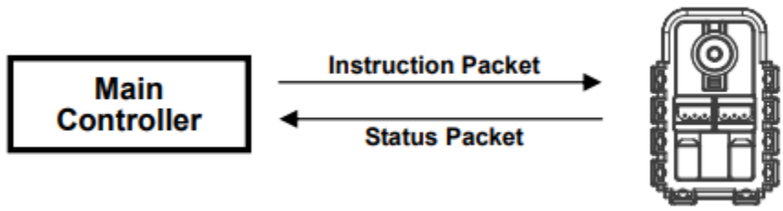


Figura 39. Paquetes Dynamixel (“a X-12,” 2006)

Para la conexión del sistema anterior, *figura 42*, si el controlador principal envía un paquete de instrucción con un ID=N, solo el Dynamixel con esta asignación de ID retornara su respectivo paquete de estado y cumplirá con la instrucción requerida.

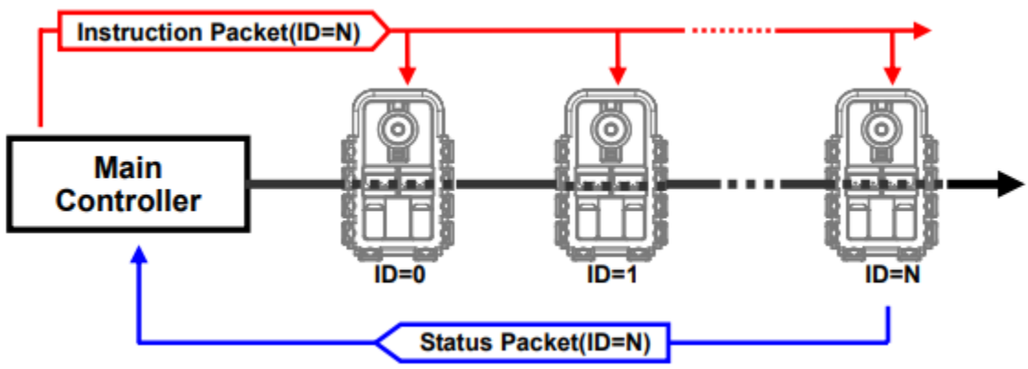


Figura 40. Comunicación Dynamixel (“a X-12,” 2006)

Por ende, si existen dos o varias ID’s con el mismo valor, los paquetes de datos colisionaran entre sí, conllevando a problemas de comunicación. Por lo que se hace necesario que la ID sea única dentro del nodo de red que comunica los Dynamixel (“a X-12,” 2006)

3.2.4 Tabla de control

La tabla de datos es una estructura de dos instrucciones implementadas en el Dynamixel, con el fin que el usuario pueda leer datos específicos y obtener el estado del servo con los paquetes de *leer instrucción*, y también modificar los datos con los paquetes *escribir instrucción* para controlar el servo.

Esta tabla consiste de múltiples campos de datos para guardar el estado o controlar el servo, las direcciones de estos paquetes de instrucción son valores únicos que son almacenados en la memoria EEPROM y en la memoria RAM, de acuerdo a su aplicación. Esta tabla está comprendida de la siguiente manera:

- Área: la tabla de control está dividida en 2 áreas. Los datos de la RAM son restaurados a los valores iniciales una vez se quita la alimentación (volátil), y por otra parte, los datos de la EEPROM se mantienen incluso después de haber interrumpido la alimentación. (No volátil).
- Tamaño: el tamaño de los datos varían de 1 a 2 bytes dependiendo del uso.
- Acceso: existen dos tipos diferentes de propiedades de acceso, 'RW' que significa permiso de acceso de lectura y escritura, mientras que 'R' permite solo el permiso de acceso solo para lectura. Los datos con la propiedad 'R' no se pueden cambiar por una instrucción de lectura. Esta propiedad es generalmente usada para monitorear y la propiedad 'RW' es usada para el control del dispositivo ("AX-12A," 2017).

Tabla 11. Tabla de control del área EEPROM, ("AX-12A," 2017)

Address	Size (Byte)	Data Name	Description	Access	Initial Value
0	2	Model Number	Model Number	R	12
2	1	Firmware Version	Firmware Version	R	-
3	1	ID	DYNAMIXEL ID	RW	1
4	1	Baud Rate	Communication Speed	RW	1
5	1	Return Delay Time	Response Delay Time	RW	250
6	2	CW Angle Limit	Clockwise Angle Limit	RW	0
8	2	CCW Angle Limit	Counter-Clockwise Angle Limit	RW	1023
11	1	Temperature Limit	Maximum Internal Temperature Limit	RW	70
12	1	Min Voltage Limit	Minimum Input Voltage Limit	RW	60
13	1	Max Voltage Limit	Maximum Input Voltage Limit	RW	140
14	2	Max Torque	Maximun Torque	RW	1023
16	1	Status Return Level	Select Types of Status Return	RW	2
17	1	Alarm LED	LED for Alarm	RW	36
18	1	Shutdown	Shutdown Error Information	RW	36

Tabla 12. Tabla de control del área RAM, ("AX-12A," 2017)

Address	Size (Byte)	Data Name	Description	Access	Initial Value
24	1	Torque Enable	Motor Torque On/Off	RW	0
25	1	LED	Status LED On/Off	RW	0
26	1	CW Compliance Margin	CW Compliance Margin	RW	1
27	1	CCW Compliance Margin	CCW Compliance Margin	RW	1
28	1	CW Compliance Slope	CW Compliance Slope	RW	32
29	1	CCW Compliance Slope	CCW Compliance Slope	RW	32
30	2	Goal Position	Target Position	RW	-
32	2	Moving Speed	Moving Speed	RW	-
34	2	Torque Limit	Torque Limit(Goal Torque)	RW	ADD 14&15
36	2	Present Position	Present Position	R	-
38	2	Present Speed	Present Speed	R	-
40	2	Present Load	Present Load	R	-
42	1	Present Voltage	Present Voltage	R	-
43	1	Present Temperature	Present Temperature	R	-
44	1	Registered	If Instruction is registered	R	0
46	1	Moving	Movement Status	R	0
47	1	Lock	Locking EEPROM	RW	0
48	2	Punch	Minimum Current Threshold	RW	32

3.3 CONTROLADOR PRINCIPAL

Consecuentemente, para la elección del controlador principal, se listan los pines y protocolos de comunicación necesarios para conectar los sensores y actuadores del robot. Ver *tabla 13*. Por lo que a la hora de elegir se puede encontrar en el mercado un gran abanico de opciones de controladores y se hace necesario filtrarlos de acuerdo a las necesidades y limitaciones del proyecto. Existen placas que van desde alto poder de procesamiento como la Raspberry y Beagle Bone, hasta la Arduino y la NodeMCU, que son controladores de menor desempeño pero muy versátiles y reconocidos igualmente dentro de la comunidad de código abierto.

En vista que necesita un arreglo de señales para comunicar los servos Dynamixel, se puede optar por la implementación de un circuito externo y librerías especiales que hagan este trabajo o hacerse al uso de controladores especializados. Para se escogen algunas placas dentro de cada gama y se analiza cada una de sus prestaciones en contraste con las necesidades del proyecto.

Tabla 13. Pines y uso

PIN	USO
UART (TX, RX)	Dynamixel
I2C (SDA,CLK)	VL53L0X
DIGITAL	Microservo

Dentro de los controladores con altas prestaciones, se toma la Raspberry PI; como opción para bajas prestaciones se toma la NodeMCU. Del mismo modo, la OpenCM09 como una de las opciones que incorpora la circuitería para el uso de servo motores Dynamixel, además de sus otras características. Cualquiera de estos controladores pueden ser empleados en este proyecto, pero se quiso hacer un análisis detallado y revisar más a profundidad cuál de esos es el más apropiado.

3.3.1 Raspberry Pi Zero W

Este computador embebido en una pequeña placa, es uno de los más recientes lanzamientos de controladores de la serie Raspberry Pi que integra comunicación inalámbrica a través de WIFI y Bluetooth 4.0, lo que es indispensable en proyectos LOT. Sin embargo, los periféricos del ratón, teclado, monitor/HDMI y cámara tienen dimensiones reducidas con la intención de ganar espacio, por lo que es necesario valerse de adaptadores y cables adicionales para poder hacer uso de ellos (Sparkfun, 2018). Como resultado, se ha de pagar un poco más por esos accesorios, pero al final, todo se compensaría con el uso al máximo de todas sus prestaciones y capacidad, de lo contrario no se justificaría su elección.

Figura 41. Periféricos y adaptadores para la Pi Zero W (Sparkfun, 2018)

Generalmente el sistema operativo nativo es Linux, y es indispensable conocer algunos comandos básicos de este entorno, también existe la posibilidad de instalar Windows como sistema operativo pero en vista de que es un sistema mucho más estructurado, ocupa mucha más memoria de procesamiento y su rendimiento no es aún el más óptimo (Pastor, 2019).

Tabla 14. Comparación de controladores de la serie Raspberry Pi

MODELO	RASPBERRY PI 1 B+	RASPBERRY PI 2 B	RASPBERRY PI 3 B	RASPBERRY PI ZERO	RASPBERRY PI ZERO W
FECHA	2012 Feb 15	2015 Feb 1	2016 Feb 29	2015 Nov 30	2017 Feb 28
PRECIO	30	35	35	5	10
SOC	Broadcom BCM2835	Broadcom BCM2836	Broadcom BCM2837	Broadcom BCM2835	Broadcom BCM2835
CORE	ARM1176JZF-S	Cortex-A7	Cortex-A53 64-bit	ARM1176JZF-S	ARM1176JZF-S
Nº CORES	1	4	4	1	1
GPU	VideoCore IV				
CPU CLOCK	700 MHz	900 MHz	1.2 GHz	1 GHz	1 GHz
RAM	512 MB	1 GB	1 GB	512 MB	512 MB
MEMORIA	Micro SD	Micro SD	Micro SD	Micro SD	Micro SD
USB	2	4	4	1 microUSB	1 microUSB
ETHERNET	SI	SI	SI	No	No
WI-FI	No	No	SI	No	SI
BLUETOOTH	No	No	SI	No	SI
HDMI	Si	Si	Si	Mini	Mini
GPIO	8	17	17	17	17
UART	Si	Si	Si	Si	Si
SPI	Si	Si	Si	Si	Si
I2C	Si	Si	Si	Si	Si
DSI (LCD)	Si	Si	Si	No	No
CAMARA	Si	Si	Si	Si	Si
ALTURA	85.6 mm	85.6 mm	85.6 mm	65 mm	65 mm
ANCHO	53.98 mm	56.5 mm	56.5 mm	30 mm	30 mm
PROFUNDIDAD	17 mm	17 mm	17 mm	5 mm	5 mm
PESO	45 g	45 g	45 g	9 g	9 g
CONSUMO	700 mA	820 mA	1400 mA	350 mA	350 mA

Dentro del rango de posibilidades y versiones, sin duda alguna la Raspberry PI Zero W es una opción bastante acertada por su conectividad y su costo reducido dentro de la gama Raspberry.

3.3.2 Open CM09

Es un microcontrolador de código libre desarrollado por la compañía Robotics, que incluye un par de periféricos especiales para conectar servomotores Dynamixel de la serie TX-320, para el uso de la serie AX y MX es necesario emplear una tarjeta de expansión OpenCM. Del mismo modo, el uso de la BT-100 y/o ZIG 110 para comunicación inalámbrica. Como puntos a favor de controlador, es que incorpora diversos sensores externos y suficientes pines de entrada y salida para conectar diversas señales de control en un micro-controlador de 32 bits.

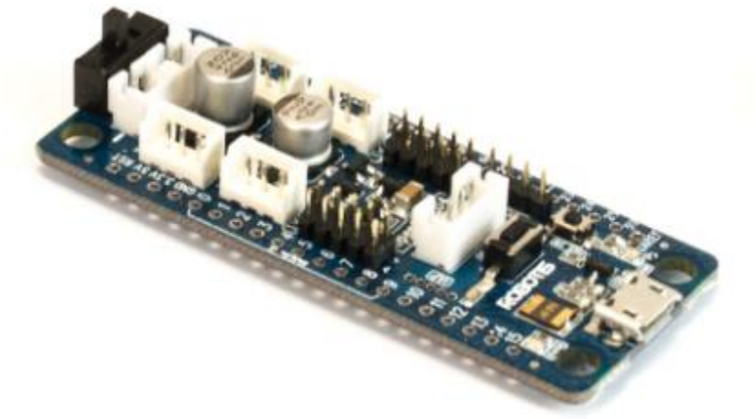


Figura 42. OpenCM9.04

Tabla 15. Características del OpenCM9.04 ("ArbotiX-M Robocontroller," 2014)

CPU	STM32F103CB (ARM Cortex-M3)
VOLTAJE DE OPERACIÓN	7-16 V (5 V programming only over USB)
PINES I/O EXTERNOS	26
TEMPORIZADORES	8
PINES ANÁLOGOS (ADC)	10 (12 bit)
FLASH	128 kB
SRAM	20 kB
VELOCIDAD DE RELOJ	72 MHz
USB 2.0 DE ALTA VELOCIDAD	1
CAN	1
USART	3
SPI	2
I²C (TWI)	2
DEPURADOR	JTAG & SWD
PUERTO TTL (3-PIN)	4
PUERTOS SENSORES EXTERNOS (5 PIN)	4
DIMENSIONES	27 x 66.5 mm
PESO	13 g

3.3.3 NodeMCU – ESP8266 V2

La ESP8266 es un microcontrolador diseñado por *Espressif Systems*. El cual tiene WIFI integrado, siendo aun así una de las placas más económicas del mercado y muy asequible tanto en proyectos de robótica e internet de las casas (LOT).

La forma de usar este módulo es a través de comandos seriales, ya que básicamente es un transmisor WIFI/serial que puede ser programado en LUA y en la IDE de arduino, por lo que con pocas líneas de código en C y el uso de librerías externas se podrá establecer: conexión WIFI, control de los pines GPIO, convertir el microcontrolador en un servidor web, etc. (Handson Technology, 2017).

Figura 43. ESP8266 NodeMCU WiFi Devkit (Handson Technology, 2017)

Tabla 16. Características ESP8266 V2 (Handson Technology, 2017)

MICROCONTROLADOR	Tensilica 32-bit RISC CPU Xtensa LX106
VOLTAJE	3.3V
VOLTAJE DE ENTRADA	5-12V
WI-FI DIRECT	(P2P), soft-AP
CONSUMO DE CORRIENTE	10uA~170mA
MEMORIA FLASH	16MB max (512K normal).
PROTOCOLOS INTEGRADOS	TCP/IP
PROCESADOR	Tensilica L106 32-bit.
VELOCIDAD DEL PROCESADOR	80~160MHz
RAM	32K + 80K.
GPIOS	17 (Multiplexados con otras funciones).
ANALOG TO DIGITAL	1 input with 1024 step resolution.
UART	1
I2C	1
SPI	1
SRAM	64KB

En la siguiente tabla se mencionan los puntos a favor y en contra de cada una de las placas de acuerdo a las especificaciones de nuestro proyecto.

Tabla 17. Elección del controlador de acuerdo a requerimientos del proyecto

Opción Controlador	Precio	Comunicación	Dimensiones	Consumo
(1). ESP8266	X	X	X	X
(2). Pi Zero W		X	X	X
(3). OpenCM9.04			X	X

Dentro de este aspecto se consideraron puntos como: el precio, ya que en definitiva, abrirse a la posibilidad de optimizar costos no afectando los objetivos del proyecto es un factor determinante. La comunicación que precisamos es WIFI y I2C y para este caso solo la opción (1) y (2) cumplen estos requisitos. La dimensión de la placa siempre es un punto a considerar, ya sea, para reducir peso y/o ganar espacio de trabajo. Por lo tanto, contar con un controlador de dimensiones reducidas ayudaría a mejorar la distribución de la electrónica y espacios para que los movimientos internos del mecanismo no se vean afectados. Por último, el consumo favorece para un mayor rendimiento de la batería, a saber que, dentro de las tres opciones todas las placas tenían tasas de consumo relativamente bajo.

Tomando estos puntos en consideración se optó por la placa de desarrollo NodeMCU que tiene el chip ESP8266. Principalmente porque ofrece el uso de comunicación WIFI inmerso en la placa y todo a un precio significativamente bajo en comparación con los demás controladores.

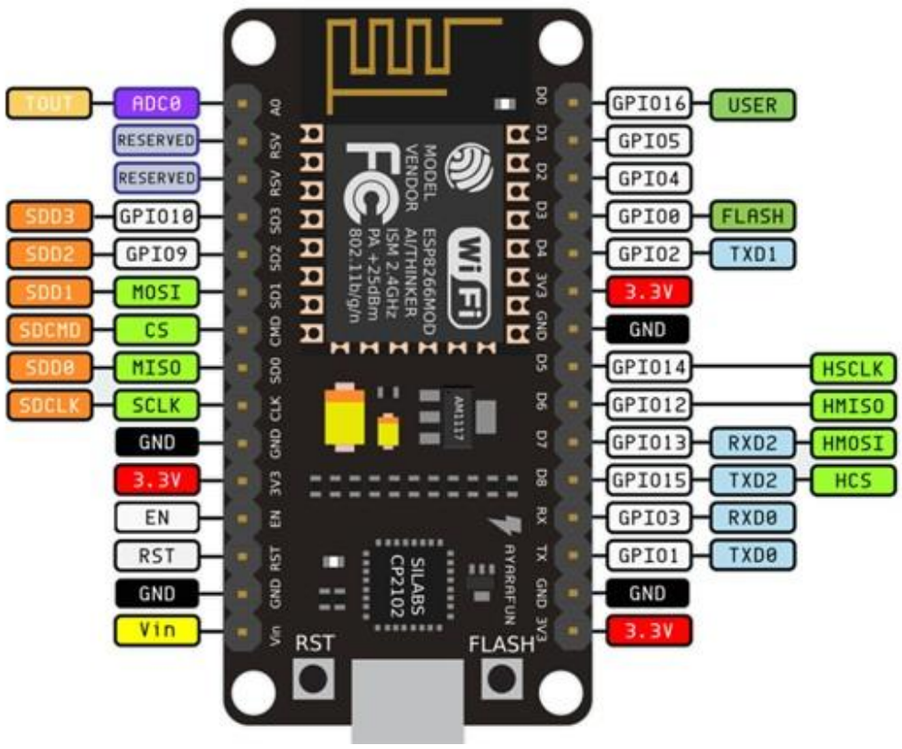
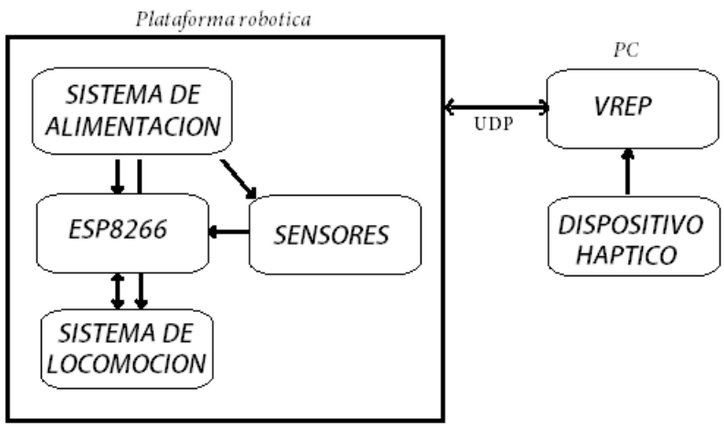


Figura 44. Disposición de pines de la ESP8266, (Handson Technology, 2017)

CAPITULO IV

4. DESARROLLO

El proyecto está comprendido por sistemas de entradas y salidas de la forma como se expresa en la siguiente figura.



45. Esquema del proyecto

Se empezará por el diseño eléctrico – que engloba, el sistema de alimentación y la circuitería electrónica en general. Posteriormente en el análisis y diseño mecánico donde se explicará el mecanismo del sensor utilizado para la detección de obstáculos, adaptación de las llantas holonomicas y estructura final de la plataforma robótica. Finalmente se abordara la comunicación UDP (VREP – ESP), el algoritmo de control y retroalimentación háptica.

4.1 DISEÑO ELECTRÓNICO

Para el análisis de consumo de corriente, se listan a continuación todas las demandas energéticas máximas de cada componente electrónico de acuerdo con las hojas del fabricante.

Tabla 18. Consumo de corriente por cada componente.

COMPONENTE	CANTIDAD	VOLTAJE	CONSUMO
DYNAMIXEL AX12A (X4)	4	9 – 12V	3200 mA
SERVO MG90S	1	4.8 – 6V	300 mA
SENSOR VL53L0X	2	3.3 – 5V	38 mA
BUFFER TRI-ESTADO 74LS241	1	5V	27 mA
ESP8266 NODEMCU	1	7 – 12V	170 mA
LED	1	5V	30 mA
TOTAL			≈3900 mA

Con base en el rango de voltajes a cubrir en el sistema electico, se opta por trabajar con una batería LiPo de 3 celdas a 11.1V suficientes para alimentar los Dynamixel, y hacer una reducción a 5V para los demás componentes con la ayuda de un regulador de voltaje.

La capacidad de la batería (mAh) se calcula de acuerdo a la autonomía que a la cual se quiera que el robot opere sin necesidad de carga. Para el proyecto se consideró que 40 minutos de autonomía es más que suficiente, tomando el caso donde todos los componentes operen al 80% de su máxima demanda eléctrica, se tiene entonces:

$$3900mA * 80\% = 3120mA \quad (15)$$

$$mAh = 3120mA * 0.67h = 2090 \quad (6)$$

Por lo que la capacidad comercial de batería que más se ajusta al cálculo anterior, son 2200 mAh. A continuación se muestras las características de la batería.

Figura 46. Batería LIPO 11.1V - 2200mAh

Tabla 19. Características de la batería

CAPACIDAD MÍNIMA	2200 mAh
CONFIGURACIÓN	3S1P / 11.1v / 3Cell
DESCARGA CONSTANTE	25C
DESCARGA MÁXIMA (10 S)	35C
PESO	188g
DIMENSIONES	105 x 33 x 24 mm
ENCHUFE DE CARGA	JST-XH
ENCHUFE DE DESCARGA	XT60

Ahora, si se hace una estimación entre la corriente proporcionada por hora de la batería y la corriente consumida por el sistema, y se realiza el cálculo para un consumo eléctrico del 100%, se tiene que:

$$Duración\ Max\_consumo = \frac{2200mAh}{3900mA} = 0.564hr\left(\frac{60min}{1h}\right) = 33.9min \quad (7)$$

Dando como resultado una autonomía de entre 30 a 35min en el caso más extremo; cabe mencionar que los motores trabajaran máximo con un consumo de 600 mA con carga y no a 900 mA siendo este el caso de obstrucción por completo el giro de la rueda.

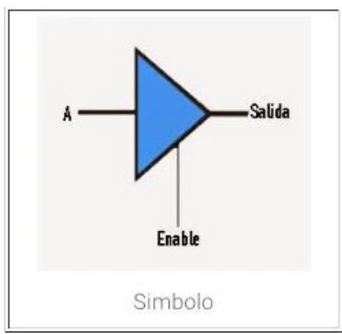
4.2 CIRCUITO ELÉCTRICO

En esta parte se toman en cuenta los diferentes voltajes de operación para cada componente electrónico mencionados en la *tabla 18*, y se procede a la implementación de:

- Regulación de voltaje (12 a 5v)
- Circuito Buffer Tri-estado para Dynamixel

4.2.1 Circuito Buffer Tri-estado para Dynamixel:

El Buffer o Compuerta lógica de tres estados presentan tres estados de salidas diferentes: un estado de bajo nivel (0), un estado de alto nivel (1), un estado de alta impedancia o estado flotante (Z) donde la salida se comporta como si aún no estuviera conectada al circuito, excepto por una pequeña corriente de fuga que puede fluir hacia adentro o hacia afuera de la terminal de salida.



EN	A	SALIDA
0	0	HI-Z
0	1	HI-Z
1	0	0
1	1	1

Tabla de verdad
HI-Z=alta impedancia

Figura 47. Tabla de verdad del Buffer o compuesta de tres estados, (“Compuerta lógica de tres estados - Ingeniería Mecafenix,” n.d.)

Normalmente, lo que se busca con el Buffer Tri-estado, es permitir que dos o más circuitos puedan compartir el mismo bus o línea de salida, de forma tal, que no transmitan datos a la vez, ya que si ambas señales circularan por la misma línea, no se podría determinar el valor que está circulando en la misma.

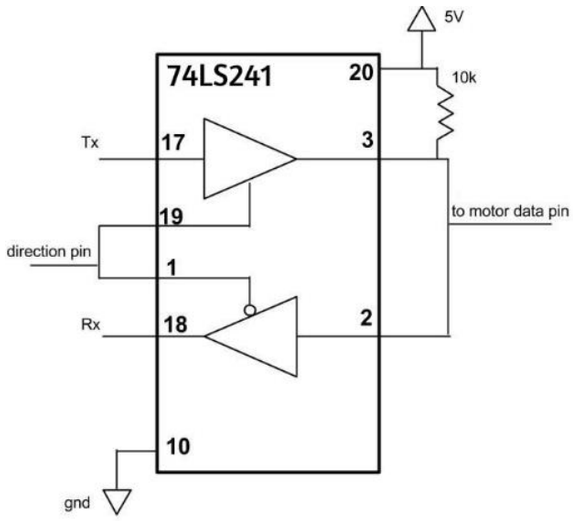


Figura 48. Uso de 2 Buffers Tri-estado del integrado 74LS241 para el control de datos del Dynamixel. (AJNA & HERSAN, 2019)

Es así, que para permitir el paso de una sola señal (lo que se conoce como semidúplex), en este caso TX o RX, se utiliza el Buffer Tri-estado 74LS241, (AJNA & HERSAN, 2019). Que es un integrado de 20 pines con una línea de 8 buffers, pero que solo se utilizan dos. El pin 17 para TX, el pin 18 para RX, el pin 1 y 19 (*enable*) para habilitar la recepción o transmisión de datos respectivamente y por último los pines 3 y 2 que se conectan al pin de datos del Dynamixel.

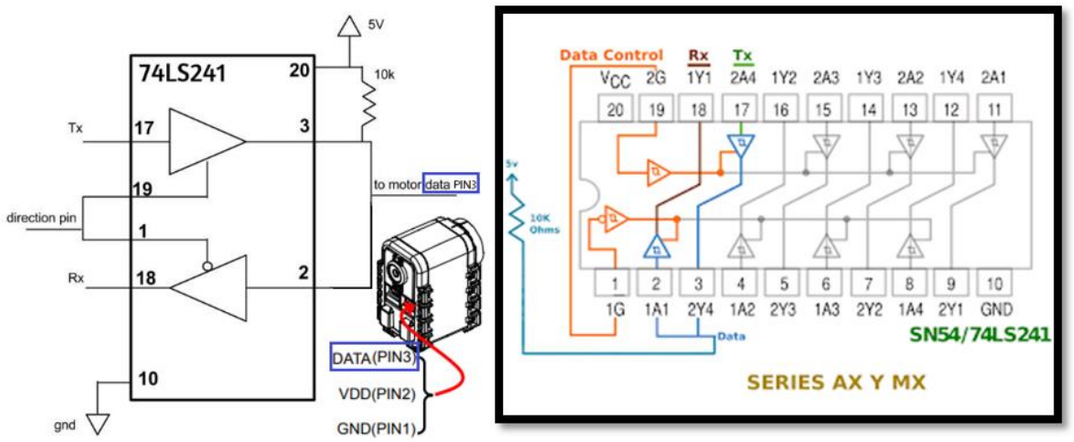


Figura 49. Conexiones para el integrado, ("Driving Robotis Dynamixel Servos with Arduino - Robosoup," 2016)

Como es de esperar, con la ayuda del buffer 74LS241 se puede escribir y leer información del Dynamixel, Ver *figura 55.* De otra forma, si no se implementa el buffer tri-estado, no se podrá leer información tal como posición, temperatura, torque, entre otras. En la *figura 54,* se observa la conexión en físico de este buffer utilizando un arduino.

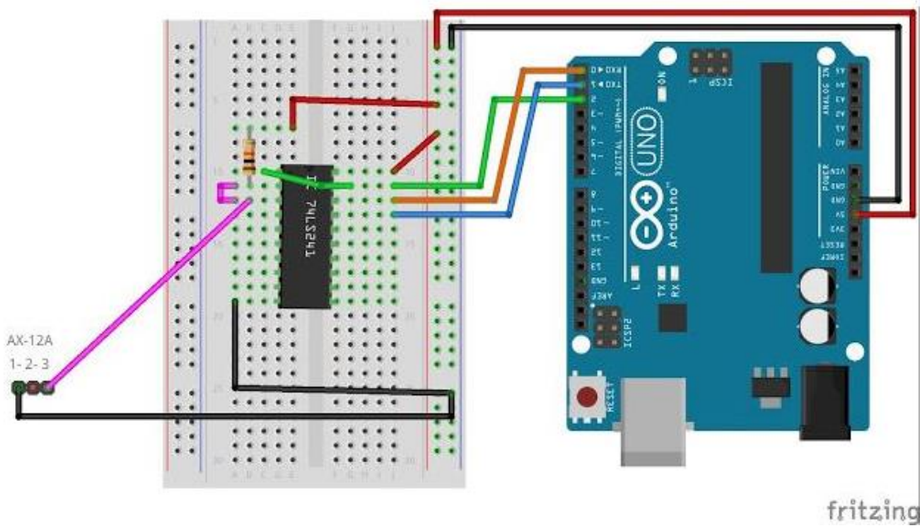


Figura 50. Conexión Buffer Tri-estado - Dynamixel con Arduino, (*driving-robotic-dynamixel-servos @ www.arduinotutorialonline.com*, 2019)

```
#include <AX12A.h>
#define DirectionPin  (2u)
#define BaudRate    (1000000ul) // <--- Servo baud rate Default 1000000
#define ID    (1u)
int reg = 0;
void setup()
{
 ax12a.begin(BaudRate, DirectionPin, &Serial);
 delay(1000);
 ax12a.move(1,100);
 delay(1000);
}
void loop()
{
 reg = ax12a.readPosition(ID);
 Serial.println(reg);
 delay(1000);
}
```

Figura 51. Ejemplo1 - Envío y escritura Dynamixel - IDE arduino

Para activar el modo giro continuo y definir una nueva ID, se deben establecer esas condiciones desde el *void setup*, con la ayuda de las funciones predefinidas en la librería. A su vez cabe mencionar que la tasa de baudios por defecto del Dynamixel es de 1M baud, por lo que si se reducen o aumenta esa cantidad, el Dynamixel no funcionará.

```
#include <AX12A.h>
#define DirectionPin  (10u)
#define BaudRate      (1000000ul)
#define ID            (1u)
#define velocidad     (250)
#define delay1         (5000)
void setup()
{
  // Se activan todos los Dynamixel en modo giro continuo
  for (int i = 1; i < 4; i++) {
    ax12a.begin(BaudRate, DirectionPin, &Serial);
    ax12a.setEndless(i, ON); //ax12a.setEndless(ID, ON)
  }
  void loop()
  {
    // ax12a.turn(ID_Dynamixel, RIGHT o LEFT, velocidad_de_0_hasta_1020);

    ax12a.turn(1, RIGHT, velocidad);
    ax12a.turn(2, LEFT, velocidad);
    ax12a.turn(3, LEFT, velocidad);
    ax12a.turn(4, RIGHT, velocidad);
    delay(delay1);
  }
}
```

Figura 52. Ejemplo2 – Modo giro continuo para 4 Dynamixel - IDE arduino

4.2.2 Regulación de voltaje (12 a 5v)

De acuerdo al cálculo de consumo de corriente, se necesita un regulador con una corriente de salida > a 1A, No obstante, se hicieron pruebas con el integrado LS7805 que entrega a la salida 1A. Sin embargo, el integrado junto al disipador de área pequeña conectado a él se calentaban en sobremanera después de un par de minutos de operación.

Otra alternativa dentro del mercado, con salida de corriente >> a 1A, es el LM2596, un integrado regulador de voltaje de hasta 40V a 5V y 3A a la salida.

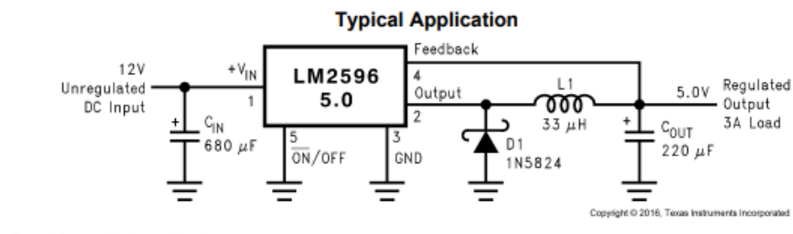


Figura 53. Regulación de voltaje utilizando el chip LM2596

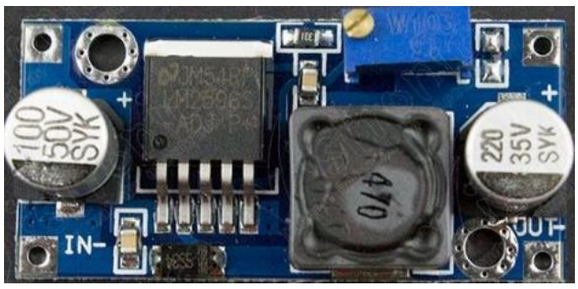


Figura 54. Modulo regulador de voltaje con LM2596

4.2.3 Circuito general – Baquelita

El circuito electrónico se compone de la ESP8266 y respectivas líneas de comunicación con los sensores, Servomotores, interruptor de encendido con led indicador.

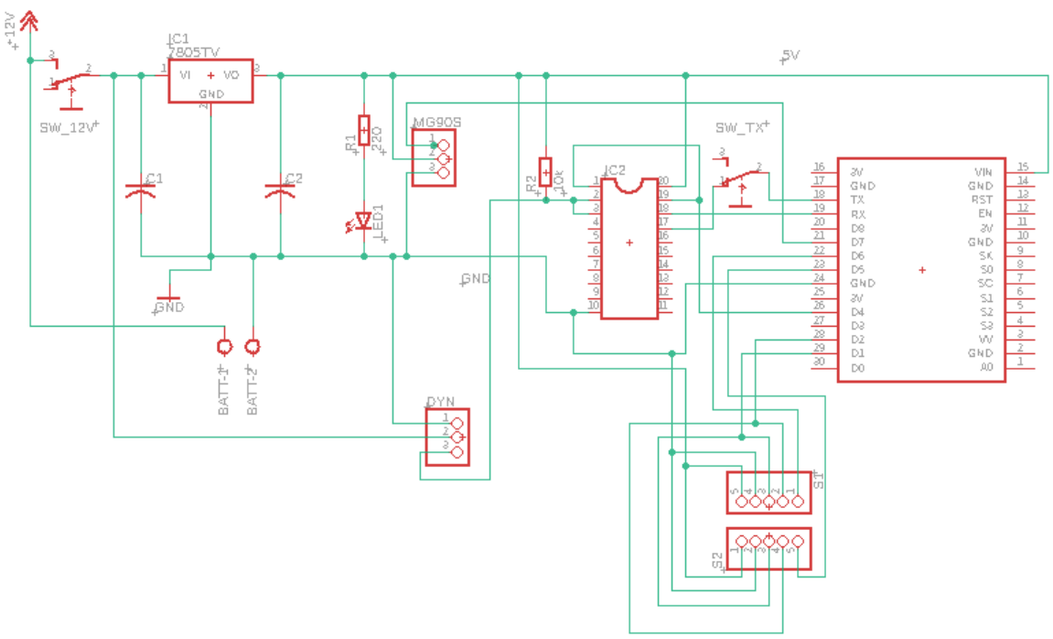


Figura 55. Esquemático del diseño eléctrico. (Autor)

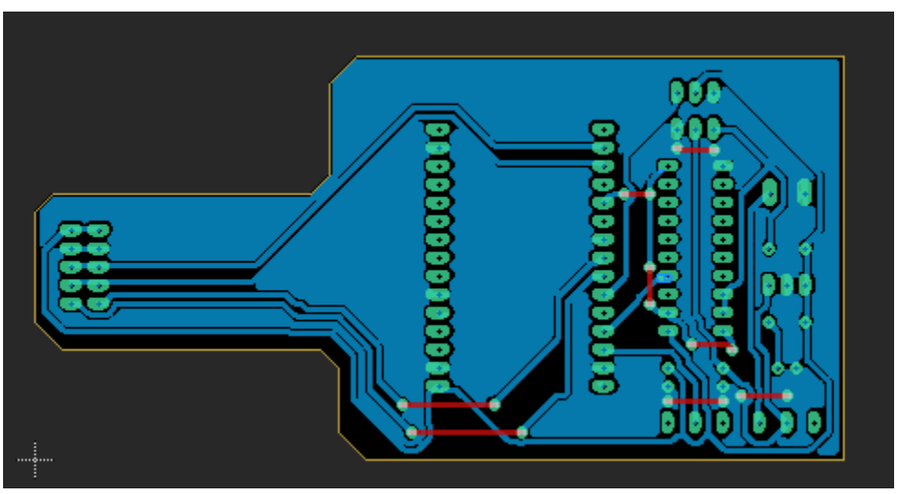


Figura 56. Diseño eléctrico final – EAGLE. (Autor)

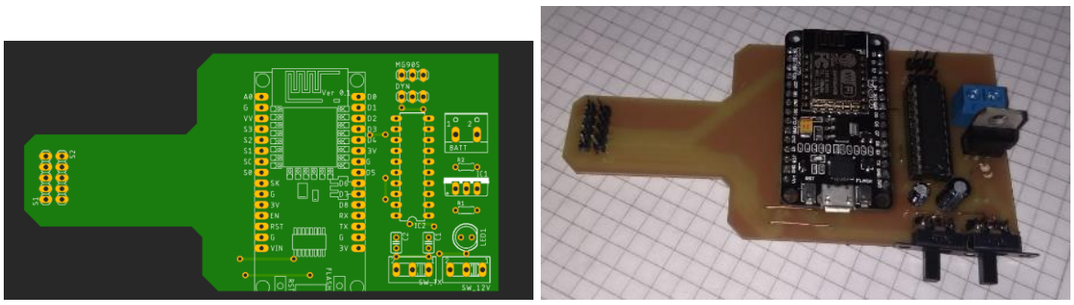


Figura 57. Placa manufacturada. (Autor)

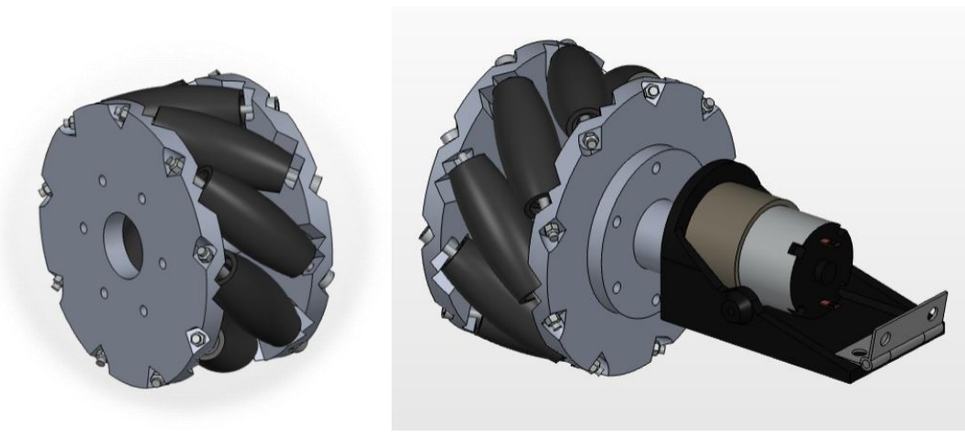
4.3 ANÁLISIS Y DISEÑO MECÁNICO

A continuación se describe de forma detallada el proceso que se llevó a cabo para el diseño y construcción de la plataforma robótica.

4.3.1 Llantas

El tipo llantas de llantas elegidas reduce la complejidad del sistema de control de movimiento, gracias a que las llantas solo giran hacia delante o hacia atrás para generar movimientos en cualquier sentido del plano (x, y), eliminando así las restricciones de los vehículos no-holonomicos.

Sin embargo, su diseño suele ser un poco complejo, ya que cada uno de los rodillos debe estar ubicado a 45° con respecto al eje de la llanta. Por este motivo, se decide no optar por el diseño de la llanta desde cero pero si por su fabricación desde un modelo CAD (“Mecanum Wheel vehicle | 3D CAD Model Library | GrabCAD,” 2017), utilizando impresión 3D, y posteriormente hacer modificaciones de acuerdo a nuestro proyecto.

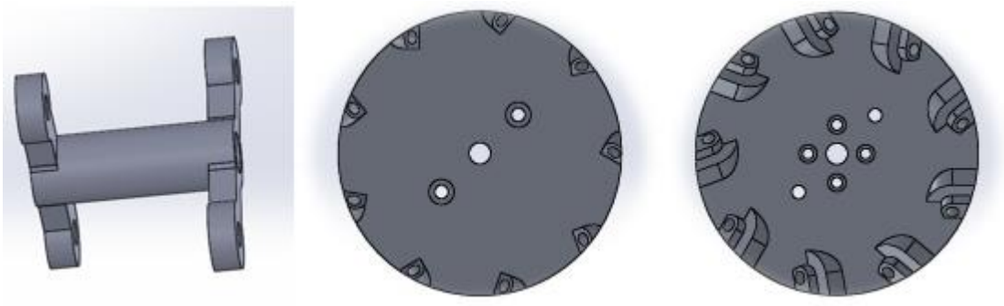


58. Modelo de llanta Mecanum, GrabCAD.

Los ajustes que se hicieron fueron los siguientes:

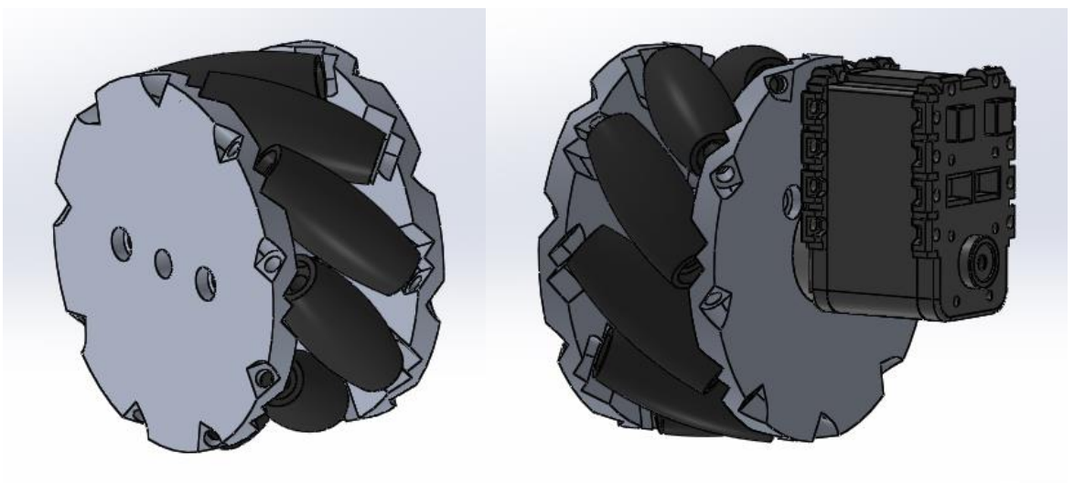
- Redimensionamiento
- Eje para mayor soporte

- Acople para motor Dynamixel



59. Cambios aplicados al modelo. (Autor)

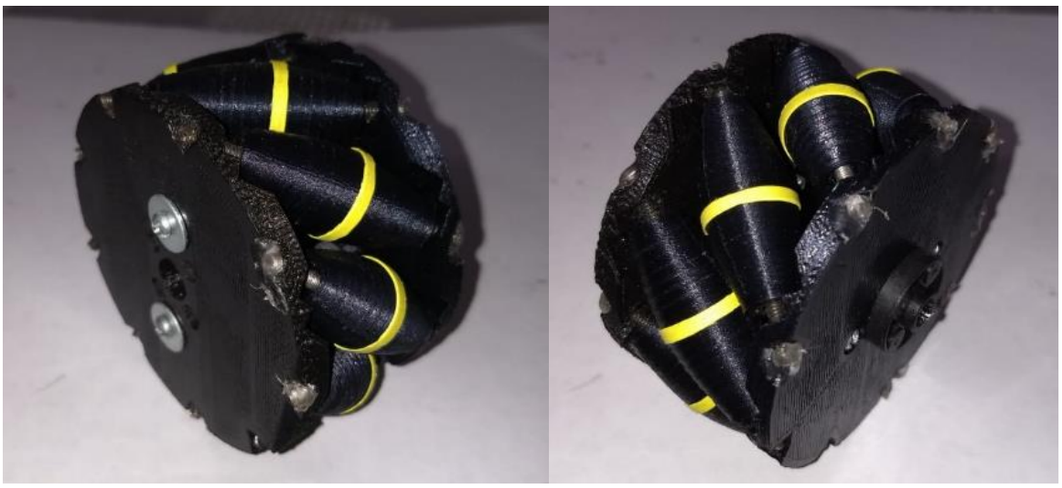
Cada uno de estos ajustes fue pensado para mejorar la rigidez de la llanta y lograr la buena sujeción de la misma.



60. Modelo CAD con las adaptaciones para nuestro proyecto. (Autor)

Los ejes empleados inicialmente para los rodillos de la llanta, eran tornillos 3x50mm los cuales permitían un giro adecuado de los mismos, pero la cabeza y la tuerca del tornillo no permitía el contacto adecuado de los rodillos con el suelo.

Para mejorar esto, se optó por dejar el tornillo sin cabeza y utilizar silicona líquida como sujeción. También, se empleó un aro de caucho (termo-incogible) colocados en la parte central del rodillo para aumentar la fricción con el suelo.



61. Prototipo final de la llanta holonómicas, con aros de caucho. (Autor)

Cabe mencionar que la capacidad de carga de las llantas impresas no se compara con la capacidad de carga de las llantas fabricadas en otro tipo de material como aluminio (alrededor de 50kg) o algún otro material mucho más resistente. No obstante, estas llantas prototipo de la figura 44, llegan a soportar entre 3 a 5kg de peso, lo cual es más que suficiente para el peso que soportará la estructura.

4.3.2 Estructura

La estructura mecánica fue pensada de acuerdo a la disposición y movimiento de los sensores, ya que como parte principal del proyecto, afectaría las dimensiones y capacidad de carga del robot. Para esto se decide utilizar el MDF (Medium Density Fiberboard) como material base, por sus propiedades mecánicas en estructuras no tan complejas, peso y costo.

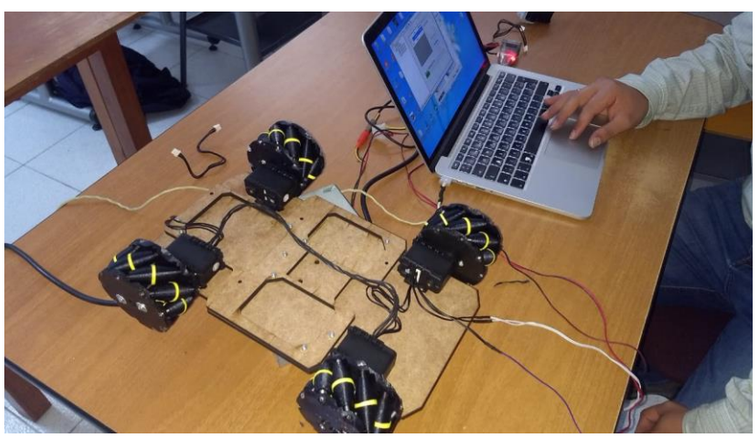


Figura 62. Montaje inicial de las llantas con termo incogible amarillo – Prueba de movimiento con USB2DYNAMIXEL. (Autor)

Figura 63. Prueba de movimiento con ARDUINO - Programación de movimientos. (Autor)

Para la sujeción de los servomotores, se emplearon las fichas de soporte FP04F2, las cuales fueron rediseñadas de forma impresa para ser sujetadas con tornillos M3, ya que los tornillos M2 que incluye el servo motor son muy pequeños, lo que hacía complicado y poco seguro sujetarlos a las placas de MDF.

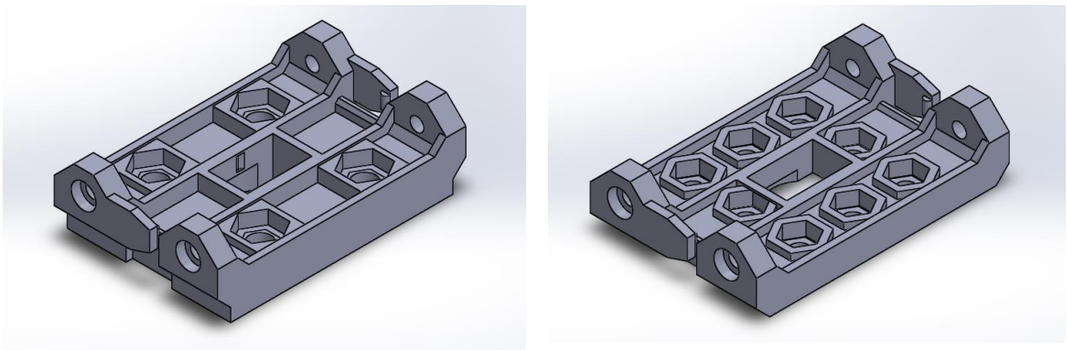


Fig. 1. Izquierda: soporte FP04F2 rediseñado. Derecha: soporte FP04F2 original. – CAD. (Autor)

Fig. 2. Derecha: Impresión del soporte FP04F2 rediseñado. Izquierda: soporte FP04F2 original. (Autor)

Por otra parte, el mecanismo de movimiento de 360° se planteó tomando como referencia un mecanismo utilizado para la detección de obstáculos en un Drone (MORCILLO MARTÍNEZ, 2018), *Figura 66.* Pero, en definitiva no fue la mejor opción, ya que el sensor ocupaba el espacio necesario para ubicar la base de un brazo robótico como posible carga externa.

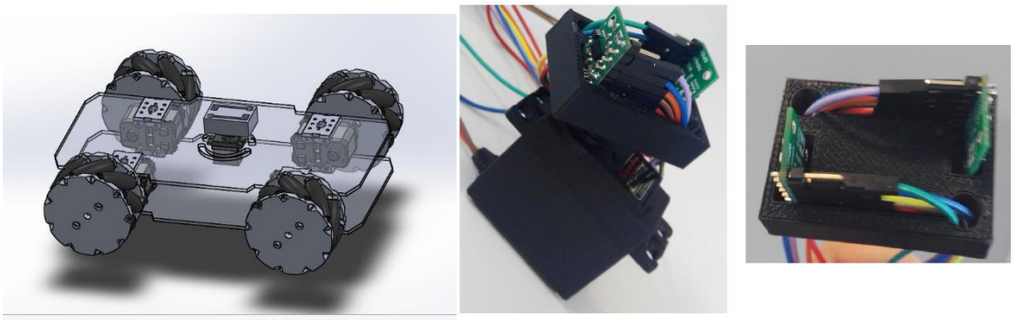


Figura 64. Propuesta inicial - Imagen a la derecha fue el usado para el Drone, (MORCILLO MARTÍNEZ, 2018).

Entonces, con base en el mismo movimiento, se diseña el mecanismo con un espacio libre de 125mm de diametro, de tal manera que se pueda alojar cargas externas justo en el centro de masa de la plataforma robótica.

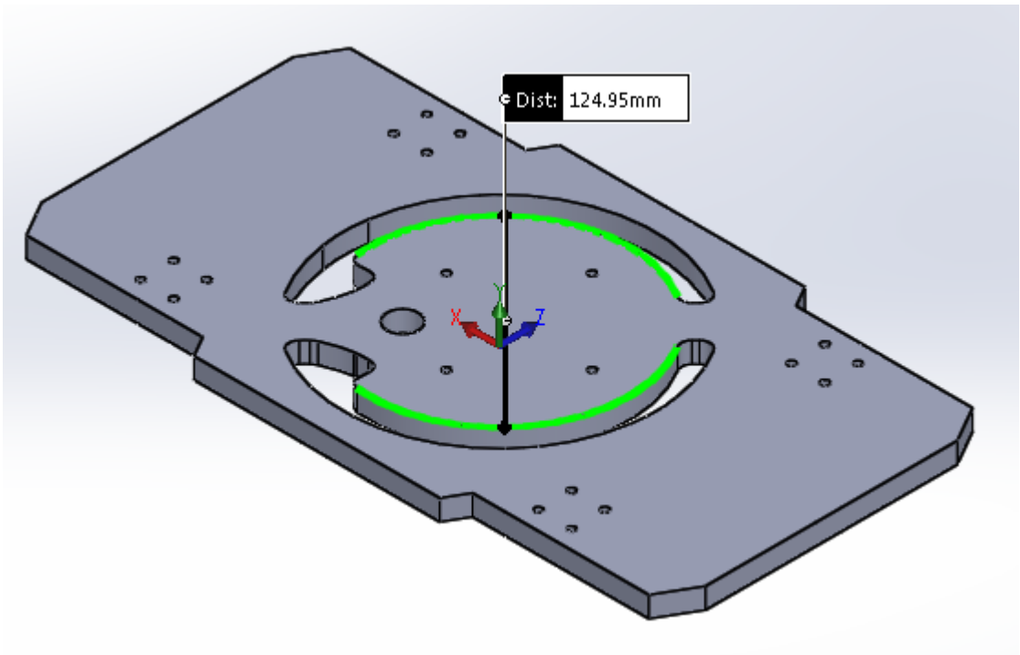


Figura 65. Diseño de placa superior – Solidworks. (Autor)

Para conocer el peso que podria soportar esta placa sin llegar a romperse, es necesario conocer algunas propiedades mecanicas del MDF y proceder al analisis de falla en solidworks.

4.3.2.1 Analisis de fuerzas

La resistencia a la tracción promedio del MDF en condición seca y espesores de 6 a 25mm es de 17.64 MPa (Martinez & Calil, 2002). Y los valores caracteristicos para el diseño de la traccion perpendicular es mucho menor que la traccion paralela (Giuliano, 2009). Con base en estos datos y el uso de *Solidworks simulation*, se analiza el diseño de la placa superior de la plataforma robotica. Siendo esta la que soportará directamente las cargas externas del robot.

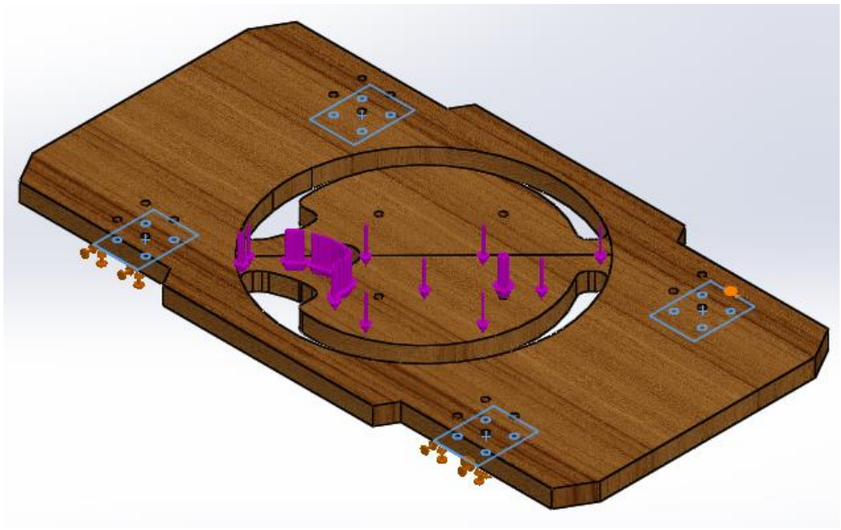


Figura 66. Soporte fijo- cuadros color azul, dirección de fuerza aplicada – flechas moradas. (Autor)

Para este análisis, se ubicó la carga a uno de los extremos donde se genera el mayor torque y así conocer cuál es la presión aproximada que se puede soportar antes del fallo. Debido a lo cual se hacen pruebas con pesos de 1 a $4kg$ o de $10\ a\ 40N$, tomando como referencia el valor de la gravedad de $10kg/m2$.

Se recogieron los siguientes datos con base en las caracteristicas del MDF:

Tabla 20. Lista de variables para la creación de material MDF en Solidworks, ("Medium Density Fiberboard (MDF) :: MakeItFrom.com," 2018)

VARIABLE	VALOR
EL RADIO DE POISSON	0,03
DENSIDAD DE MASA	788kg/m^3
MODULO DE ELASTICIDAD	580000 psi

A continuacion se presenta en las *figuras 69-72* el comportamiento del MDF con la colocacion de un peso de 1Kg hasta 4Kg respectivamente, lo que es igual a una aproximacion de fuerza aplicada de 1N a 4N

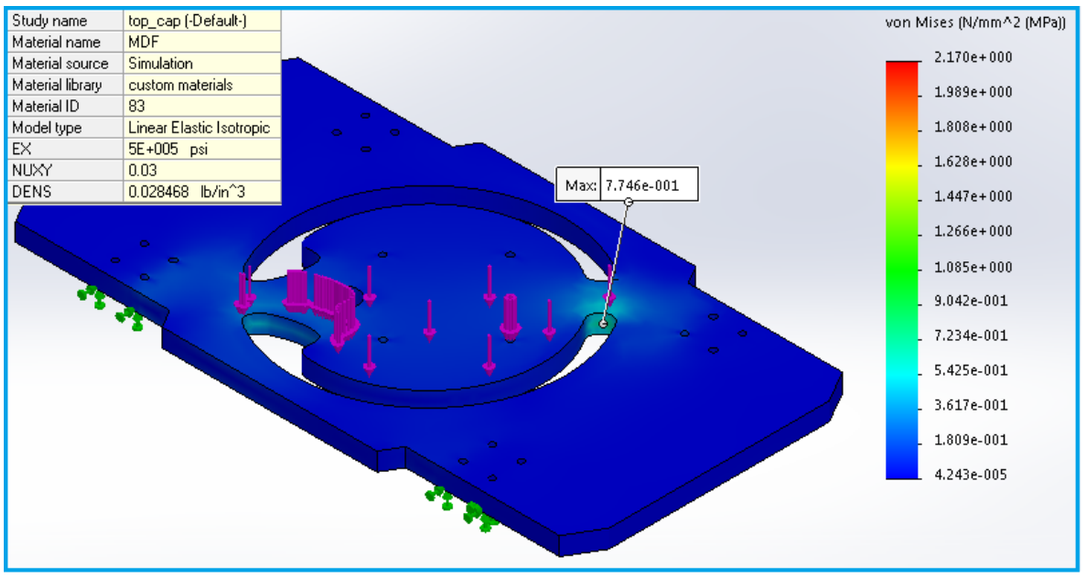


Figura 67. Estado de la pieza al aplicar una fuerza de 10N. (Autor)

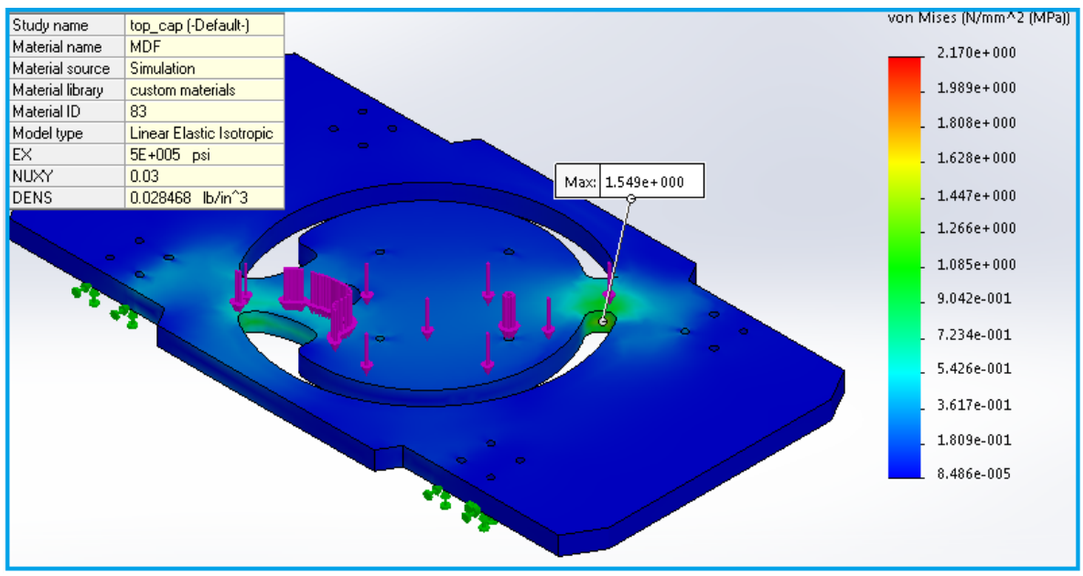


Figura 68. Estado de la pieza al aplicar una fuerza de 20N. (Autor)

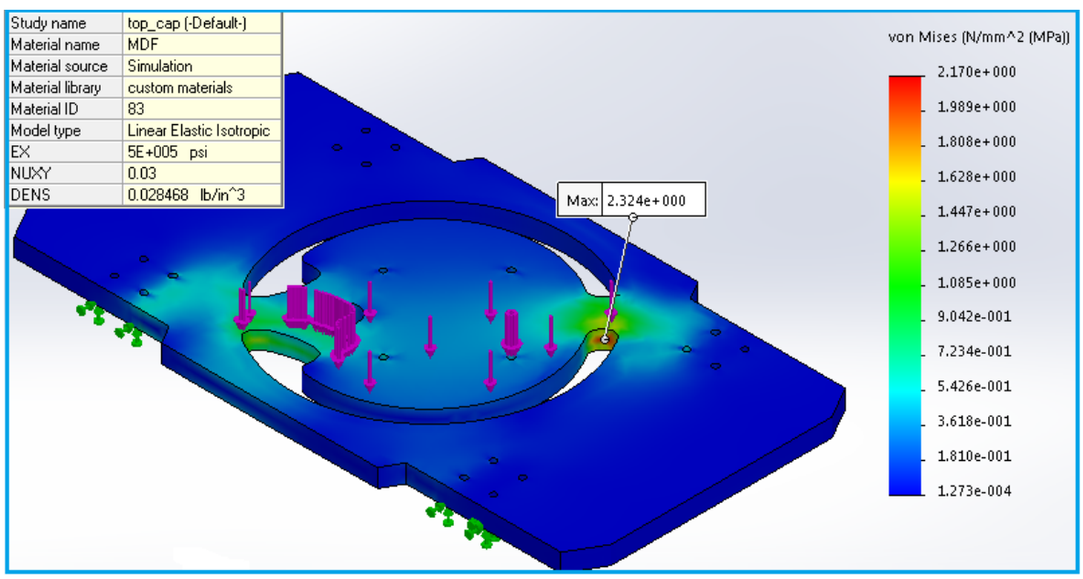


Figura 69. Estado de la pieza al aplicar una fuerza de 30N. (Autor)

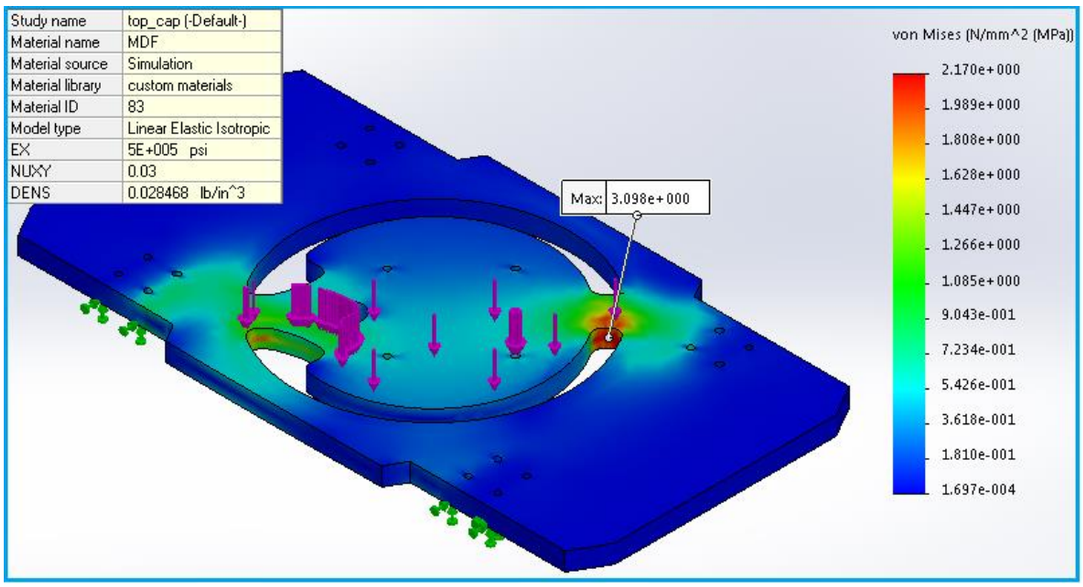


Figura 70. Estado de la pieza al aplicar una fuerza de 40N. (Autor)

De lo anterior se deduce que el material puede sorportar sin romperse hasta 3kg de peso en sus zonas mas criticas. Mas halla de este peso la plataforma robotica puede presentar rompimiento por fatiga. Cabe mensionar que el análisis se realizó para cargas dinámica. De otra forma, si se aloja una carga estatica, entonces el valor que puede llegar a soportar la plataforma aumenta hasta dos veces el valor aceptable para una carga dinámica. Esto debido a que con una carga estatica los pesos estarán equilibrados homogeneamente sobre toda la base sin ejercer un torque excesivo.

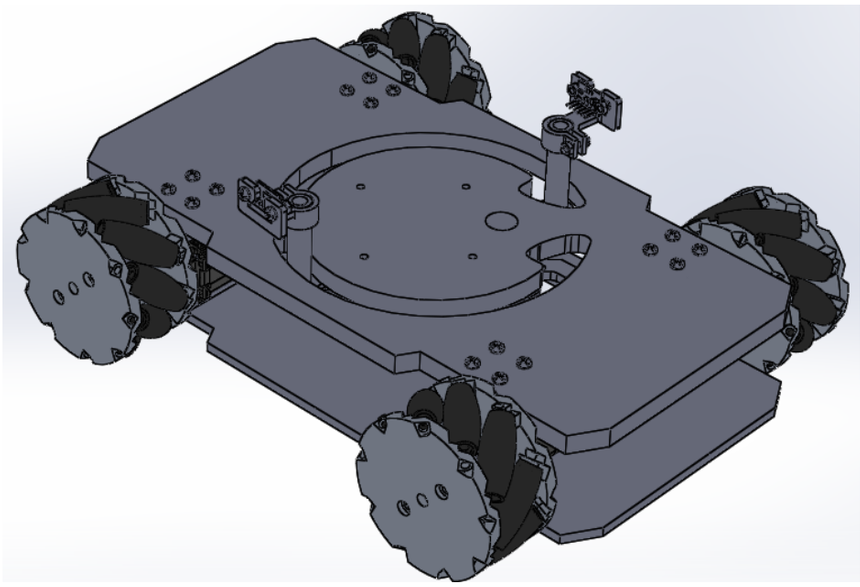


Figura 71. Propuesta final - Plataforma Robótica, Solidworks. (Autor)

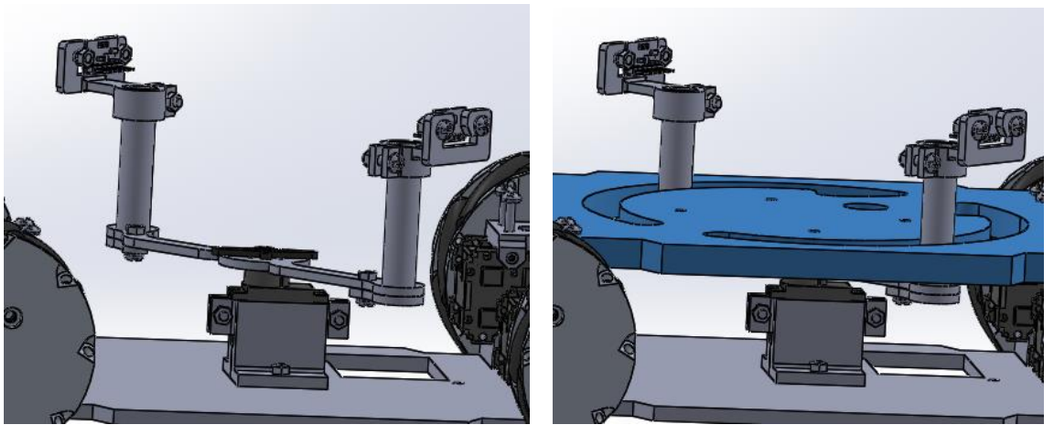
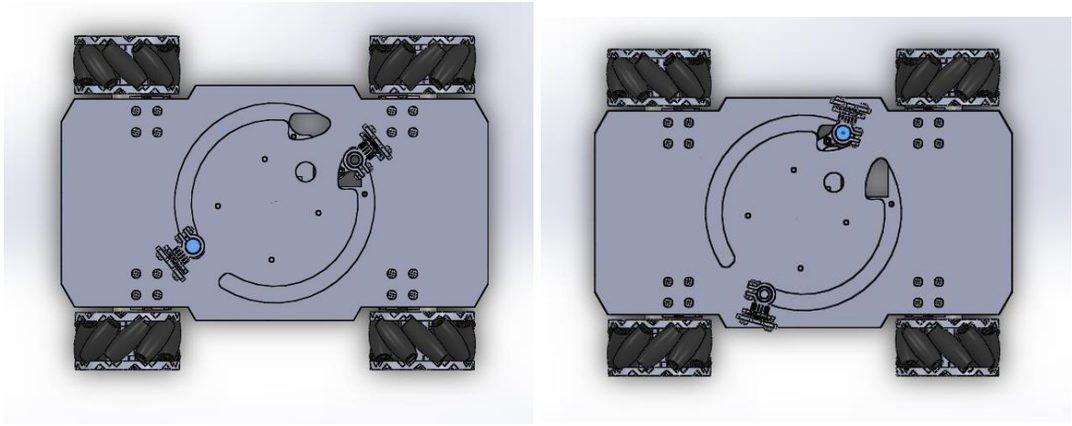


Figura 72. Mecanismo de detección de obstáculos, Un sensor dispuesto a 180° uno con respecto al otro. (Autor)

En esta condición de diseño, el soporte que se usa para dar rigidez al espacio libre de la plataforma limita del movimiento de barrido de 180° que es el ideal para cubrir los 360° de detección. Lo que quiere decir que el barrido total se reduce a 340°. No obstante, de acuerdo a las pruebas de rango realizadas, esta falencia no se hace muy evidente gracias al rango de medición de 25° del sensor, logrando reducir y hacer casi nulo el espacio donde no se censa directamente. Esto se ilustra en la siguiente figura.



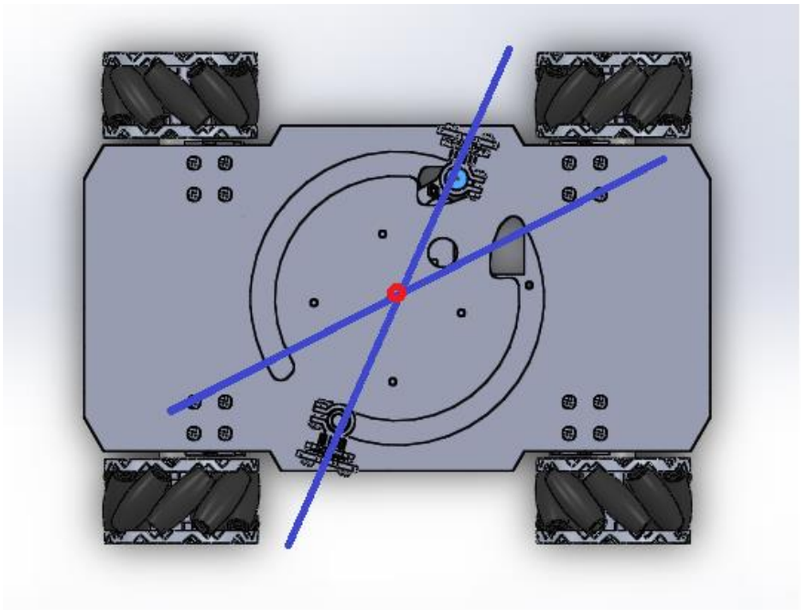


Figura 73. Movimiento límite del mecanismo para cada giro. Líneas azules. Solidworks. (Autor)

A continuación se muestra el soporte empleado para los sensores *figura 78*. La altura a la cual se ubicaron los sensores con respecto al suelo, permite censar obstáculos de hasta 100mm de altura.

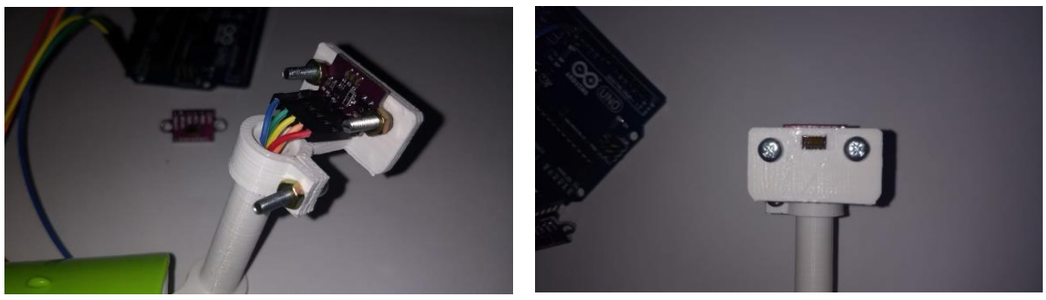


Figura 74. Parte del mecanismo de movimiento final. (Autor)

Las imágenes a continuación, muestran el diseño final en vista inferior, donde se observa la ubicación de la placa NodeMCU V1, la cual, nos permite la conexión vía WIFI y es bastante asequible.

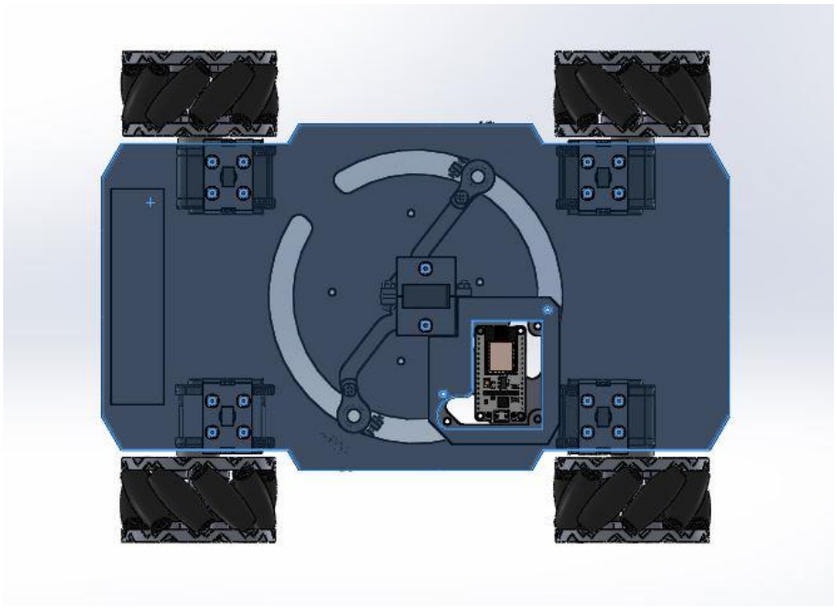


Figura 75. Vista inferior. (Autor)

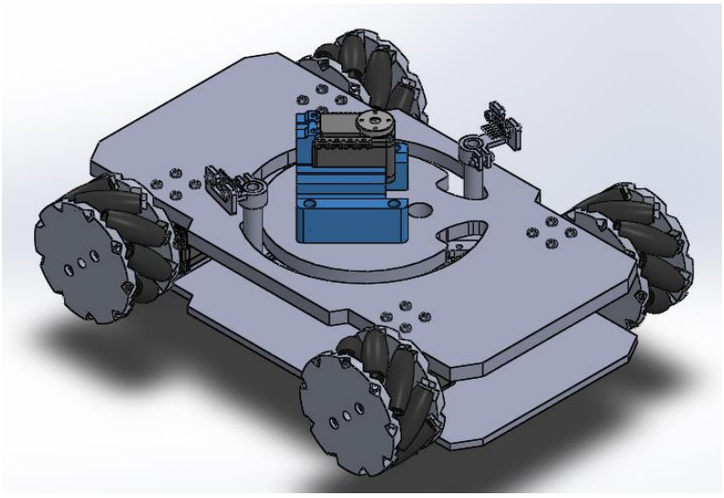


Figura 76. Vista isométrica – soporte inicial del brazo robótico. Carga dinámica. (Autor)

4.4 Sistema de control

En este apartado se explorará el uso de la placa de control, la cual una vez conectada a una red WIFI, pueda enviar la distancia y ángulo de los obstáculos hasta el dispositivo háptico y del mismo modo este envíe las coordenadas de posición a la que el usuario desea que el robot se dirija. *Ver figura.*

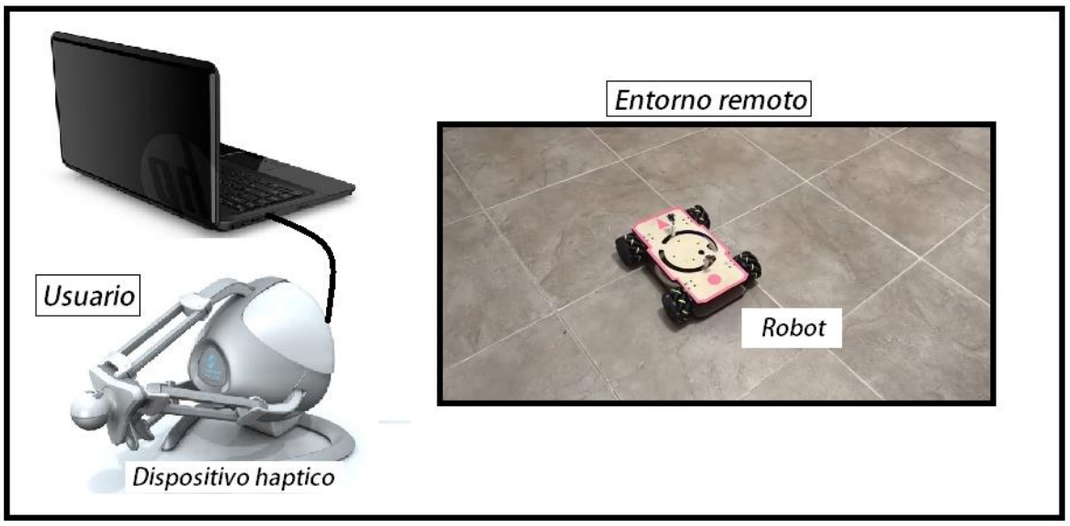


Figura 77. Mando (usuario) – Robot. (Autor)

El envío y recepción de los datos se llevaran a cabo a través del protocolo UDP que, como se explicó anteriormente es uno de los protocolos que nos permiten comunicar dos terminales a través de una red.

Por un lado se tiene la placa de control (robot) y por el otro, se tiene el dispositivo de control háptico (mando). La forma en que se puede programar el robot es a través de la *IDE de arduino* – allí se incluirán además, las librerías necesarias para operar los sensores, servomotores y, protocolos de comunicación. Por otra parte, la forma en que se puede programar el mando, es a través de *VREP* empleando librerías de *CHAI3D.*

Por lo anterior se hace necesario tener nociones claras del entorno de trabajo de cada una de estas plataformas y abordar cómo se llevó a cabo el cumplimiento de los objetivos del proyecto.

4.4.1 IDE de arduino

El software Arduino de código abierto (IDE) permite escribir código y subirlo a la placa. Se ejecuta en Windows, Mac OS X y Linux. El entorno está escrito en Java y se basa en el procesamiento y otro software de código abierto. Este software se puede usar con cualquier placa Arduino y por consiguiente existen librerías para programar la ESP en este mismo entorno.

Una vez se dispone de la placa ESP8266 y el software de arduino, se procede a la instalación de la librería e inclusión de la placa en el gestor de tarjetas de arduino. [https://github.com/esp8266/Arduino/tree/master/libraries/ESP8266WiFi]

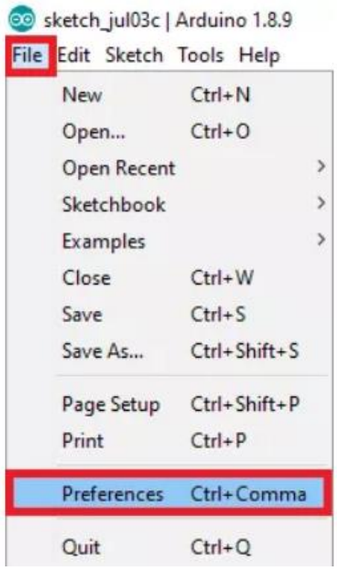


Figura 78. Instalación ESP8266 en Arduino (paso 1)

El siguiente paso es para facilitar la instalación y actualización de las plataformas Arduino. Para proporcionar soporte de instalación al gestor de tarjetas para una plataforma específica. En nuestro caso es la ESP8266 con su respectivo archivo de índice con formato JSON.

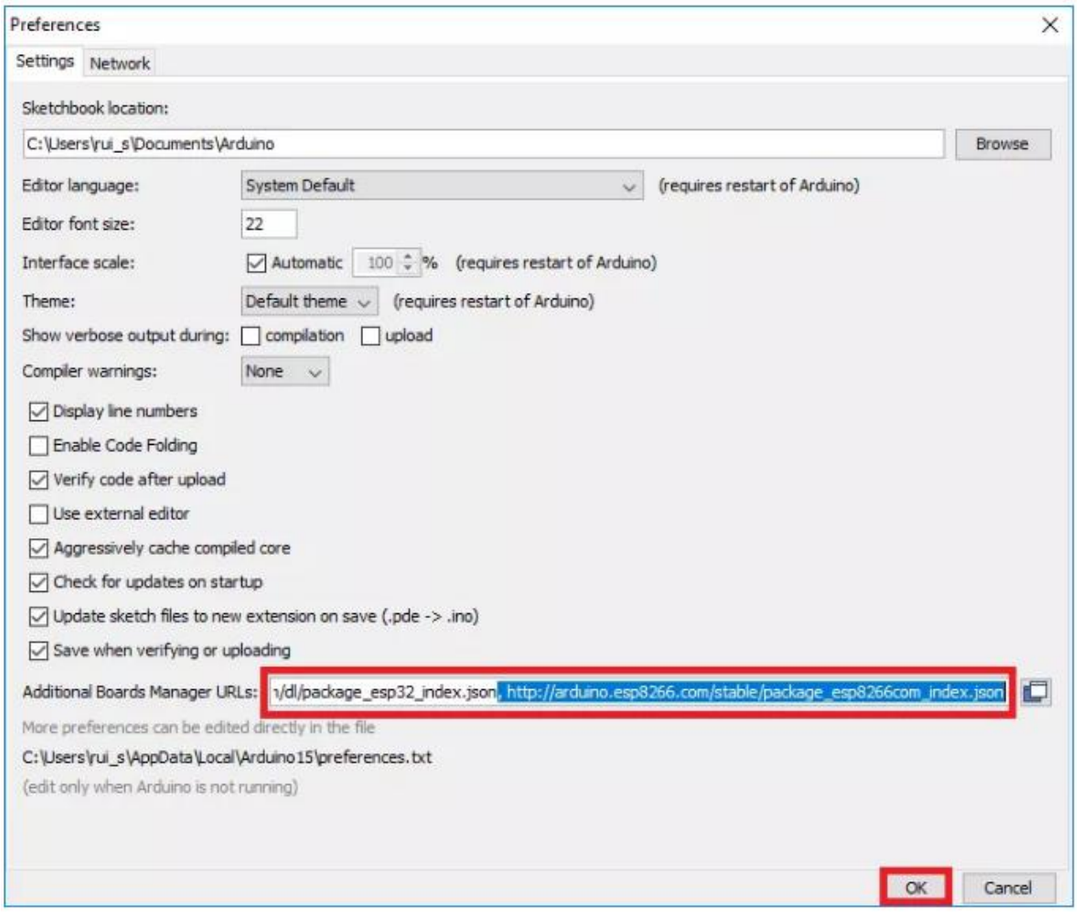


Figura 79. Instalación ESP8266 en Arduino (paso 2)

Si ya aparecen por defecto las URL de las placas ESP32, se puede separar las URL con una coma de la siguiente manera:

```
https://dl.espressif.com/dl/package_esp32_index.json,
http://arduino.esp8266.com/stable/package_esp8266com_inde
x.json
```

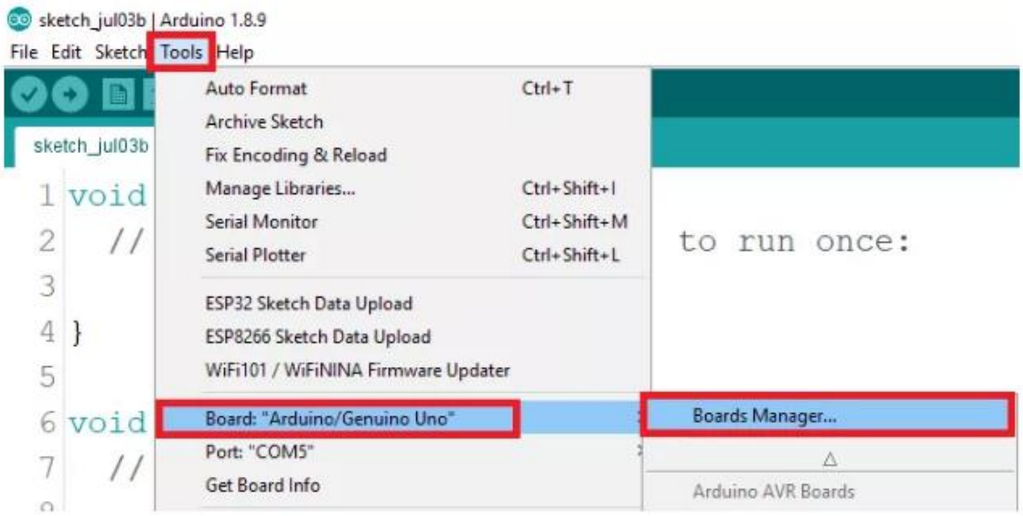


Figura 80. Instalación ESP8266 en Arduino (paso 3)

Figura 81. Instalación ESP8266 en Arduino (paso 4)

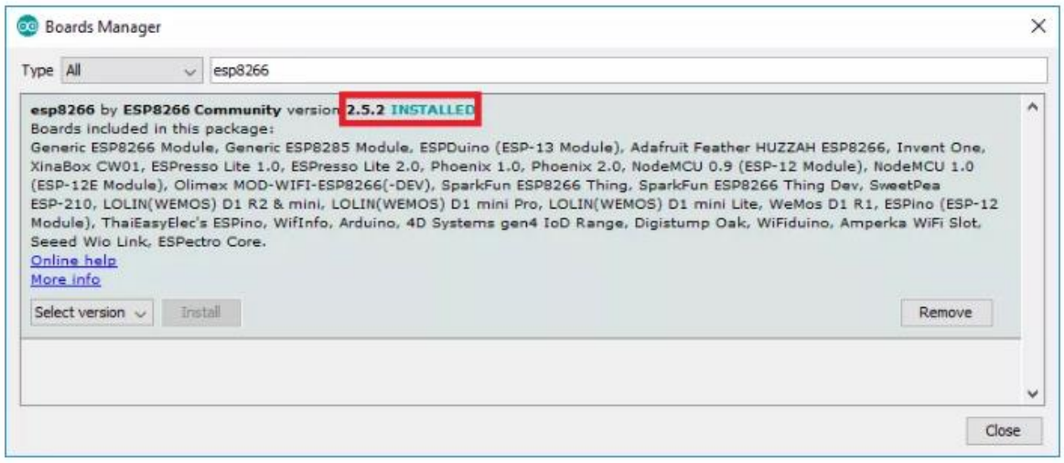


Figura 82. Instalación ESP8266 en Arduino (paso 5)

Ahora ya es posible utilizar todas las bondades de placa en el entorno de Arduino. El último paso será, incluir las demás librerías para el manejo de la API (*Application Programming Interface*) de los sensores y actuadores. *Ver figura*

Figura 83. Gestor de librerías Arduino

4.4.1.1 Entorno de programación:

Para cargar un nuevo código en el Arduino, se necesita tener acceso al código que puede pegar en el programador, o escribir nuestro propio sketch o boceto utilizando el lenguaje de programación Arduino.

Un sketch de Arduino generalmente tiene cinco partes: un encabezado que describe el sketch y su autor; una sección que define variables; una rutina de configuración que establece las condiciones iniciales de las variables y ejecuta código preliminar; una rutina de bucle, que es donde agrega el código principal que se ejecutará repetidamente hasta que deje de ejecutar el boceto; y una sección donde puede enumerar otras funciones que se activan durante la configuración y las rutinas de bucle. Todos los bocetos deben incluir la configuración y las rutinas de bucle. *Ver Figura*

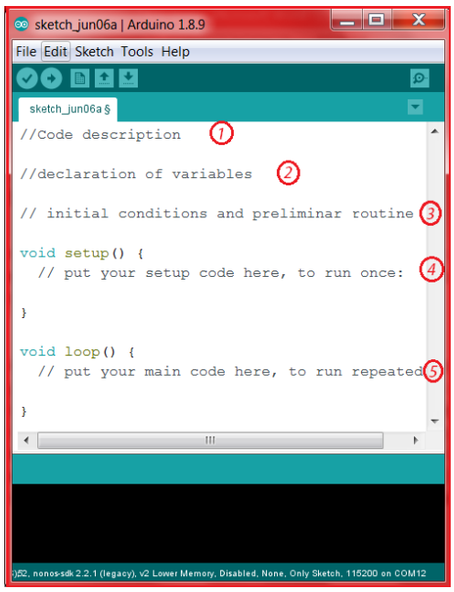


Figura 84. Entorno de programación Arduino

4.4.1.2 Explicación del código

- **Librerías:**

- VL53L0X.h

Esta es una biblioteca para Arduino IDE que ayuda a interactuar con el sensor de distancia de tiempo de vuelo VL53L0X de ST. La biblioteca simplifica la configuración del sensor y la lectura de los datos del rango a través de I²C. [https://github.com/pololu/vl53l0x-arduino]

- Servo.h

Esta biblioteca permite que una placa Arduino controle los servomotores RC (hobby). Los servos tienen engranajes integrados y un eje que se puede controlar con precisión. Los servos estándar permiten que el eje se coloque en varios ángulos, generalmente entre 0 y 180 grados. Los servos de rotación continua permiten que la rotación del eje se ajuste a varias velocidades. [https://www.arduino.cc/en/Reference/Servo]

- WIFI.h – WIFIUdp.h

Esta biblioteca establece la espera de paquetes UDP en un puerto local. Cuando se recibe un paquete válido, se envía un paquete de confirmación al cliente en un puerto saliente especificado. Los paquetes perdidos no se recuperan. [https://www.arduino.cc/en/Tutorial/WiFiSendReceiveUDPString]

- AX12A.h

Esta biblioteca le permite controlar los servomotores *Robotis* que utilizan un protocolo serie semidúplex personalizado. Puede controlar los modelos TTL directamente desde Arduino, sin ningún hardware adicional, utilizando hardware o software UART. La velocidad de comunicación de hasta 1 MBd es compatible con el hardware en serie. Las funciones más útiles (velocidad, posición, modo rueda / articulación) se proporcionan a través de una interfaz de alto nivel muy simple [https://github.com/jumejume1/AX-12A-servo-library]

- **Variables:**

Tabla 21. Variable globales Arduino

Variable globales	Función
XSHUT_pin1	Pin del XSHUT del sensor 1
XSHUT_pin2	Pin del XSHUT del sensor 2
DirectionPin	Sentido de la transmisión (leer/escribir)
BaudRate	Taza de baudios del Dynamixel
const char* ssid	Nombre de la red WIFI
const char* password	Clave de la red WIFI
unsigned int localUdpPort	Puerto local
char incomingPacket[255]	Buffer para recibir datagramas
int avatar_position_x	Ubicación del avatar en eje x
int avatar_position_y	Ubicación del avatar en eje y
VL53L0X sensor1	sensor1
VL53L0X sensor2	sensor2
Servo sensor2	Microservo movimiento de sensores
int Angulo = 0	Ángulo sensores A
int Angulo1 = 0	Ángulo sensores B

- **Void setup:**

Se conecta la placa a una red WIFI, ingresando el nombre de la red y contraseña: WiFi.begin(ssid, password) tener en cuenta las demás líneas de código en cada explicación de línea general. Posteriormente se inicia el puerto local para transmisión de datos UDP: Udp.begin(localUdpPort) y se inicia el puerto serie para propósitos de depuración *Serial.begin(115200).*

Se configuran los tiempos de respuesta al error de los sensores VL530X uno y dos *sensor1.setTimeout(500)* y se escoge el modo toma de datos continuos: *sensor1.startContinuous().* De acuerdo a la versión de librería en cuestión, la configuración inicial de los sensores VL530X puede cambiar, se recomienda revisar los ejemplos para cada rango. En mi caso fue necesario declarar los pines XSHUT como salida y fijar la ID a uno de los sensores: *sensor1.setAddress(sensor1_newAddress), esto para evitar errores de compilación.*

Se estable el movimiento giro continuo a cada uno de los servomotores Dynamixel: *ax12a.setEndless(i, ON)* al igual que la taza de baudios a 1MBd.

- **Void loop:**

El servidor recibe los paquetes de datos del cliente como cadena de caracteres y son almacenados en buffer *incomingPacket*, se procede a separarlos por medio de la función strtok: *char *ptr = strtok (incomingPacket, " ")* la cual identifica dentro de la cadena de caracteres donde está la posición de la tecla espacio y toma los valores anteriores a ella, posteriormente este dato es convertido a numero con la función *atoi* y guardado en la variable avatar_position_x. De igual forma, se procesa el mismo datagrama pero esta vez con el parámetro NULL: *ptr = strtok (NULL, " ")* el cual identifica, la tecla espacio dentro de la cadena de caracteres y toma los valores después de esta para posteriormente hacer la conversión a número y guardar el valor en la variable *avatar_position_y*.

Teniendo los valores numéricos de la posición del dispositivo háptico y partiendo de que el valor en X, se recibe con valores negativos y positivos (i.e., de -145 a 145) lo cual representa el movimiento hacia adelante (x negativo) y hacia atrás (x positivo) se procede a mapear esos valores para convertirlos en la velocidad de los servomotores: *int mov1 = map(avatar_position_x, 0, -145, 0, 800)* lo que quiere decir que la velocidad máxima los Dynamixel será de 800. Entonces de -140 a 0 (i.e., movimiento de dispositivo háptico hacia adelante) los valores que toma la variable mov1 serán positivos (de 0 a 800) y de 0 a 145 los valores de la variable *mov1* serán negativos (0 a -800). Con esto en mente, se realizan los condicionales de movimiento hacia adelante y hacia atrás. Para esto se establecen rangos en que los valores de *avatar_position_x* y *avatar_position_y* indiquen que el robot no se mueve. *Ver figura*.

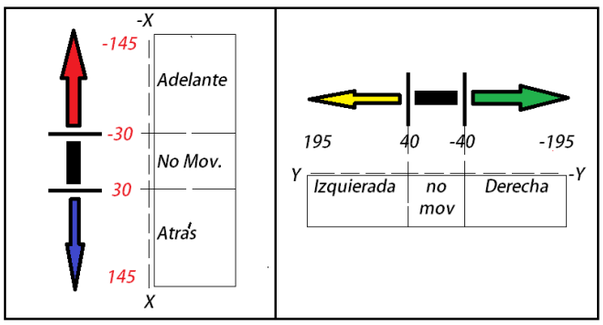


Figura 85. Parametrización de movimientos laterales, adelante y atrás del robot. (Autor)

Si avatar_position_x es mayor que 30 y avatar_position_y está en el rango de no movimiento, entonces el robot se moverá hacia adelante. Si avatar_position_y y avatar_position_x están en rango de no movimiento, entonces el robot no se moverá. Si avatar_position_y es menos que -30 y avatar_position_x está en el rango de no movimiento, entonces el robot se moverá hacia atrás.

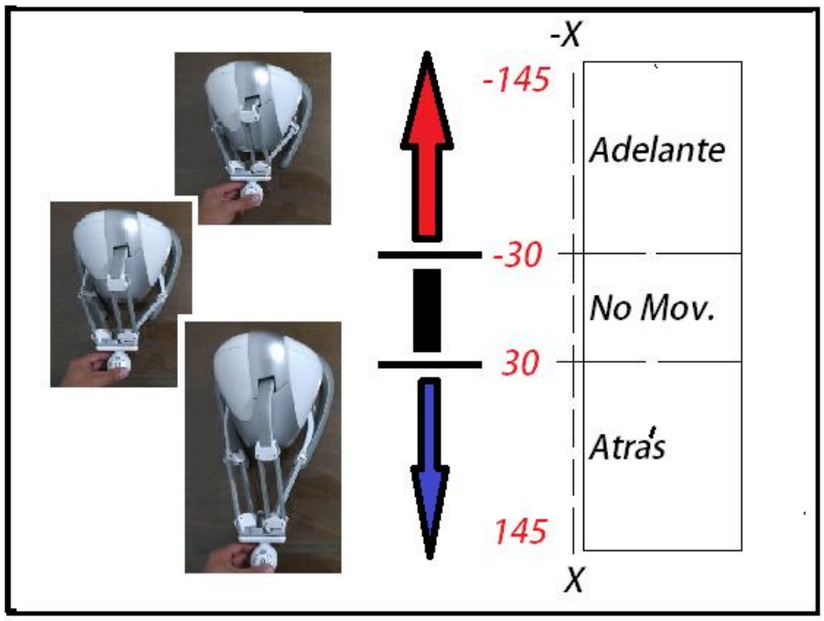


Figura 86. Movimiento del dispositivo háptico - Robot hacia adelante y hacia atrás. (Autor)

Para los movimientos laterales (eje y) se determina la velocidad de los motores con el mapeo de avatar_position_y, almacenado en la variable *mov2:* int *mov2 = map(avatar_position_y, 0, -195, 0, 800) siendo* la lógica de programación la misma.

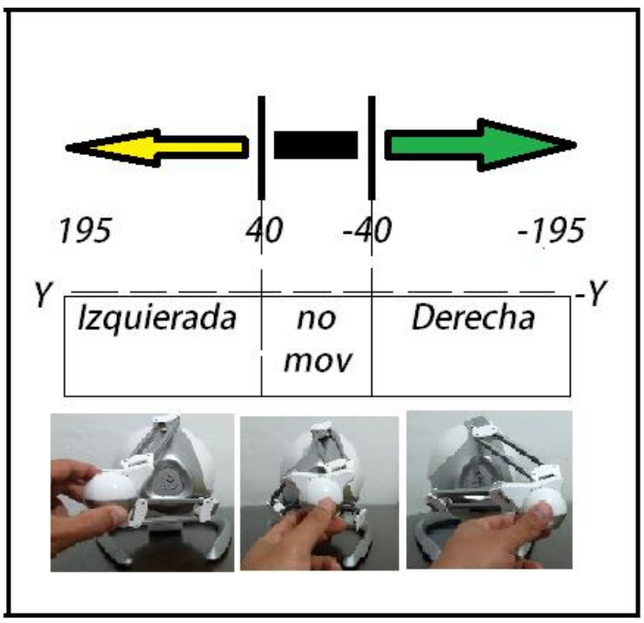


Figura 87. Movimiento del dispositivo háptico - Robot movimientos laterales. (Autor)

Para los giros en el mismo eje hacia la izquierda y hacia la derecha, se establece un rango tanto en X como en Y.

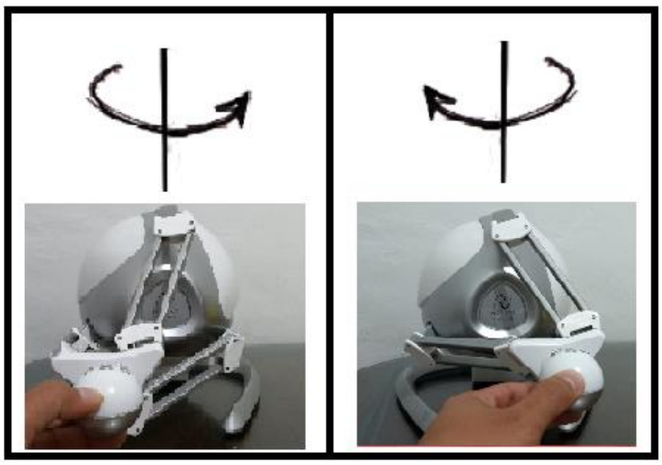


Figura 88. Movimiento del dispositivo háptico - Robot rotación hacia la recha e izquierda. (Autor)

A continuación, las funciones a() y b() son los movimientos y toma de datos de los sensores, es decir, a() es el barrido en sentido horario y b() es el barrido antihorario. Tomando en cuenta que la función a(), es similar en gran medida a la función b(), solo explicaremos la primera.

La función para el barrido en sentido horario, ejecuta un bucle de 1 a 168 en rangos de 12 grados: *for (Angulo = 1; Angulo < 168; Angulo += 12) para un total de 14 sectores, es decir, que como son dos sensores, en cada sector se toma los datos de cada sensor, tendremos un total de 28 distancias y 28 ángulos.*

Inicialmente, se crean 4 vectores de 14 posiciones, 2 para guardar la distancia y ángulo del sensor 1 y, 2 para guardar la distancia y ángulo del sensor 2. Cuando *Angulo* es igual a 12 se lee la distancia del *sensor1* en la variable *distance1*: *int distance1 = sensor1.readRangeContinuousMillimeters()* y se condiciona para detectar distancias menores a 400mm. Una vez se detecta una distancia en ese rango, se guardan: distancia y el ángulo: *dist1[index1] = distance1 y ang1[index1] = Angulo,* caso contrario, la distancia y ángulo serán cero. Seguidamente se toma la distancia del sensor 2 en la variable *distance2* y se hace el mismo procedimiento de almacenamiento en los vectores *dist2* y *ang2.*

Una vez tomados los datos de ambos sensores en cada iteración, tendremos en total 28 datos de las distancias y 28 ángulos. Ver figura.

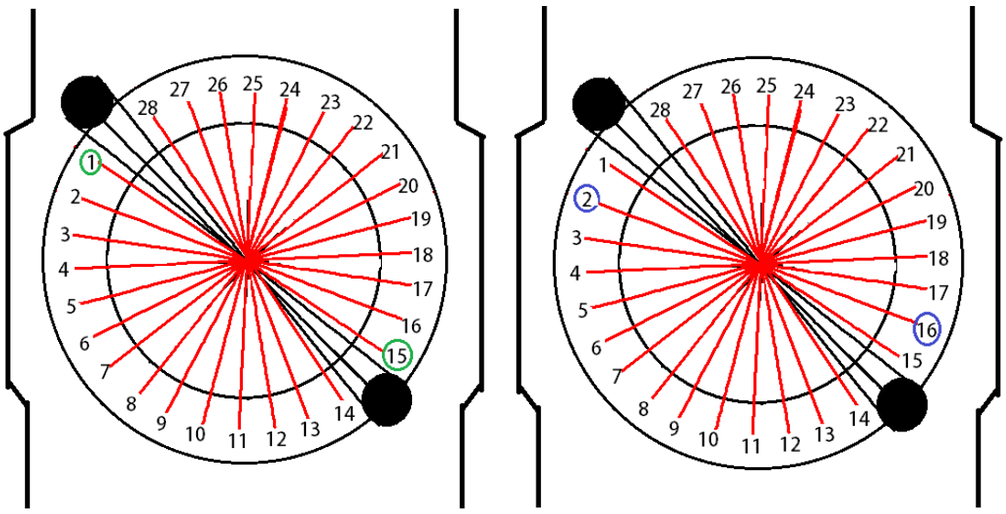


Figura 89. Total de sensores en cada barrido. (Autor)

Posteriormente, estos datos son enviados a través de UDP a la misma dirección IP y puerto del cual se recibió la posición del avatar. Para esto, se inicia el paquete con: *Udp.beginPacket(Udp.remoteIP(),Udp.remotePort())* y se declara el vector tipo char donde se almacenará toda la información: *char data[10]* y con la función *sprintf* se arman los datos de la siguiente manera*: sprintf (data,"%u,%u,%u…(48 veces)",dist1[1],dist1[2]…(hasta dist1[14]),ang1[1], ang1[2]…(hasta ang1[14]), dist2[1],dist2[2]…(hasta dist2[14]),ang2[1], ang2[2]…(hasta ang2[14]),* y finalmente se envían con la función: *Udp.write(data).*

4.4.2 CHAI3D

CHAI3D es un framework de renderizado háptico multiplataforma de código abierto, utilizado en una gran cantidad de proyectos de investigación y producción, en áreas tan diversas como juegos, simuladores, software educativo, arte interactivo, visualización científica y aplicaciones médicas. Al admitir diferentes tipos de dispositivos de retroalimentación de fuerza, CHAI3D ofrece una interfaz única para diseñar e implementar fácilmente soluciones táctiles informáticas avanzadas.

4.4.3 VREP - CoppeliaSim

CoppeliaSim, anteriormente conocido como V-REP, es un simulador de robot, con un entorno de desarrollo integrado, basado en una arquitectura de control distribuido, es decir, cada objeto/modelo puede controlarse individualmente a través de un script, un plugin, ROS o un nodo BlueZero, un cliente API remoto o una solución personalizada. Esto hace que CoppeliaSim sea muy versátil para aplicaciones con robots. Los controladores se pueden escribir en C / C ++, Python, Java, Lua, Matlab u Octave.

La descarga de la versión educativa con todas las funciones se puede realizar desde Copeliarobotic.com/downloads. Una vez abierto, puedes dirigirte a la opción archivos > escenas para ver ejemplos y darte una idea de cómo están estructuradas las escenas. En lo personal te recomendamos la escena *hapticRobot* y hacer pequeñas pruebas conectando el dispositivo háptico. Con esto en mente, se pasara

a la explicación de los conceptos más importantes para profundizar un poco más en este software.

4.4.3.1 Embedded script

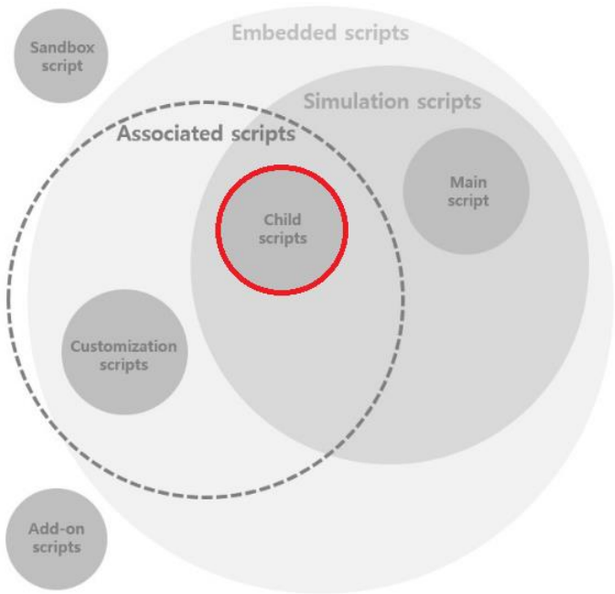


Figura 90. Jerarquía del embedded script, (CoppeliaSim, 2017)

NOTA: en la *figura 91* se resalta el *child script* porque es ahí donde se escribe y edita los script Lua, y por lo tanto, es en este punto donde se hará mayor énfasis.

El *embedded script* es uno de los tantos métodos que permiten customizar un modelo o escena de tal forma que el robot ejecute las tareas que le sean configuradas. Esta configuración consiste en programar scripts Lua. Lua es un lenguaje de programación basado en scripts, muy ligero y ampliamente utilizado en el entorno de copeliaSiim para los embedded scripts en general.

- **Simulation scripts o script de simulación**

Los scripts de simulación son scripts que se ejecutan solo durante la simulación y que se utilizan para personalizar una simulación o un modelo de simulación. El bucle de simulación principal se maneja a través del script principal, y los modelos / robots se controlan a través de guiones secundarios.

Una escena se compone de un script principal (generalmente se recomienda no editarlo) y de uno o varios child scripts (editables). Cuando se abre una nueva escena, el script principal aparece por defecto (*ver figura 92*). El child script se agrega a los objetos (e.g., obstáculos, esferas, cubos) o viene incluido en robots con rutinas predeterminadas (*ver figura 92)*.

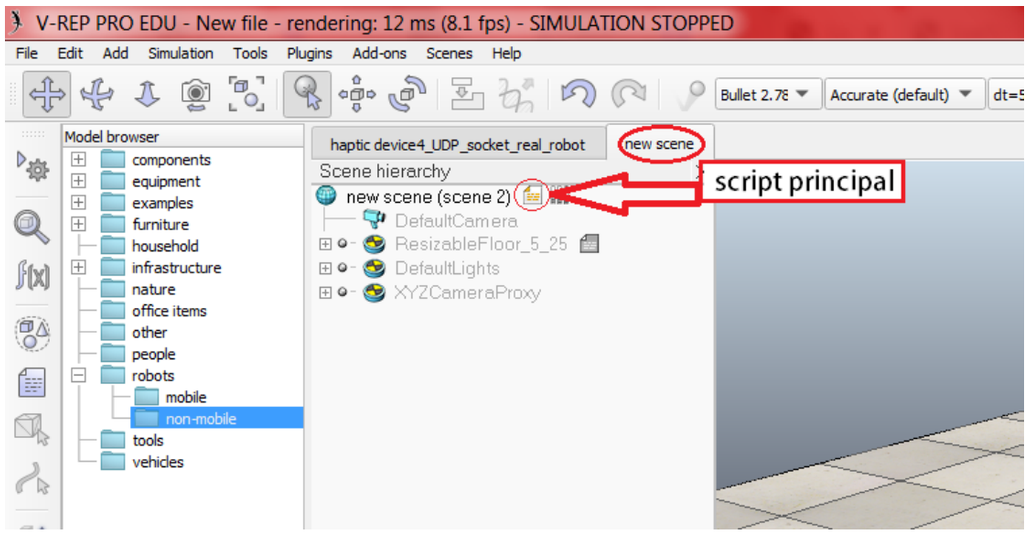


Figura 91. Nueva escena - Main script

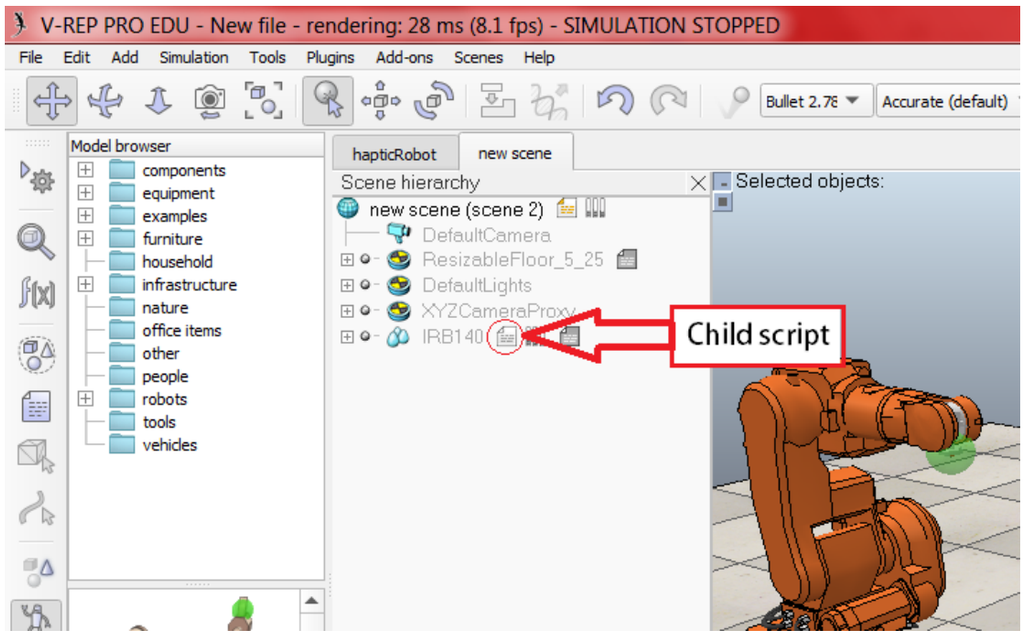


Figura 92. Child script asociado al Robot IRB140. (Autor)

Existen dos tipos de child script, no obstante, debido a que uno de ellos presenta fuertes desventajas con respecto al otro, se decide profundizar solo uno de ellos. Y en su defecto es el que utilizaremos para escribir nuestro código.

- **Non-threaded child script:**

Los *Non-threaded child script (Scripts secundarios sin hilos, en español)* contienen una serie de funciones que una vez llamadas deben ejecutar esa tarea y devolver la acción de control, de lo contrario la simulación entera se detendrá. CopeliaSim recomienda el uso de este tipo de child script sobre el *threaded child script* (o, script secundario con hilos, en español). (CoppeliaSim, 2017)

Este tipo de child script debe estar segmentado en 4 funciones principales:

La función de inicialización: ***sysCall_init.*** Esta parte solo se ejecuta solo una vez dentro del código. Esta función puede ser incorporada al inicio o en la mitad de las cuatro funciones principales. Se debe tener en cuenta que los objetos asociados con un child script pueden ser copiados y pegados en la escena en cualquier momento, incluso cuando la simulación está siendo ejecutada.

La función de actuación: ***sysCall_actuation.*** Esta función se ejecutará en cada siclo de la simulación, durante la fase de actuación y aquí van principalmente incluido los comandos para accionar los actuadores de la escena.

La función de censado: ***sysCall_sensing.*** Esta función se ejecutará en cada siclo de la simulación, durante la fase de censado. Aquí van principalmente incluido los comandos para ejecutar los sensores de la escena.

La función de restauración: ***sysCall_cleanup***. Esta parte se ejecutará una vez justo antes de que termine una simulación, o antes de que se destruya el script debido a algún error de compilación.

Hasta el momento hemos aprendido cuales son las funciones presentes en cualquier child script tipo threaded y que podemos programar de cada una de ellas. Ahora, nos resta poner manos a la obra.

4.4.3.2 Explicación del código

Antes de iniciar, es necesario instalar los driver para el dispositivo háptico de acuerdo a la referencia. En nuestro caso se dispone del dispositivo háptico Novint Falcon de 3GDL y se hizo uso de la SDK que ofrece Force Dimension para Windows. De la misma forma se debe instalar la librería CHAI3D. Se recomienda revisar la documentación de chai3d.org/download/releases y descargar la versión más reciente. Aquí, más información: hapticshouse.com/pages/drivers

- ***sysCall_init:***

Se inicializa el plugin CHAI3D con la función *sim.getModuleName* y *simCHAI3D.start.* Debido a que solo se trabajará el eje X, Y, se establecen las restricciones del plano para trabajo en 2D: simCHAI3D.addConstraintPlane.

- **sysCall_cleanup:**

Una vez se detiene la simulación el programa se reinicia: simExtCHAI3D_reset

- ***sysCall_actuation:***

Se toma la posición x, y del dispositivo háptico *p=simCHAI3D.readPosition(0)* y continuación se guardan en la variable avatar_location[1] y avata_location[3]. Estas variables se declaran como variable locales para poder ser utilizadas en la función sysCall_sensing.

- **sysCall_sensing:**

A continuación se exponen las líneas de código principales, por lo tanto, se invita a un análisis más detallado en la sección de anexos código VREP y utilizar esta explicación como complemento.

Siempre y cuando el plugin de CHAI3D este activo, se procede con el envío y recepción de paquetes UDP. Para esto se crea la variable socket: *local socket = require("socket")* y se especifica la dirección IP y el puerto del servidor *udp:setpeername('192.168.43.21', 7777).* Una vez definidos estos parámetros podemos enviar datos con la función *udp:send(msg),* No obstante, esta función solo permite del envío de valores String, por lo que no es posible enviar una tabla (i.e., vector) con X posiciones. Por tal motivo se decide concatenar ambos valores en un solo String y se anexar la tecla espacio en la String.

NOTA: Posteriormente en arduino se identifica la tecla espacio y se toman los valores antes y después de ella (i.e., x, y).

Tabla 22. Vector con las posiciones del avatar

Avatar_location[1]	Avatar_location[2]	Avatar_location[3]
123.454546645675		-12.33334546444

Tabla 23. Reducción de decimales e inclusión de la tecla espacio

Avatar_location[1]	Avatar_location[2]	Avatar_location[3]
123	“tecla espacio”	-12

Tabla 24. String final a enviar

msg
“123 -12”

Una vez enviado en datagrama al servidor (i.e., ESP8266), este responde de vuelta. Se utiliza la variable *data* para guardar la respuesta: *data = udp:receive()*. Los datos recibidos están igualmente en formato String pero separados en este caso por comas. Por lo tanto, cada String recibido, es analizado en un bucle que selecciona los datos a excepción de las comas, los convierte a número y los guarda en un vector: *for x in string.gmatch(data, regxEverythingExceptComma) do,* donde regxEverythingExceptComma = '([^,]+)', la función *r = tonumber(x)* convierte el valor a número y *table.insert(a, r)* anexa el valor de r en el vector a. Ahora el vector “a” tiene todos los datos en formato número enviados por el servidor, cuyos datos estan de la siguiente manera: *Ver tabla*

Tabla 25. Contiendo del String recibido

Contenido de “a”	String con [48] datos
Posición 1 hasta 14	Distancias del sensor 1
Posición 15 hasta 28	Ángulo del sensor 1
Posición 29 hasta 33	Distancia del sensor 2
Posición 34 hasta 48	Ángulo del sensor 2

Con base en estos datos, tenemos que hacer que la distancia al objeto sea inversamente proporcional a la fuerza. Por lo tanto, habiendo predefinido el eje X, Y del dispositivo háptico, *figura 95*, se debe definir el sistema coordenado para el robot. *Figura 96.*

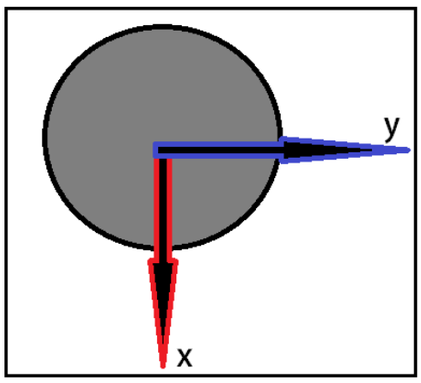


Figura 93. Sistema coordenado del dispositivo háptico. (Autor)

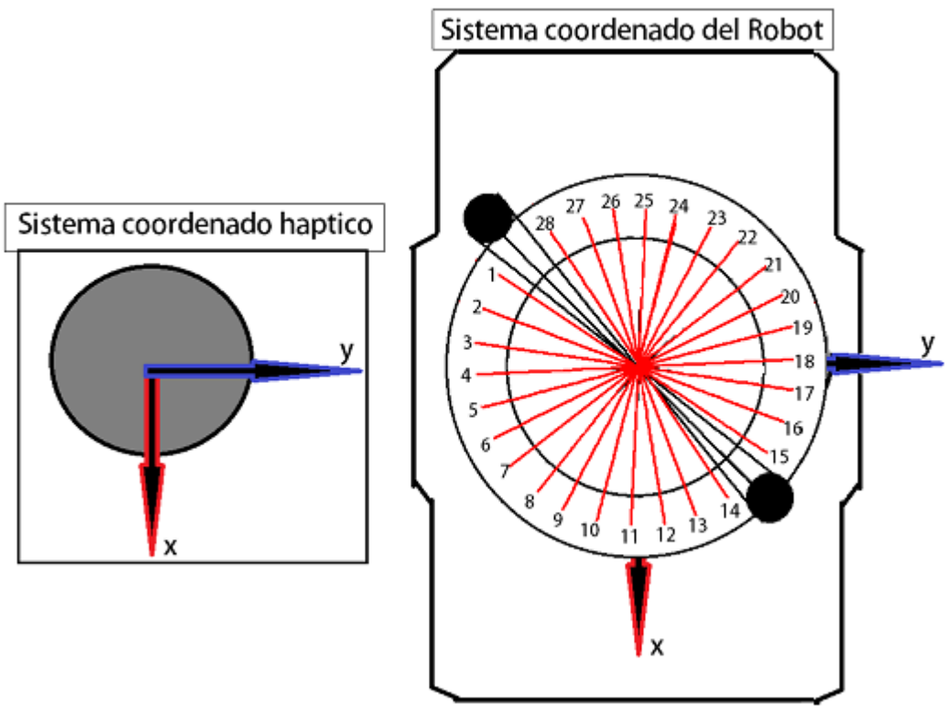


Figura 94. Representación sistema coordenado Robot – Mando. (Autor)

Lo que quiere decir que, si se detecta un obstáculo en el sensor 1, entonces, la componente (X, Y) quedaría de la siguiente manera.

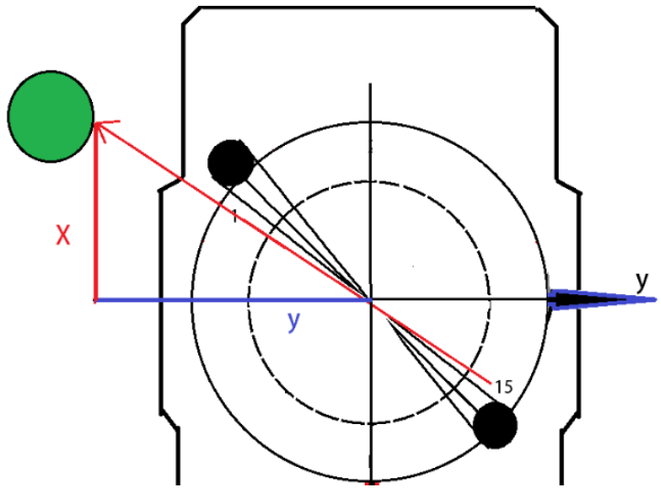


Figura 95. Verde: Obstáculo. Descomposición de la fuerza X, Y. (Autor)

Por lo tanto, cada distancia tendrá una componente en X, Y que se utilizará para reflejar una fuerza inversamente proporcional a la distancia, con la función: simCHAI3D.updateConstraint(obstacleConstr[i],{x,y,0},{0,0,0},0.25,0,f) siendo f la intensidad de la fuerza de retroalimentación.

CAPITULO V

5. RESULTADOS

Tras el montaje y programación del proyecto explicados con detalles en el capítulo 4, nos resta la realización de algunas pruebas y estudiar la respuesta en cada una de estas. Para esto, se decide poner en funcionamiento la plataforma robótica siendo teleoperada desde una distancia de 8 y 14 metros respectivamente para observar sus tiempos de respuesta y la sensación de control experimentada por el usuario.

Las pruebas se realizaron utilizando la opción de red *hotspot* en teléfono *Android*. Una vez conectado el robot y el portátil a esta misma red se procede a operar el robot a una distancia de 8m. *Ver figura.*

Figura 96. Prueba 1.A (Autor)

En la prueba 1.A – se teleopera la plataforma robótica a 8m de distancia en campo abierto mostrándose un control bastante aceptable y sin ningún tipo de interferencias o pérdida en la señal. El robot fue operado alrededor de 3 minutos mostrando incluso que en campo abierto esta distancia puede extenderse un poco más.

La prueba 1.B, fue realizada a una distancia de 14 metros durante el mismo periodo de tiempo y bajo las mismas condiciones, *ver figura.*

Figura 97. Prueba 1.B. (Autor)

No obstante, en este escenario, la comunicación mostraba un retraso un poco mayor con respecto a la prueba anterior, lo que valida que la distancia máxima promedio esta entre 8 a 10m en campo abierto.

Por otra parte se permitió que el robot fuera teleoperado en un entorno estructurado por diferentes usuarios entre los siguientes rangos de edades: Niño, adulto joven, adulto mayor. (**Prueba 2.A – 2.C**) Una vez explicado el sistema de mando se hicieron pruebas para observar el grado complejidad del control y la sensación experimentada por el usuario al encontrarse con un obstáculo al rededor del robot.

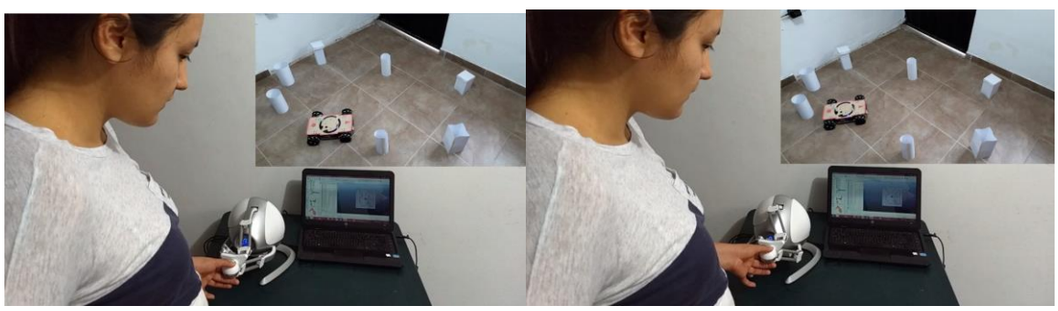


Figura 98. Adulto joven teleoperando el robot. (Autor)

Una vez se detectara un obstáculo a distancias menores de 400mm el usuario experimentaría dicha fuerza. El objetivo principal fue la manipulación del dispositivo háptico para dirigir el robot hacia la posición deseada y acercarse lo más posible hacia cualquiera de los obstáculos en la escena.

Figura 99. Niño teleoperando el Robot. (Autor)

Permitiéndole al usuario percibir la fuerza de repulsión inversamente proporcional a la distancia a la que se acercaba al objeto.

Figura 100. Prueba 2.C. (Autor)

De acuerdo a los parámetros establecidos se configuro previamente una fuerza de retroalimentación casi nula o de baja intensidad y, posteriormente una retroalimentación de alta intensidad (**Prueba 3**).

Figura 101. Prueba 3. (Autor)

A diferencia de las pruebas realizadas con los sensores con fuerza de retroalimentación casi nula (**Prueba 2.A – 2.C**), y la prueba realizada con fuerza de retroalimentación alta (**Prueba 3**), es que en este último, el control del robot se torna más complejo. Esto debido a que las fuerzas de repulsión “muy fuertes” no permiten entrar fácilmente en el rango de mando preestablecido para cada movimiento. En cuanto a la respuesta de control experimentada por los participantes, esta se describe como satisfactoria, aun cuando el tiempo desde que se envía la señal de movimiento y la acción, toma unos instantes en ser ejecutada. Esto es debido a la inclusión de 28 sensores que tienen que ser ejecutados individualmente en el bucle de barrido.

CAPITULO VI

6 CONCLUSIONES

Con base en el desarrollo y puesta en marcha de cada una de las etapas que se llevaron a cabo en este proyecto, se sacan a colación las conclusiones de mayor peso.

- ✓ En cuanto a las ruedas mecanum al ser un mecanismo que precisa de que cada rodillo este a 45 con respecto al eje de rueda, cualquier falla en uno de los ejes ocasionará que la rueda no gire o en su defecto, interfiera con el movimiento correcto de las mismas. Se comprobó que el algoritmo para hacer que el robot se dirija hacia cualquier dirección es sencillo en comparación a los movimientos que se le pueden programar y en comparación con los demás estructuras de locomoción.

- ✓ El protocolo de comunicación UDP, a pesar de ser un protocolo que no tiene en cuenta el orden de envío y pérdida de datos, es esto lo que lo hace bastante rápido e ideal en nuestro proyecto. Ya que se precisaba de un sistema que no tuviera una latencia muy elevada para teleoperar la plataforma robótica y que se lograra apreciar una fuerza inversamente proporcional a la distancia casi en tiempo real. Fue algo que se pudo observar en la prueba de movimiento con los sensores ubicados en la parte posterior y anterior del robot y siendo teleoperado a tres metros de distancia aproximadamente.

- ✓ La teleoperación háptica es un campo que aún está a la espera de nuevas investigaciones y mejoras de renderizado háptico y la calidad de sensación percibida. Con este proyecto se logró formar bases sólidas, a través de la documentación presentada y los códigos de programación que fueron explicados de la forma más sencilla posible.

- ✓ El consumo de energía logro ser optimizado, logrando que el robot tuviera una autonomía de más de 30 minutos de operación continua. Gracias al trabajo previamente realizado en la elección de materiales y dispositivos con bajo consumo.

Dicho esto se concluye que el mecanismo y módulo de comunicación empleados cumplieron a cabalidad la función y objetivos propuestos, dando lugar también a la realización de futuras mejorar al código de retroalimentación propuesto en este proyecto, considerando además de que las bases para este fin han sido gratificantemente planteadas.

6.1 ANEXO A (Código Arduino)

```
#include <ESP8266WiFi.h>
#include <WiFiUdp.h>
#include <Wire.h>
#include <VL53L0X.h>
#include <AX12A.h>
#include <Servo.h>

#define XSHUT_pin1 14 //5 in arduino 14
#define XSHUT_pin2 12 //6 in arduino 12
#define sensor1_newAddress 42 //el sensor1 no requere cambio de id

//lines needed for dynamixel config
#define DirectionPin   (2u)
#define BaudRate      (1000000u)//1000000u

const char* ssid = "RubenMG";
const char* password = "RubenMG123";

WiFiUDP Udp;
unsigned int localUdpPort = 7777;  // local port to listen on
char incomingPacket[255];  // buffer for incoming packets
//String incomingPacket;  // buffer for incoming packets
char  replyPacket[] = "Hi there! Got the message from ESP:-)";  // a
reply string to send back
int data = 0;
int avatar_position_x = 0;
int avatar_position_y = 0;

int Dist_1;
int Dist_2;
VL53L0X sensor1;
VL53L0X sensor2;
Servo myservo;
int Angulo = 0;
int Angulo1 = 0;

void setup()
{
  pinMode(XSHUT_pin1,OUTPUT);
  pinMode(XSHUT_pin2,OUTPUT);
  Serial.begin(115200);
  Serial.println();
  Wire.begin(); // sensor library
  Wire.setClock(50000);

  Serial.printf("Connecting to %s ", ssid);
  WiFi.begin(ssid, password);
  while (WiFi.status() != WL_CONNECTED)
  {
    delay(500);
    Serial.print(".");
  }
  Serial.println(" connected");

  Udp.begin(localUdpPort);
  Serial.printf("Now listening at IP %s, UDP port %d\n",
WiFi.localIP().toString().c_str(), localUdpPort);

//Cambiar la direccion de los sensores
  pinMode(XSHUT_pin2,INPUT);
  delay(30);
  sensor1.setAddress(sensor1_newAddress);
  pinMode(XSHUT_pin1,INPUT);
  delay(30);
  sensor1.init();
  sensor2.init();
  sensor1.setTimeout(500);
  sensor2.setTimeout(500);
  sensor1.startContinuous();
  sensor2.startContinuous();
```

```
 // code to set dynamixels ------------------------------------
  for (int i = 1; i < 4; i++) {
    ax12a.begin(BaudRate, DirectionPin, &Serial);
    ax12a.setEndless(i, ON);
  }
    myservo.attach(13);
}
void loop()
{
  int packetSize = Udp.parsePacket();
  if (packetSize)
  {

    int len = Udp.read(incomingPacket, 255);
    if (len > 0)
    {
      incomingPacket[len] = 0;
      //Serial.printf("UDP packet contents: %s\n", incomingPacket);

          //code to substrac the haptic avatar position from the
//string sent it from VREP
          char *ptr = strtok (incomingPacket, " ");  // Look for the
//spacebar and take the value before of it
          avatar_position_x = atoi(ptr);         //converts char into
//number
          ptr = strtok (NULL, " ");                  // look for the
//spacebar and take the value after of it
          avatar_position_y = atoi(ptr);
  }
}

    int mov1 = map(avatar_position_x, 0, -145, 0, 800);
//
    if (avatar_position_x>30 and (avatar_position_y>=(-40) &&
avatar_position_y<=40))  { // go forward if Y keeps the moveless space
    ax12a.turn(1, LEFT, -mov1);
    ax12a.turn(2, RIGHT, -mov1);
    ax12a.turn(3, RIGHT, -mov1);
    ax12a.turn(4, LEFT, -mov1);
    }
    if ((avatar_position_x>=(-30) && avatar_position_x<=30) and
(avatar_position_y>=(-40) && avatar_position_y<=40)) {
    ax12a.turn(1, RIGHT, 0);
    ax12a.turn(2, LEFT, 0);
    ax12a.turn(3, LEFT, 0);
    ax12a.turn(4, RIGHT, 0);
    }
    if (avatar_position_x<(-30) and (avatar_position_y>=(-40) &&
avatar_position_y<=40)) { // go backward if Y keeps the moveless space
    ax12a.turn(1, RIGHT, mov1);
    ax12a.turn(2, LEFT, mov1);
    ax12a.turn(3, LEFT, mov1);
    ax12a.turn(4, RIGHT, mov1);
    }

//-----------------------------------------------
```

```
int mov2 = map(avatar_position_y, 0, -195, 0, 800);
    if (avatar_position_y>40 and (avatar_position_x>=(-30) &&
avatar_position_x<=30))  { //go right if x keeps the moveless space
    ax12a.turn(1, LEFT, -mov2); //
    ax12a.turn(2, LEFT, -mov2);
    ax12a.turn(3, RIGHT, -mov2);
    ax12a.turn(4, RIGHT, -mov2);
    }
    if ((avatar_position_y>=(-40) && avatar_position_y<=40) and
((avatar_position_x>=(-30) && avatar_position_x<=30))) {
    ax12a.turn(1, RIGHT, 0);
    ax12a.turn(2, LEFT, 0);
    ax12a.turn(3, LEFT, 0);
    ax12a.turn(4, RIGHT, 0);
    }
    if (avatar_position_y<(-40) and (avatar_position_x>=(-30) &&
avatar_position_x<=30)) { //go left if x keeps the moveless space
    ax12a.turn(1, RIGHT, mov2);
    ax12a.turn(2, RIGHT, mov2);
    ax12a.turn(3, LEFT, mov2);
    ax12a.turn(4, LEFT, mov2);
    }

//--------------------------------------------------------------------
    if((avatar_position_x>40 and avatar_position_x<80) and
(avatar_position_y>125 and avatar_position_y<180)) {// move clockwise
    ax12a.turn(1, RIGHT, mov1);
    ax12a.turn(2, RIGHT, mov1);
    ax12a.turn(3, RIGHT, mov1);
    ax12a.turn(4, RIGHT, mov1);
    }

    if (avatar_position_x>50 and avatar_position_x<90 and
avatar_position_y<(-170)){
    ax12a.turn(1, RIGHT, mov2);
    ax12a.turn(2, RIGHT, mov2);
    ax12a.turn(3, RIGHT, mov2);
    ax12a.turn(4, RIGHT, mov2);
    }
      a();
    //b();
}
 void a(){
  int index1=0, index2=0;
  int dist2[14], dist1[14];
  int ang2[14], ang1[14];

  for (Angulo = 1; Angulo < 170; Angulo += 12)
  {
    myservo.write(Angulo-1);           // tell servo to go to position
in variable 'pos'
    delay(20);                        // waits 15ms for the servo to
reach the position
    int distance1 = sensor1.readRangeContinuousMillimeters();
    if (distance1 <= 400) {
      dist1[index1] = distance1;
      ang1[index1] = Angulo-1;
      index1++;
    }
    else {
      dist1[index1] = 0;
      ang1[index1] = 0;
      index1++;
    }
```

```
int distance2 = sensor2.readRangeContinuousMillimeters();
    if (distance2 <= 400) {
      Angulo1 = Angulo + 180;
      dist2[index2] = distance2;
      ang2[index2] = Angulo1;
      index2++;
    }
    else {
      dist2[index2] = 0;
      ang2[index2] = 0;
      index2++;
    }
 }
    Udp.beginPacket(Udp.remoteIP(),Udp.remotePort());//192.168.1.13
    char data[50]; // vector to save sensor's data (e.g.,
//[123,300])
    sprintf
(data,"%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u",dist1[1],dist1[2],dist1[3],dist1[4],dist1[5],dis
t1[6],dist1[7],dist1[8],dist1[9],dist1[10],dist1[11],dist1[12],dist1[1
3],dist1[14],ang1[1],ang1[2],ang1[3],ang1[4],ang1[5],ang1[6],ang1[7],a
ng1[8],ang1[9],ang1[10],ang1[11],ang1[12],ang1[13],ang1[14]); // sends
values as a char to be converted into number
    Udp.write(data);
    Udp.endPacket();
}

void b(){
int index1=0, index2=0;
int dist2[14], dist1[14];
int ang2[14], ang1[14];

for (Angulo = 170; Angulo > 1; Angulo -= 12) // we're going to cover
14 sensors
  {
    myservo.write(Angulo-1);                    // tell servo to go to
position in variable 'pos'
    delay(20);                                 // waits 15ms for the
servo to reach the position
    int distance1 =sensor1.readRangeContinuousMillimeters();
    if (distance1<=400){
      dist1[index1] = distance1;
      ang1[index1] = Angulo-1;
      index1++;
    }
    else {
      dist1[index1] = 0;
      ang1[index1] = 0;
      index1++;
    }
    int distance2 =sensor2.readRangeContinuousMillimeters();
    if (distance2<=400){
      Angulo1 = Angulo + 180;
      Serial.print(Angulo1);
      dist2[index2] = distance2;
      ang2[index2] = Angulo1;
      index2++;
    }
    else {
      dist2[index2] = 0;
      ang2[index2] = 0;
      index2++;
    }
    Serial.println();
  }
[123,300])
    sprintf
(data,"%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u,%u",dist1[1],dist1[2],dist1[3],dist1[4],dist1[5],dis
t1[6],dist1[7],dist1[8],dist1[9],dist1[10],dist1[11],dist1[12],dist1[1
3],dist1[14],ang1[1],ang1[2],ang1[3],ang1[4],ang1[5],ang1[6],ang1[7],a
ng1[8],ang1[9],ang1[10],ang1[11],ang1[12],ang1[13],ang1[14]); // sends
values as a char to be converted into number
    Udp.write(data);
   Udp.endPacket();
}
```

6.2 ANEXO B (Código VREP)

```
1.
2. function sysCall_init()
3.     usensors={}
4.     sensorDetections={}
5.     laserSensorDetection={}
6.     lSensors={}
7.     avatarPosition={}
8.     avatar_location={}
9.
10.
11.     proportionalController={ 0.5000,         0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000
   , 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000,
12.                              0.5000,         0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000
   , 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000,
13.                              0.5000,         0.5000, 1.0000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000
   , 0.5000, 1.0000, 1.0000, 1.0000, 1.0000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000,
14.                              0.5000,         0.5000, 0.5000, 0.5000, 0.5000, 1.0000, 0.5000, 0.5000, 0.5000
   , 1.0000, 1.0000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000,
15.                              0.5000,         0.5000, 0.5000, 0.5000, 0.5000, 1.0000, 0.5000, 0.5000, 0.5000
   , 1.0000, 0.5000, 0.5000, 0.5000, 1.0000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000,
16.                              0.5000,         0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 1.0000, 0.5000, 0.5000
   , 1.0000, 0.5000, 0.5000, 0.5000, 0.5000, 1.0000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000,
17.                              0.5000,         0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 1.0000, 0.5000
   , 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 1.0000, 0.5000, 0.5000, 0.5000, 0.5000,
18.                              0.5000,         0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000
   , 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000,
```

```
19.                                 0.5000,         0.1000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000
  , 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000,
20.                                 0.5000,         0.1000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000
  , 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000,
21.                                 0.5000,         0.1000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000
  , 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 1.0000, 0.5000, 0.5000, 0.5000,
22.                                 0.5000,         0.1000, 0.1000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000
  , 0.5000, 1.0000, 0.5000, 0.5000, 0.5000, 0.5000, 1.0000, 0.5000, 0.5000, 0.5000, 0.5000,
23.                                 0.5000,         0.5000, 0.1000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000
  , 0.1000, 0.1000, 0.5000, 0.5000, 1.0000, 1.0000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000,
24.                                 0.5000,         0.5000, 0.1000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000
  , 0.1000, 0.5000, 0.5000, 1.0000, 1.0000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000,
25.                                 0.5000,         0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000
  , 0.5000, 0.1000, 0.5000, 0.5000, 1.0000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000,
26.                                 0.5000,         0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000
  , 0.5000, 0.5000, 0.5000, 0.5000, 1.0000, 1.0000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000,
27.                                 0.5000,         0.5000, 0.5000, 0.1000, 0.5000, 0.5000, 0.5000, 0.5000, 0.1000
  , 0.1000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000,
28.                                 0.5000,         0.5000, 0.5000, 0.1000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000
  , 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000,
29.                                 0.5000,         0.5000, 0.5000, 0.5000, 0.5000, 0.1000, 0.5000, 0.5000, 0.5000
  , 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000,
30.                                 0.5000,         0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000
  , 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000, 0.5000}
31.
32.
33.         for i=1,16,1 do
34.             usensors[i]=sim.getObjectHandle("Pioneer_p3dx_ultrasonicSensor"..i)
```

```
35.             sensorDetections[#sensorDetections+1]=99
36.             sensorDetections[#sensorDetections+1]=99
37.             sensorDetections[#sensorDetections+1]=99
38.         end
39.         motorLeft=sim.getObjectHandle("Pioneer_p3dx_leftMotor")
40.         motorRight=sim.getObjectHandle("Pioneer_p3dx_rightMotor")
41.         modelBase=sim.getObjectAssociatedWithScript(sim.handle_self)
42.         pathHandle=sim.getObjectHandle("Path")--CP
43.         sphereHandle=sim.getObjectHandle("Sphere")--CP
44.         targetHandle=sim.getObjectHandle("Target")--CP
45.
46.         -- check if the haptic plugin is loaded
47.         moduleName = 0
48.         moduleVersion = 0
49.         index = 0
50.         pluginNotFound = true
51.         while (moduleName) do
52.             moduleName, moduleVersion = sim.getModuleName(index)
53.             if (moduleName=='CHAI3D') then
54.                 pluginNotFound = false
55.             end
56.             index = index+1
57.         end
58.
59.         if (pluginNotFound) then
60.             sim.displayDialog('Error', 'CHAI3D plugin was not found, or was not correctly initialized
    (v_repExtCHAI3D.dll).', sim.dlgstyle_ok, false, nil, {0.8, 0, 0, 0, 0, 0}, {0.5, 0, 0, 1, 1, 1})
61.         else
```

```
            device = 0
            toolRadius = 0.01
            workspaceRadius = 0.2
            --if (simExtCHAI3D_start(device, toolRadius, workspaceRadius) == -1) then
            if (simCHAI3D.start(device, toolRadius, workspaceRadius) == -1) then
                sim.displayDialog('Error', 'Device failed to
  initialize.', sim.dlgstyle_ok, false, nil, {0.8, 0, 0, 0, 0, 0}, {0.5, 0, 0, 1, 1, 1})
            else
                CHAI3DPluginInitialized = true

                -- add plane to constrain the joystick in 2D
                --(number deviceIndex, table_3 pos, table_3 normal, number Kp, number Kv, number Fmax))
                plane = simCHAI3D.addConstraintPlane(0, {0, 0, 0}, {0, 0, 1}, 0.25, 0.33, 10)

                -- add X/Y guides
                --(number deviceIndex, table_3 posA, table_3 posB, number Kp, number Kv, number Fmax)
                segmentX = simCHAI3D.addConstraintSegment(0,{-0.1,0,0},{0.2,0,0},0.125,0.066,1)
                segmentY = simCHAI3D.addConstraintSegment(0,{0,-0.1,0},{0,0.2,0},0.125,0.066,1)
                -- Add obstacles constraints that will be placed appropriately according to sensor
  readings:
                obstacleConstr={}
                controlTarget=simExtCHAI3D_addConstraintPoint(0,{0,0,0},0.25,0,0)--CP

                for i=1,16,1 do
                    --(number deviceIndex, table_3 pos, number Kp, number Kv, number Fmax)
                    obstacleConstr[i]=simCHAI3D.addConstraintPoint(0,{0,0,0},0.25,0,0)--0
                end
                maxVel=10
```

```
88.             end
89.         end
90.
91.     end
92.
93.
94.     function sysCall_cleanup()
95.      if (CHAI3DPluginInitialized) then
96.             simCHAI3D.reset()
97.         end
98.     end
99.
100.    function sysCall_actuation()
101.        avatarPosition=sim.getObjectPosition(modelBase,-1)--CP
102.        distanceOnPath=sim.getClosestPositionOnPath(pathHandle,avatarPosition)--CP
103.        lengthPath=sim.getPathLength(pathHandle)--CP
104.        --print('lengthPath: ', lengthPath)--CP
105.        positionOnPath=sim.getPositionOnPath(pathHandle,distanceOnPath)--CP
106.        --print('Position Pioneer: ', avatarPosition[1], avatarPosition[2], avatarPosition[3])--CP
107.        spherePosition=sim.getObjectPosition(sphereHandle,-1)--CP
108.        targetPosition=sim.getObjectPosition(targetHandle,-1)--CP
109.        --print('Position Sphere: ', spherePosition[1], spherePosition[2], spherePosition[3])--CP
110.
111.        spherePosition[1]=positionOnPath[1]--CP
112.        spherePosition[2]=positionOnPath[2]--CP
113.        spherePosition[3]=0.25--CP
114.        sim.setObjectPosition(sphereHandle,-1,spherePosition)--CP
115.        --print(sphereHandle,spherePosition) -- RM
```

```
116.        LR=0.3--0.41/2 -- Distancia entre las ruedas al centro del Robot --CP
117.        LRp=LR/lengthPath --Proporcion entre LR y lengthPath --
118.        distanceOnPathT = distanceOnPath+LRp--CP 0.05
119.        if (distanceOnPathT > 1) then
120.            distanceOnPathT = 1
121.        end
122.        positionOnPath=sim.getPositionOnPath(pathHandle,distanceOnPathT)--CP
123.        targetPosition[1]=positionOnPath[1]--CP
124.        targetPosition[2]=positionOnPath[2]--CP
125.        targetPosition[3]=0.25--CP
126.        sim.setObjectPosition(targetHandle,-1,targetPosition)--CP
127.
128.         if (CHAI3DPluginInitialized) then
129.            local rotM=sim.buildMatrix({0,0,0},{0,0,math.pi})
130.            p=simCHAI3D.readPosition(0)
131.            p=sim.multiplyVector(rotM,p)
132.            avatar_location[1] = p[1]*800 -- RM
133.            avatar_location[3] = p[2]*800   -- RM
134.            --print(avatar_location[1],avatar_location[2]) -- haptic devie position x,y -- RM
135.            local forward=p[1]*8   --/ *5
136.            local rot=p[2]*13--/ *13
137.            ---print('forward,rot:',{forward,rot})
138.            vLeft=forward-rot
139.            vRight=forward+rot
140.
141.            sim.setJointTargetVelocity(motorLeft,vLeft)
142.            sim.setJointTargetVelocity(motorRight,vRight)
143.
```

```
144.            if (distanceOnPath > 0.99) then
145.                --FmaxControl=0
146.                vLeft = 0
147.                vRight = 0
148.                sim.setJointTargetVelocity(motorLeft,vLeft)
149.                sim.setJointTargetVelocity(motorRight,vRight)
150.            end
151.
152.        end
153.    end
154.
155.    function sysCall_sensing()
156.    if (CHAI3DPluginInitialized) then
157.                    --sim.buildMatrix(position,eulerAngles,matrix)
158.     --[[
159.    The x-axis of the orientation component of the matrix is (matrix[0],matrix[4],matrix[8])
160.    The y-axis of the orientation component of the matrix is (matrix[1],matrix[5],matrix[9])
161.    The z-axis of the orientation component of the matrix is (matrix[2],matrix[6],matrix[10])
162.    The position component of the matrix is (matrix[3],matrix[7],matrix[11])
163.     --]]
164.            local rot=sim.buildMatrix({0,0,0},{0,0,math.pi})
165.            for i=1,16,1 do
166.                -- res {deteccion 1, no deteccti?n 0, error -1}
167.                -- dist distacia al punto detectado
168.                -- pt coordenada relativa al sensor del punto detectado
169.                local res,dist,pt=sim.readProximitySensor(usensors[i])
170.                if res>0 then
```

```
171.                    local m=sim.getObjectMatrix(usensors[i],modelBase)-- Retrieves transformation matrix
  of an object
172.                    local p=sim.multiplyVector(m,pt)
173.                    sensorDetections[3*(i-1)+1]=p[1]
174.                    sensorDetections[3*(i-1)+2]=p[2]
175.                    sensorDetections[3*(i-1)+3]=p[3]
176.                else
177.                    sensorDetections[3*(i-1)+1]=99
178.                    sensorDetections[3*(i-1)+2]=99
179.                    sensorDetections[3*(i-1)+3]=99
180.                end
181.            end
182.            -- code for udp socket communication
183.            local socket = require("socket")
184.            udp = socket.udp()
185.            udp:setpeername('192.168.43.21', 7777)-- 127.0.0.1 192.168.1.17
186.            udp:settimeout(1) -- it had cero
187.            avatar_location[1] = tonumber(string.format("%.0f", avatar_location[1]))
188.            avatar_location[2] = ' ' -- needed to separated the two values in ESP
189.            avatar_location[3] = tonumber(string.format("%.0f", avatar_location[3]))
190.            local msg = table.concat(avatar_location) -- to put both sensor data together
191.            udp:send(msg)
192.            udp:settimeout(10)
193.            data = udp:receive()
194.            if data then
195.                --print("Received: ", data)
196.            else
197.                print("Nothing received yet :|")
```

```
198.            end
199.            --udp:close() -- it seems it's not needed
200.            -- end of udp socket communication
201.            --
202.            -- code to convert a string into number
203.            --local str = "11, 22" - data in this case
204.            local regxEverythingExceptComma = '([^,]+)' --"%S+"
205.            local a={}
206.            for x in string.gmatch(data, regxEverythingExceptComma) do --(str, regxEverythingExceptComma)
207.            r = tonumber(x)
208.            table.insert(a, r)
209.            end
210.            print(a)-- it shows the number 22
211.
212.            lSensors[1]=a[1]/1000
213.            lSensors[2]=a[2]/1000
214.            lSensors[3]=a[3]/1000
215.            lSensors[4]=a[4]/1000
216.            lSensors[5]=a[5]/1000
217.            lSensors[6]=a[6]/1000
218.            lSensors[7]=a[7]/1000
219.            lSensors[8]=a[8]/1000
220.            lSensors[9]=a[9]/1000
221.            lSensors[10]=a[10]/1000
222.            lSensors[11]=a[11]/1000
223.            lSensors[12]=a[12]/1000
224.            lSensors[13]=a[13]/1000
225.            lSensors[14]=a[14]/1000
```

```
226.
227.            laserSensorDetection[1]=-lSensors[1]*math.cos(a[15]*3.1416/180)
228.            laserSensorDetection[2]=-lSensors[1]*math.sin(a[15]*3.1416/180)
229.            laserSensorDetection[3]=0
230.            laserSensorDetection[4]=-lSensors[2]*math.cos(a[16]*3.1416/180)
231.            laserSensorDetection[5]=-lSensors[2]*math.sin(a[16]*3.1416/180)
232.            laserSensorDetection[6]=0
233.            laserSensorDetection[7]=-lSensors[3]*math.cos(a[17]*3.1416/180)
234.            laserSensorDetection[8]=-lSensors[3]*math.sin(a[17]*3.1416/180)
235.            laserSensorDetection[9]=0
236.            laserSensorDetection[10]=-lSensors[4]*math.cos(a[18]*3.1416/180)
237.            laserSensorDetection[11]=-lSensors[4]*math.sin(a[18]*3.1416/180)
238.            laserSensorDetection[12]=0
239.            laserSensorDetection[13]=-lSensors[5]*math.cos(a[19]*3.1416/180)
240.            laserSensorDetection[14]=-lSensors[5]*math.sin(a[19]*3.1416/180)
241.            laserSensorDetection[15]=0
242.            laserSensorDetection[16]=-lSensors[6]*math.cos(a[20]*3.1416/180)
243.            laserSensorDetection[17]=-lSensors[6]*math.sin(a[20]*3.1416/180)
244.            laserSensorDetection[18]=0
245.            laserSensorDetection[19]=-lSensors[7]*math.cos(a[21]*3.1416/180)
246.            laserSensorDetection[20]=-lSensors[7]*math.sin(a[21]*3.1416/180)
247.            laserSensorDetection[21]=0
248.            laserSensorDetection[22]=-lSensors[8]*math.cos(a[22]*3.1416/180)
249.            laserSensorDetection[23]=-lSensors[8]*math.sin(a[22]*3.1416/180)
250.            laserSensorDetection[24]=0
251.            laserSensorDetection[25]=-lSensors[9]*math.cos(a[23]*3.1416/180)
252.            laserSensorDetection[26]=-lSensors[9]*math.sin(a[23]*3.1416/180)
253.            laserSensorDetection[27]=0
```

```
254.            laserSensorDetection[28]=-lSensors[10]*math.cos(a[24]*3.1416/180)
255.            laserSensorDetection[29]=-lSensors[10]*math.sin(a[24]*3.1416/180)
256.            laserSensorDetection[30]=0
257.            laserSensorDetection[31]=-lSensors[11]*math.cos(a[25]*3.1416/180)
258.            laserSensorDetection[32]=-lSensors[11]*math.sin(a[25]*3.1416/180)
259.            laserSensorDetection[33]=0
260.            laserSensorDetection[34]=-lSensors[12]*math.cos(a[26]*3.1416/180)
261.            laserSensorDetection[35]=-lSensors[12]*math.sin(a[26]*3.1416/180)
262.            laserSensorDetection[36]=0
263.            laserSensorDetection[37]=-lSensors[13]*math.cos(a[27]*3.1416/180)
264.            laserSensorDetection[38]=-lSensors[13]*math.sin(a[27]*3.1416/180)
265.            laserSensorDetection[39]=0
266.            laserSensorDetection[40]=-lSensors[14]*math.cos(a[28]*3.1416/180)
267.            laserSensorDetection[41]=-lSensors[14]*math.sin(a[28]*3.1416/180)
268.            laserSensorDetection[42]=0
269.
270.
271.            for i=1,14,1 do -- for i=1,16,1 do RM
272.
273.             --   local x=sensorDetections[3*(i-1)+1]
274.             --   local y=sensorDetections[3*(i-1)+2]
275.                local x=laserSensorDetection[3*(i-1)+1]
276.                local y=laserSensorDetection[3*(i-1)+2]
277.                local l=math.sqrt((x*x)+(y*y))
278.                x=x/l
279.                --if i>8 then  -- sensores parte trasera
280.                --    y=-y/l
281.                --else
```

```
282.                --    y=y/l
283.                --end
284.                local f=0
285.                local maxF=12
286.                local minFR=0.5 --metros!
287.                local maxFR=0.2
288.
289.             avatarPosition[1]= 0--RM
290.             avatarPosition[2]= 0--RM
291.             avatarPosition[3]=0.25--RM
292.             sim.setObjectPosition(modelBase,-1,avatarPosition)--RM sphereHandle
293.
294.             avatarPosition=sim.getObjectPosition(modelBase,-1)  -- muestra la posicion del pioneer
295.             --print('position_pioneer:',avatarPosition)         -- muestra la posicion del pioneer
296.             Xp = avatarPosition[1]
297.             Yp = avatarPosition[2]
298.             Cx = math.floor(Xp/0.25)+11
299.             Cy = 10-math.floor(Yp/0.25)
300.             k = proportionalController[Cx+(Cy-1)*20]--Cx+(Cy-1)*20
301.             k = 1 -- 1: con asistencia  0: sin asistencia
302.             --print('Celda y k:',Cx,Cy,  k)
303.               if l<minFR then
304.                    local t=maxF/(maxFR-minFR) --ley control fuerza
305.                    local s=-minFR*t
306.                    f=math.max(0,l*t+s)
307.                end
308.                --simExtCHAI3D_updateConstraint(number objectID, table_3 pos, table_3 dir, number Kp,
  number Kv, number Fmax)
```

```
309.          --simCHAI3D.updateConstraint(obstacleConstr[i],{x,y,0},{0,0,0},0.25*k,0,f*k)--f --Generacion
  del vector e setido opuesto desde la fuerza detectada
310.          simCHAI3D.updateConstraint(obstacleConstr[i],{x,y,0},{0,0,0},0.25,0,f)--f --Generacion del
  vector e setido opuesto desde la fuerza detectada
311.
312.           end
313.           directionForce=sim.getObjectPosition(targetHandle,modelBase)
314.           directionForce[1]=directionForce[1]*8/13;--5/13--0.7 0.3 ok--CP
315.           --print('directionForce: ', directionForce[1], directionForce[2], directionForce[3])--CP
316.           --simExtCHAI3D_updateConstraint(controlTarget,{-1*directionForce[1], -1*directionForce[2], -
  1*directionForce[3]},{0,0,0},0.25,0,5)--CP
317.
318.         k = 0 -- 1: con asistencia  0: sin asistencia --rm
319.         kpControl=0.4*k --0.75
320.         kvControl=0.2*k-- 0.7 0.1
321.         FmaxControl=5*k
322.
323.     if (distanceOnPath > 0.99) then
324.          FmaxControl=0
325.       end
326.
327.       simExtCHAI3D_updateConstraint(controlTarget,{-1*directionForce[1], -1*directionForce[2], -
  1*directionForce[3]},{0,0,0},kpControl,kvControl,FmaxControl)--CP
328.
329.       end
330.
331.   end
```

6.3 REFERENCIAS BIBLIOGRAFÍCAS

3WD Triangular 100mm omni wheel mobile robotics car. (2018). Retrieved August 7, 2018, from http://www.microrobo.com/3wd-triangular-100mm-omni-wheel-mobile-robotics-car-c003.html

a X-12. (2006). *Communication*.

Ackerman, E. (2013). *Adept Introduces Lynx Autonomous Mobile Platform - IEEE Spectrum*. Retrieved from https://spectrum.ieee.org/automaton/robotics/industrial-robots/adept-introduces-lynx-autonomous-mobile-platform

Adept. (2011). Pioneer 3-DX. *Adept Mobile Robots*, 2.

AJNA, R., & HERSAN, T. (2019). The Dynamixel AX-12A Servos. Retrieved August 19, 2019, from https://memememememememe.me/post/the-dynamixel-ax-12a-servos/

Andreu, V., Torronteras, A., & Mora, F. (2015). *Trabajo final de carrera*.

ArbotiX-M Robocontroller. (2014). Retrieved May 28, 2019, from https://www.trossenrobotics.com/p/arbotix-robot-controller.aspx

AX-12A. (2017). Retrieved August 19, 2019, from http://emanual.robotis.com/docs/en/dxl/ax/ax-12a/

Bogado Torres, J. M. (2007). *Control bilateral de Robots*. 204.

Brooks, D. J., Tsui, K. M., Lunderville, M., & Yanco, H. (2015). Methods for Evaluating and Comparing the Use of Haptic Feedback in Human-Robot Interaction with Ground-Based Mobile Robots. *Journal of Human-Robot Interaction*, *4*(1), 3. https://doi.org/10.5898/JHRI.4.1.Brooks

Category:Physical layer protocols - Wikipedia. (2018). Retrieved January 1, 2020, from https://en.wikipedia.org/wiki/Category:Physical_layer_protocols

CM-5 - ROBOTIS. (2015). Retrieved December 31, 2019, from http://www.robotis.us/cm-5/

Communication Protocols in Embedded Systems - Types, Advantages & Disadvantages. (2018). Retrieved January 1, 2020, from https://electricalfundablog.com/communication-protocols-embedded-systems/

Compuerta lógica de tres estados - Ingeniería Mecafenix. (n.d.). Retrieved January 8, 2020, from https://www.ingmecafenix.com/electronica/compuerta-tres-estados/

CoppeliaSim. (2017). Child scripts. Retrieved June 11, 2020, from https://www.coppeliarobotics.com/helpFiles/en/childScripts.htm

driving-robotic-dynamixel-servos @ www.arduinotutorialonline.com. (2019). Retrieved from http://www.arduinotutorialonline.com/2018/01/driving-robotic-dynamixel-servos.html

Driving Robotis Dynamixel Servos with Arduino - Robosoup. (2016). Retrieved June 3, 2019, from https://www.robosoup.com/2014/03/driving-robotis-dynamixel-servos-with-arduino.html

Dudek, G., & Jenkin, M. (2016). Robotics Handbook. In *Springer Handbook of Robotics*. https://doi.org/978-3-319-32552-1

DYNAMIXEL Shield | SANDOROBOTICS. (2018). Retrieved December 31, 2019, from https://sandorobotics.com/producto/902-0146-000/

Electronic Communication Protocols Basics and Types with Functionality. (2017). Retrieved January 3, 2020, from https://www.elprocus.com/communication-protocols/

Fong, T., & Thorpe, C. (2001). Vehicle teleoperation interfaces. *Autonomous Robots*, *11*(1), 9–18. https://doi.org/10.1023/A:1011295826834

Giuliano, G. (2009). Diseño Madera. Retrieved January 14, 2020, from https://es.slideshare.net/cjvial/diseo-madera

Greicius, T. (2015). *Mars Science Laboratory - Curiosity*. Retrieved from https://www.nasa.gov/mission_pages/msl/index.html

Gross, H.-M., Meyer, S., Scheidig, A., Eisenbach, M., Mueller, S., Trinh, T. Q., … Fricke, C. (2017). Mobile robot companion for walking training of stroke patients in clinical post-stroke rehabilitation. *2017 IEEE International Conference on Robotics and Automation (ICRA)*, 1028–1035. https://doi.org/10.1109/ICRA.2017.7989124

Gunawan, A. A. S., William, Hartanto, B., Mili, A., Budiharto, W., Salman, A. G., & Chandra, N. (2017). Development of Affordable and Powerful Swarm Mobile Robot Based on Smartphone Android and IOIO board. *Procedia Computer Science*, *116*, 342–350. https://doi.org/10.1016/j.procs.2017.10.057

Guo, W., Jiang, S., Zong, C., & Gao, X. (2014). Development of a transformable wheel-track mobile robot and obstacle-crossing mode selection. *2014 IEEE International Conference on Mechatronics and Automation, IEEE ICMA 2014*, 1703–1708. https://doi.org/10.1109/ICMA.2014.6885957

Handson Technology. (2017). User Manual V1.2: ESP8266 NodeMCU WiFi Devkit. *Handson Technology*, 1–22. Retrieved from http://www.handsontec.com/pdf_learn/esp8266-V10.pdf

Harvest Automation. (2008). Harvest Automation | Mobile Autonomous Robots for Agriculture. Retrieved August 10, 2018, from https://www.public.harvestai.com/

I2C | Aprendiendo Arduino. (2019). Retrieved January 7, 2020, from https://aprendiendoarduino.wordpress.com/2017/07/09/i2c/

iRobot. (2017). iRobot 510 PackBot Multi-Mission Robot - Army Technology. Retrieved August 10, 2018, from https://www.army-technology.com/projects/irobot-510-packbot-multi-mission-robot/

Isogawa, Y. (2017). LEGO Mindstorm mecanum wheel vehicles by Yoshihito Isogawa | The Kid Should See This. Retrieved August 13, 2018, from http://thekidshouldseethis.com/post/lego-mindstorm-mecanum-wheel-vehicles-by-yoshihito-isogawa

Kaliński, K. J., & Mazur, M. (2016). Optimal control of 2-wheeled mobile robot at energy performance index. *Mechanical Systems and Signal Processing*, *70–71*, 373–386. https://doi.org/10.1016/j.ymssp.2015.09.047

Kit Chasis con Motores 2WD miniQ - VISTRONICA SAS. (2018). Retrieved August 7, 2018, from https://www.vistronica.com/robotica/robot/kit-chasis-con-motores-2wd-miniq-detail.html

Kit Robot MiniQ 2WD v2.0 - VISTRONICA SAS. (2018). Retrieved August 7, 2018, from https://www.vistronica.com/robotica/robot/kit-robot-miniq-2wd-v2-0-detail.html

Klančar, G., Zdešar, A., Blažič, S., & Škrjanc, I. (2017). *Wheeled Mobile Robotics*.

Konduri, S., Cobos Torres, E. O., & Pagilla, P. R. (2014). Effect of wheel slip in the coordination of wheeled mobile robots. In *IFAC Proceedings Volumes (IFAC-PapersOnline)* (Vol. 19). https://doi.org/10.3182/20140824-6-ZA-1003.02716

Kuchenbecker, K. J., Fiene, J., & Niemeyer, G. (2006). Improving contact realism through event-based haptic feedback. *IEEE Transactions on Visualization and Computer Graphics*, *12*(2), 219–229. https://doi.org/10.1109/TVCG.2006.32

Kuka Robotics. (2014). Weblet Importer. Retrieved August 7, 2018, from 2014-01-06 website: https://www.eu-robotics.net/sparc/success-stories/enabling-researchers-to-innovate-in-small-scale-for-the-factory-of-the-future.html?changelang=2

Le, K. D., Nguyen, H. D., Ranmuthugala, D., & Forrest, A. (2016). Artificial potential field for remotely operated vehicle haptic control in dynamic environments. *Proceedings of the Institution of Mechanical Engineers. Part I: Journal of Systems and Control Engineering*, *230*(9), 962–977. https://doi.org/10.1177/0959651816660484

Li, H., & Savkin, A. V. (2018). An algorithm for safe navigation of mobile robots by a sensor network in dynamic cluttered industrial environments. *Robotics and Computer-Integrated Manufacturing*, *54*(March 2017), 65–82. https://doi.org/10.1016/j.rcim.2018.05.008

Librerías — documentación de ESP8266 Arduino Core - 2.4.0. (2019). Retrieved January 7, 2020, from https://esp8266-arduino-spanish.readthedocs.io/es/latest/libraries.html

Luo, Z., Shang, J., Wei, G., & Ren, L. (2018). A reconfigurable hybrid wheel-track mobile robot based on Watt II six-bar linkage. *Mechanism and Machine Theory*, *128*, 16–32. https://doi.org/10.1016/j.mechmachtheory.2018.04.020

MakeBlock Robotics. (n.d.). How to Make an All-direction Vehicle With Mecanum Wheels: 8 Steps (with Pictures). Retrieved August 7, 2018, from 2015-29-09 website: https://www.instructables.com/id/All-direction-Vehicle-with-Mecanum-Wheels/

Malu, S. K., & Majumdar, J. (2014). Kinematics, Localization and Control of Differential Drive Mobile Robot. *Global Journal of Researches in Engineering*, *14*(1), 1–8.

Marcano Gamero, C. R. (2008). *Interfaces para Aplicaciones de Telerrobóticas y de Teleoperación*. 118.

Martinez, E., & Calil, C. (2002). RESISTENCIA MECANICA DE LOS TABLEROS DE DENSIDAD MEDIA: PARTE 1: RESISTENCIA A LA TRACCION PARALELA A LA SUPERFICIE. Retrieved January 10, 2020, from https://scielo.conicyt.cl/scielo.php?script=sci_arttext&pid=S0718-221X2002000200008

Mecanum Wheel vehicle | 3D CAD Model Library | GrabCAD. (2017). Retrieved July 3, 2019, from https://grabcad.com/library/mecanum-wheel-vehicle-1

Medium Density Fiberboard (MDF) :: MakeItFrom.com. (2018). Retrieved January 15, 2020, from https://www.makeitfrom.com/material-properties/Medium-Density-Fiberboard-MDF

Microautomacion. (2018). *Automatización Y Control*. https://doi.org/10.0.0.0

Microelectronics, S. (2018). *VL53L0X World ' s smallest Time-of-Flight ranging and gesture detection*. (April), 40.

MORCILLO MARTÍNEZ, L. (2018). *Sistema de detección de obstáculos para drones basado en sensor láser*. Retrieved from https://riunet.upv.es:443/handle/10251/105633

Nexus, R. (2016). (4 inch) 100mm Mecanum Wheel Left /Bearing Rollers14094L | NEXUS Robot. Retrieved August 29, 2018, from http://www.nexusrobot.com/product/4-inch-100mm-mecanum-wheel-left-bearing-rollers14094l.html

Nguyen, B.-H., & Ryu, J.-H. (2010). Design of a master device for the teleoperation of wheeled and tracked vehicles. *Control Automation and Systems (ICCAS), 2010 International Conference On*, 1643–1648.

Nicolau, R. (2018). *Omnidirectional scanner using a time of flight sensor by*. (February).

Nu, E. (2014). *Teleoperación [de robots]: técnicas , aplicaciones , entorno sensorial y teleoperación inteligente Teleoperación : técnicas , aplicaciones , entorno sensorial y teleoperación inteligente Emmanuel Nuño Ortega , Luis Basañez Villaluenga IOC-DT-P-2004-05 A*. (May).

Olimpiu, M., Mândru, D., Ardelean, I., & Ple, A. (2014). Design and Development of an Aut onomous Directional Mobile Rob bot with h Mecanum Wheels. *Design and Development of an Aut Onomous Omni-Directional Mobile Ro Bot with Mecanum Wheels*, 1–6.

Ollero, A. (2001). Modelos cinematicos de robots. In MARCOMBO (Ed.), *Robotica Manipuladores y robots moviles.* Barcelona (España).

Ortigoza, R. (2007). Robots Móviles: Evolución y Estado del Arte. *Polibits.Gelbukh.Com*, 12–17. Retrieved from http://polibits.gelbukh.com/2007_35/Robots Moviles_ Evolucion y Estado del Arte.pdf

Pastor, J. (2019). Windows 10 ya funciona en la Raspberry Pi 3: dos proyectos independientes lo hacen posible. Retrieved September 7, 2019, from https://www.xataka.com/ordenadores/windows-10-funciona-raspberry-pi-3-dos-proyectos-independientes-hacen-posible

Qiu, Q., Fan, Z., Meng, Z., Zhang, Q., Cong, Y., Li, B., … Zhao, C. (2018). Extended Ackerman Steering Principle for the coordinated movement control of a four wheel drive agricultural mobile robot. *Computers and Electronics in Agriculture*, *152*(December 2017), 40–50. https://doi.org/10.1016/j.compag.2018.06.036

Robot móvil SUMMIT XL HL | Robotnik. (2017). Retrieved August 7, 2018, from https://www.robotnik.es/robots-moviles/summit-xl-hl/

RPLIDAR-A1 360°Laser Range Scanner _ Domestic Laser Range Scanner|SLAMTEC. (2018). Retrieved July 11, 2019, from http://www.slamtec.com/en/lidar/a1

Savage, J. (2019). Arduino y Biblioteca Dynamixel AX-12A. Retrieved December 31, 2019, from 2019 website: https://savageelectronics.com/blog/arduino-biblioteca-dynamixel

Sempere, A. D., Serna-Leon, A., Gil, P., Puente, S., & Torres, F. (2015). Control and guidance of low-cost robots via gesture perception for monitoring activities in the home. *Sensors (Switzerland)*, *15*(12), 31268–31292. https://doi.org/10.3390/s151229853

Shield - Dynamixel AX. (2017). Retrieved December 31, 2019, from http://tdrobotica.co/shield-dynamixel-ax/534.html

SMARS modular robot by tristomietitoredeituit - Thingiverse. (2017). Retrieved August 7, 2018, from https://www.thingiverse.com/thing:2662828

SMP Robotics. (2009). Robot Guard | SMP Robotics - Autonomous mobile security systems - S5 Bot. Retrieved August 10, 2018, from https://smprobotics.com/application_autonomus_mobile_robots/robot-guard/

Sparkfun. (2018). Getting Started with the Raspberry Pi Zero Wireless - learn.sparkfun.com. Retrieved August 29, 2019, from https://learn.sparkfun.com/tutorials/getting-started-with-the-raspberry-pi-zero-wireless/all

STMicroelectronics. (2016). *UM2039 User Manual World smallest Time-of-Flight ranging and gesture detection sensor Application Programming Interface*. (June), 26. Retrieved from www.st.com

Tarjeta interfaz Dynamixel. (2019). Retrieved December 31, 2019, from https://www.savageelectronics.com/blog/tarjeta-interfaz-dynamixel

Transcend Robotics. (2015). GROUND DRONE PROJECT: A VERSATILE MOBILE ROBOTIC PLATFORM by Transcend Robotics — Kickstarter. Retrieved August 10, 2018, from https://www.kickstarter.com/projects/1145776805/ground-drone-project-a-versatile-mobile-robotic-pl?ref=category

Trubin, J. (2013). Telepresence & Teleoperation & Telerobotics: Experiments, Studies and Background Information. Retrieved August 14, 2018, from https://www.juliantrubin.com/encyclopedia/electronics/telepresence.html

Wang, T., Wu, Y., Liang, J., Han, C., Chen, J., & Zhao, Q. (2015). Analysis and experimental kinematics of a skid-steering wheeled robot based on a laser scanner sensor. *Sensors (Switzerland)*, *15*(5), 9681–9702. https://doi.org/10.3390/s150509681

Wheels for a course stable selfpropelling vehicle movable in any desired direction on the ground or some other base. (1972). Retrieved from https://patents.google.com/patent/US3876255