# TECNOLÓGICO DE COSTA RICA

## CARRERA DE INGENIERÍA MECATRÓNICA


# “Robótica inteligente: Implementación de sensores 3D para desenvolvimiento de robots móviles y vehículos autónomos”

**Laboratório de Robótica Móvel- ICMC- Universidade de São Paulo**

**Informe de Proyecto de Graduación para optar por el título de Ingeniero en Mecatrónica con el grado académico de Licenciatura**

**Estudiante:**

**Sergio Valverde Moreno**

**201121914**

**Diciembre de 2015**

**INSTITUTO TECNOLÓGICO DE COSTA RICA**

**CARRERA DE INGENIERÍA MECATRÓNICA**

**PROYECTO DE GRADUACIÓN**

**ACTA DE APROBACIÓN**

royecto de Graduación defendido ante el presente Tribunal Evaluador como requisito ara optar por el título de Ingeniero en Mecatrónica con el grado académico de icenciatura, del Instituto Tecnológico de Costa Rica.

Estudiante: Sergio Valverde Moreno

**Nombre del proyecto: Robótica Inteligente: Implementación de sensores 3D para desenvolvimiento de robots móviles y vehículos autónomos.**

Miembros del Tribunal

Dr. Mauricio Muñoz

Profesor lector

Dr. Juan Luis Crespo Mariño

Profesor asesor

Los miembros de este Tribunal dan fe de que el presente trabajo de graduación ha sido aprobado y cumple con las normas establecidas por la Carrera de Ingeniería Mecatrónica

Cartago, Diciembre 2015

# Declaración de autenticidad

Declaro que el presente Proyecto de Graduación ha sido realizado enteramente por mi persona, utilizando y aplicando literatura referente al tema e introduciendo conocimientos propios.

En los casos en que he utilizado bibliografía, he procedido a indicar las fuentes mediante las respectivas citas bibliográficas.

En consecuencia, asumo la responsabilidad total por el trabajo de graduación realizado y por el contenido del correspondiente informe final.

São Carlos, São Paulo, Brasil,
Diciembre 2015

Sergio Valverde Moreno

Céd: 115370017

## HOJA DE INFORMACIÓN DEL PROYECTO

**Datos del estudiante:**

**Nombre:** Sergio Valverde Moreno

**Cédula:** 1-1537-0017 **Carné ITCR:** 201121914

**Dirección de su residencia:** De la última parada de buses de Platanares de Moravia, 800 metros este, 25 metros norte. Casa amarilla de dos pisos, portón rojo.

**Teléfono de residencia:** 2529-0259

**Teléfono celular:** 8895-0007

**Correo electrónico:** sergiovalverdem@hotmail.com

**Información del proyecto:**

**Nombre del Proyecto:** Robótica Inteligente: Implementación de sensores 3D para desenvolvimiento de robots móviles y vehículos autónomos.

**Información de la empresa:**

**Nombre:** Laboratório de Robótica Móvel- ICMC- Universidade de São Paulo

**Actividad Principal:** Centro de Investigación

**Zona:** São Carlos, São Paulo, Brasil

**Dirección:** Avenida Trabalhador São-carlense, 400 - Centro

**Teléfono:** +55 (16) 3373-9700

**Información del encargado/asesor en la empresa:**

**Nombre:** Dr. Fernando Santos Osório

**Puesto que ocupa:** Profesor investigador

**Departamento:** Laboratório de Robótica Móvel

**Profesión:** Profesor **Grado académico:** Doctor

**Correo electrónico:** fosorio@icmc.usp.br

# Resumen

El presente proyecto se realizó en la *Universidade de São Paulo* en Brasil, en la sede de la ciudad de *São Carlos*. Específicamente se trabajó en el *Laboratório de Robótica Móvel*, perteneciente al *Instituto de Ciências Matemáticas e de Computação*. Como su nombre lo dice, en este laboratorio se realiza investigación y desarrollo de aplicaciones relacionadas al área de robótica móvil.

El proyecto consistió en desarrollar una aplicación para que un robot móvil pueda navegar de forma autónoma, evitando obstáculos, hasta llegar a una coordenada específica definida por el usuario. Para diseñar e implementar esta aplicación se debe tomar en cuenta que la navegación y detección de obstáculos se hizo con base en sensores 3D, para lo cual se seleccionó un Kinect.

La aplicación se simuló y probó en un ambiente de simulación especial para aplicaciones robóticas, llamado V-REP. El proceso de desarrollo del proyecto se inició con la elaboración de algoritmos de navegación para únicamente evitar obstáculos. Una vez alcanzado este objetivo, se procedió a implementar la teoría de campos potenciales y se utilizaron otros sensores para determinar la posición y orientación del robot en el espacio, con el objetivo de definir una trayectoria hasta la coordenada meta y así hacer la navegación más eficiente.

Por último, se trabajó con los datos reales de los sensores en físico para corroborar el funcionamiento de la aplicación y así compararlo con los resultados obtenidos anteriormente de la parte simulada.

**Palabras clave:** Navegación autónoma; Robótica móvil; Pioneer P3-DX; Kinect; Campos potenciales; Waypoints.

# Abstract

This project was developed in the *Universidade de São Paulo* in Brasil, on the campus located in the city of *São Carlos*. The work was done specifically in the *Laboratório de Robótica Móvel*, which belongs to the *Instituto de Ciências Matemáticas e Computação*. As its name indicates, this laboratory does research and develops applications related to the area of mobile robotics.

The project consisted on developing an application that allows a mobile robot to navigate autonomously, avoiding obstacles, until it reaches a specific coordinate point defined by the user. For the design and implementation of this application, it is necessary to keep in mind that navigation and obstacle detection are based on the use of 3D sensors. For this purpose, the Kinect was selected.

The simulation and testing of the project was done in a special simulation environment for robotic applications, called V-REP. The project development process began with the elaboration of navigation algorithms for obstacle avoidance. Once this goal was achieved, the potential fields theory was implemented and other sensors were used to determine the robot's spatial position and orientation, with the objective of defining a trajectory to the goal coordinate point, thus doing a more efficient navigation.

Finally, real data from the physical sensors was used to corroborate the functioning of the application, with the purpose of comparing it with the previously obtained results from the simulated part.



**Keywords:** Autonomous navigation; Mobile robotics; Pioneer P3-DX; Kinect; Potential fields; Waypoints.

# Dedicatoria

*Dedico este trabajo a mi familia y Diana, porque gracias a su incondicional apoyo y guía me han ayudado a superar los desafíos que se han presentado a lo largo de mi vida y por ustedes es que estoy cada día más cerca de cumplir mis metas y de convertirme en profesional.*

# Agradecimientos

Quiero agradecer profundamente al doctor Fernando Santos Osório y al doctor Juan Luis Crespo Mariño, los cuales con sus conocimientos me asesoraron de la mejor manera para llevar a cabo este proyecto exitosamente.

# ÍNDICE GENERAL

# ÍNDICE DE FIGURAS

# ÍNDICE DE TABLAS

# Capítulo 1: Introducción

## 1.1. Contextualización y motivación

El ser humano siempre ha buscado métodos para facilitar su trabajo y siempre que sea posible hacerlo más eficiente. Por ello se ha visto que a lo largo de la historia, se ha valido de la tecnología para llevar a cabo esta tarea. Es aquí donde la robótica ha tomado un papel fundamental para cumplir este objetivo.

Los ejemplos más claros de esto se pueden ver en el sector industrial, con los brazos robóticos. Su velocidad de trabajo y precisión han ayudado a mejorar la producción y el desarrollo de las empresas. Sin embargo, este tipo de robots en su mayoría se encuentran restringidos a una única posición y campo de trabajo por el hecho de tener una base fija.

Es aquí donde el área de la robótica móvil toma importancia. La robótica móvil es el área de investigación que se encarga del control de vehículos autónomos y semiautónomos. Lo que diferencia esta área de investigación con respecto a otras áreas tradicionales como manipuladores robóticos, inteligencia artificial, y visión computacional, es el énfasis en problemas relacionados a entender regiones del espacio substancialmente más grandes que aquellas que pueden ser observadas a partir de un solo punto de vista. Por esta razón, tareas como moverse en el espacio de forma adecuada, realizar mediciones del ambiente por el que se desplaza el robot, así como el procesamiento y razonamiento acerca de la información del ambiente, son tareas básicas en la robótica móvil. (Dudek & Michael, 2010)

Esta área ha tomado gran relevancia en los últimos años y se puede evidenciar que en los próximos tendrá aún más impacto. Por esta razón, se decidió realizar el proyecto de graduación en esta área. El proyecto se llevó a cabo en la *Universidade de São Paulo* (USP) en la sede São Carlos, específicamente en el *Laboratório de Robótica Móvel* perteneciente al *Instituto de Ciências Matemáticas e de Computação*.

De forma general la *Universidade de São Paulo* ha destacado por ser uno de los principales centros de educación superior en el mundo, lo cual se ha reflejado en distintos rankings de universidades. No solo es la universidad número uno de Brasil sino que de

Latinoamérica también lo es y está entre las cien primeras a nivel mundial, según los reconocidos rankings de QS World University Rankings, World Reputation Ranking, Webometrics Ranking of Universities, entre otros.

La universidad fue creada en el año 1934 y desde entonces se ha caracterizado por su calidad y excelencia. Actualmente, la USP es responsable por el 22% de la producción científica del país. Para desarrollar sus actividades, la universidad cuenta con ocho campus en distintas ciudades del estado de São Paulo, además de museos y centros de investigación ubicados en distintos puntos del país. (Universidade de São Paulo, 2015)

De forma más específica, el *Laboratório de Robótica Móvel* desarrolla investigación en diversas áreas relacionadas a robótica y sistemas de transporte inteligente. Dentro de ellas se destacan: visión computacional, aprendizaje de máquina, robots y vehículos autónomos. Actualmente, la mayor parte de la investigación realizada en el laboratorio está relacionada al desarrollo de vehículos robóticos inteligentes para ambientes urbanos y agrícolas.

A través de colaboraciones con empresas e institutos de investigación de Brasil y del exterior, el laboratorio desarrolla sistemas robustos, eficientes y capaces de operar en ambientes reales, actuando en conjunto o sustituyendo humanos en tareas que representen un riesgo. (Laboratório de Robótica Móvel, 2015)

El proyecto que se llevó a cabo en este laboratorio, consiste en desarrollar e implementar algoritmos de navegación para que un robot móvil pueda navegar de forma autónoma y llegar hasta una coordenada o punto específico. Para la simulación de estos algoritmos se usará un modelo de un robot Pioneer P3-DX de la empresa Adept.

A pesar de que este robot cuenta con sensores ultrasónicos alrededor de él para ayudar en la navegación y evitar colisiones, no se hará uso de ellos. Esto se debe a que se usó un sensor 3D, el Kinect de Microsoft, con el cual se trabajó la implementación. La razón de esto es porque se quiere investigar en la tecnología de las nubes de puntos o "point clouds" y las aplicaciones que puede tener. Por otro lado, el Kinect es un sensor relativamente barato y fácil de implementar en muchas aplicaciones a diferencia de otros sensores 3D.

Este proyecto no solo representa la implementación de algoritmos para la navegación autónoma de vehículos por medio de sensores 3D, sino que su implementación se hace con el objetivo de poder usarlo en situaciones cotidianas, donde se le pueda facilitar la vida a las personas por medio de la robótica móvil. Por ejemplo, este tipo de proyecto puede utilizarse para transportar sustancias u objetos peligrosos en fábricas donde puede representar un riesgo para la integridad de una persona y la a vez se podría tratar de hacer el proceso más eficiente y organizado. Por otro lado, se le podría dar uso en aeropuertos, donde estos vehículos autónomos seguirían a los viajeros para cargar su equipaje y maletas. Este punto de vista se puede extender para ayudar a personas que tienen discapacidades físicas o adultas mayores que no pueden hacer esfuerzos o podrían hacerse daño si intentan transportar elementos por sí mismos.

Como puede verse, la realización de este proyecto representa un aporte a un campo que está comenzando a tener auge, cuyas contribuciones no solo se limitan al sector industrial sino que también pueden contribuir a mejorar la calidad de vida de las personas.

Desde el punto de vista de la ingeniería mecatrónica, este proyecto involucra distintos campos, como lo es la robótica, la electrónica, la mecánica, sistemas de visión, programación, entre otros; por lo que se constituye como un proyecto muy completo para optar por el título de ingeniero en mecatrónica.

## 1.2. Objetivos

### 1.2.1. Objetivo general

Implementar un sistema de procesamiento de información de sensores 3D para que un robot móvil o vehículo se mueva de forma autónoma, evitando obstáculos y a la vez mapee un lugar determinado con el fin de que alcance una coordenada específica.

### 1.2.2. Objetivos específicos

- Procesar la información generada a partir del sensor 3D.
- Diseñar algoritmos de navegación para el sistema, de la información obtenida anteriormente.
- Simular la aplicación que se quiere realizar por medio del programa V-REP, especial para aplicaciones en robótica.
- Diseñar la aplicación para la implementación del proceso simulado.
- Implementar el sistema utilizando los sensores en físico.

## 1.3. Organización del trabajo

El trabajo consta de seis capítulos. En el primer capítulo se realizó una introducción, se brindó una contextualización y motivación del por qué se llevó a cabo este proyecto en específico y también se presentaron los objetivos que se pretenden alcanzar.

En el Capítulo 2, se presentan aspectos bibliográficos. Primero se da una descripción de lo que es la robótica y conforme se avanza en el capítulo se guía al lector hacia el tema en específico de la robótica móvil. Se detalla un poco de los inicios de la navegación autónoma y los vehículos móviles, hasta mostrar los proyectos de mayor relevancia en esta área actualmente. Se explican también las técnicas y sensores necesarios para poder llevar a cabo el proyecto y por último se describen ambientes de simulación en los que podría probarse el trabajo realizado.

En el Capítulo 3, se describe a fondo el problema que se desea resolver y el diseño de ingeniería para escoger las mejores soluciones para cada requerimiento del proyecto. Una vez que se solventa cada requerimiento, se definen las especificaciones a partir de las cuales se desarrollará el trabajo.

En el Capítulo 4, se describe la parte principal del proyecto, la simulación. El capítulo explica el procedimiento para avanzar de una navegación donde hay un comportamiento totalmente reactivo y solo se evitan obstáculos, hasta llegar a una navegación inteligente donde se calculan las trayectorias para llegar hasta un punto específico. También se puede observar cómo se pone en práctica la teoría brindada en el Capítulo 2 y el diseño planteado en el Capítulo 3.

En el Capítulo 5, se explica la implementación realizada para verificar los datos o comportamientos obtenidos del capítulo anterior. Se utilizan los sensores reales, así como información real del sensor 3D.

En los últimos capítulos, se muestran los principales resultados del proyecto y el análisis de los mismos. Con el fin de determinar posibles trabajos futuros y los puntos

donde se podrían realizar posibles mejoras. También se muestran las limitaciones del proyecto y las conclusiones del mismo.

# Capítulo 2: Revisión bibliográfica

## 2.1. Robótica

En el año de 1921 aparece por primera vez la palabra robot. Esta fue usada por el novelista checo Karel Capek en su novela "*Rossum's Universal Robots*". En su idioma la palabra "robota" significa servidumbre o fuerza de trabajo.

Con el paso del tiempo, los robots han tomado un papel sumamente importante en las actividades realizadas por los seres humanos. Es por ello que ahora existen diversos tipos y con funciones especializadas. De acuerdo con Onwubolu (2005) existen básicamente dos tipos de robots: móviles y estacionarios. Los robots móviles tienen la libertad de moverse en el campo de trabajo. Los robots estacionarios se encuentran fijos en una posición (caso de un brazo robótico).

**Figura 1. Robot móvil al lado izquierdo y robot estacionario al lado derecho. Recuperado de: (Mihai, 2015)**

Los brazos articulados constituyen la mayoría de los robots industriales. Según el *Robot Institute of America*, un robot industrial es un manipulador programable multifuncional diseñado para mover materiales, piezas, herramientas o dispositivos especiales, mediante movimientos variados, programados para la ejecución de distintas tareas.

El desarrollo de robots móviles responde a la necesidad de extender el campo de aplicación de la robótica, que se encuentra limitado por el campo de trabajo definido por una estructura mecánica fija en uno de sus extremos. También busca minimizar en la medida de lo posible la intervención humana, con el fin de incrementar la autonomía. (Ollero, 2001)

Sin importar el tipo de robot con el que se esté trabajando, los robots en su mayoría están compuestos por diversos dispositivos y módulos. Los más relevantes son los sensores, por medio de los cuales se tiene una percepción del ambiente. Los sistemas de procesamiento de información también son una parte muy importante, ya que por medio de ellos se logra planificar qué hará el robot, se tomarán decisiones y en general se controlará al robot. Por último están los actuadores. Por medio de ellos se ejecutan las acciones del robot, generalmente a través de motores. (Sociedade Brasilera de Computação, 2015)

### 2.1.1. Inicios de la robótica móvil y navegación autónoma

Se puede decir que la robótica móvil inicia con el *Machina Speculatrix* de los años 50, el cual consistía en un robot móvil de tres ruedas diseñado por Walter Grey. Este robot móvil contaba con un sensor de luz, un sensor de contacto, un motor para la propulsión, un motor para direccionar el robot y dos tubos analógicos de vacío. A pesar del simple diseño que tenían, Grey logró demostrar que sus robots mostraban comportamientos complejos, debido a la tendencia especulativa para explorar el ambiente.

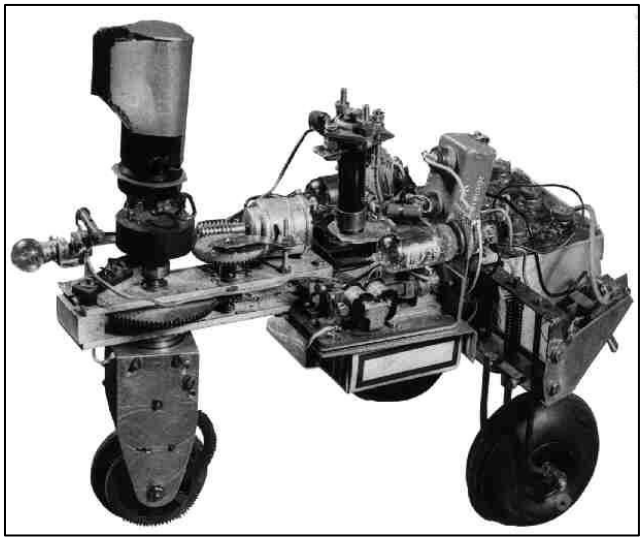


**Figura 2. Primer modelo del Machina Speculatrix de Walter Grey. Recuperado de: (Gasperi, 2015)**

A finales de los años 60, se desarrolló en la Universidad de Stanford el primer robot móvil controlado por visión, *Shakey*. Este robot tenía una única tarea, la cual consistía en reconocer un objeto usando su sensor de visión, para después encontrar su camino hacia él y realizar una acción sobre dicho objeto.

De igual forma en Stanford, en los años 70 se desarrolló un robot capaz de evitar obstáculos utilizando una cámara, CART. Este robot tomaba nueve fotos desde un punto específico para crear un modelo bidimensional del ambiente. Este robot fue bastante exitoso evitando obstáculos. Sin embargo, era muy lento y tenía problemas en obtener su ubicación exacta y a la vez visualizar obstáculos con carencia de alto contraste. (Nehmzow, 2003)

En 1984, Valentino Braitenberg, escribió el libro titulado "Vehículos", el cual trataba de un grupo de vehículos que pueden moverse de forma autónoma. Estos vehículos cuentan con sensores primitivos que permiten medir algún estímulo en un punto y también cuentan con ruedas que son accionadas por motores independientes, los cuales pueden verse como efectores. En su configuración más simple, la señal percibida por un sensor afecta directamente el accionamiento de un efector. De esta forma, el vehículo puede presentar comportamientos distintos. En determinadas situaciones, pueden comportarse

como atraídos hacia una señal u objeto específico y en otras tratar de evitarlos. Los aportes generados por Braitenberg en el área de navegación autónoma, siguen teniendo una gran importancia actualmente. (Braitenberg, 1986)

En los años 90, los robots móviles comienzan a utilizarse no solo para fines de investigación o académicos, sino que para aplicaciones en exploración. Por ejemplo, entre 1992 y 1994 se desarrollan en la Universidad de Carnegie Mellon el Dante I y el Dante II. Ambos eran robots caminantes, los cuales utilizaban sensores láser para monitorear el terreno y determinar un camino seguro para desplazarse. Su principal aplicación fue para la exploración en volcanes, con el fin de determinar la composición de gases emanados en los cráteres. (Carnegie Mellon University, 2015)

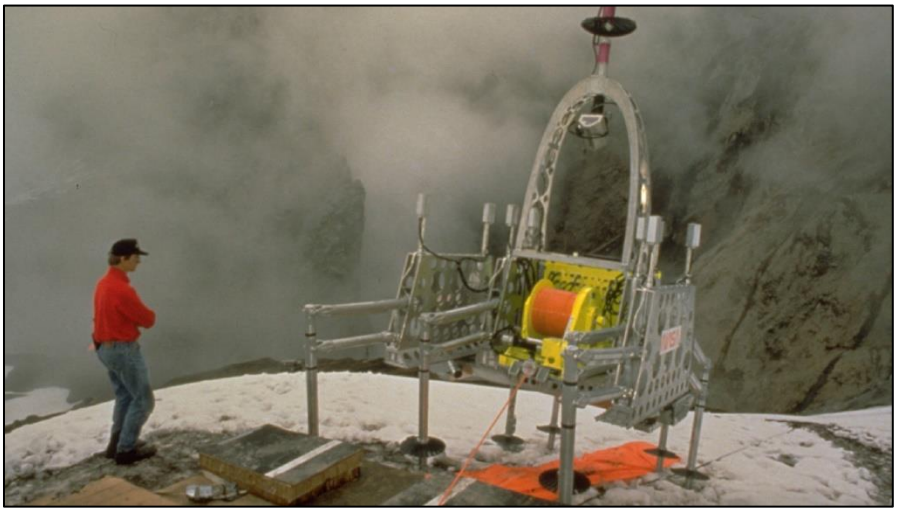


**Figura 3. Robot Dante II durante una misión de exploración en un cráter volcánico. Recuperado de: (Braga, 2014)**

A finales de los 90, otra de las aplicaciones más importantes de la historia de la robótica móvil se dio con el Mars Rover, sobre todo en el campo de la navegación autónoma. Este robot se diseñó con el propósito de explorar la superficie del planeta Marciano y a la vez enviar esa información a la Tierra. Este robot contaba con tres cámaras de navegación, dos blanco y negro y una a colores. (NASA, 2015)

En el año 2002 se presenta una aplicación de la robótica móvil enfocada en la vida cotidiana, la cual ha tenido mucha popularidad. Este paso se dio con *Roomba*, un robot

móvil autónomo que limpia el suelo y que actualmente se sigue vendiendo. En su primera versión, el movimiento del robot se realizaba en un patrón de espiral hasta encontrar una pared, luego su patrón cambiaba para funcionar como un seguidor de paredes. Este robot continuaba con estos patrones hasta que el tiempo de funcionamiento terminaba. (iRobot, 2015)

Si bien es cierto tratar de resumir la historia de la robótica móvil en unas cuantas páginas no es una tarea fácil, se han mostrado algunos de los ejemplos más importantes y que han marcado un hito en este campo, ya que gracias a ellos se ha llegado a consolidar esta área de la robótica a lo que es hoy.

### 2.1.2. Actualidad

En el año 2004 se dio por primera vez el *DARPA Grand Challenge*, evento que marcó un antes y después en la historia vehículos autónomos. El evento organizado por la *Defense Advanved Research Agency* de los Estados Unidos, tenía como propósito el desarrollo de tecnologías necesarias para la creación del primer vehículo terrestre completamente autónomo, capaz de completar una trayectoria fuera de carretera en el menor tiempo posible.

El evento se llevó a cabo en el desierto de Mojave y se debía recorrer una ruta de 240 km. Sin embargo, ninguno de los vehículos logró la terminar el evento. La mayor distancia recorrida fue de 11.78 km. Por esta razón, el premio no fue entregado y el próximo año se llevó a cabo la segunda edición de este evento. (DARPA, 2014)

En la segunda edición, casi todos los vehículos superaron la máxima distancia de la edición anterior y 5 de los 23 participantes lograron terminar de forma exitosa el evento. Posteriormente, en el año 2007 se llevó a cabo la tercera edición del evento, solo que esta vez se hizo en un ambiente urbano, en el que se debían recorrer 96 km con tráfico y obstáculos, donde se debían obedecer todas las regulaciones de tránsito. Todo el trayecto debía realizarse en menos de seis horas. Los tres primeros equipos lograron terminar la competición en menos de cuatro horas y cuarenta minutos. (DARPA, 2014)

Los avances conseguidos en estas competencias demostraron la relevancia que tiene esta área de investigación de la robótica, así como una tendencia para los próximos años. Ejemplo de ello es que en el año 2009, Google inició con uno de los proyectos más conocidos de la robótica móvil actualmente, el *Google Driverless Car*.

Este es en un vehículo autónomo capaz de conducir por las carreteras de las ciudades sin necesidad alguna de un conductor. Mediante el uso de sus sensores y software, puede detectar objetos como ciclistas, peatones, vehículos y más, y está diseñado para manejar de forma segura alrededor de ellos.

Para su funcionamiento, el vehículo está equipado con cámaras, sensores láser y radares que detectan objetos en todas las direcciones. Toda esta información es procesada por una computadora que está diseñada específicamente para el manejo autónomo.

Para lograr el manejo por sí solo, el primer paso que realiza el robot es preguntarse dónde está. Para ello, procesa tanto la información de los sensores como la de los mapas para obtener su posición en el espacio. El automóvil es capaz de saber en qué calle se encuentra así como el carril en el que va. El siguiente paso es detectar los objetos que se encuentran a su alrededor. Una vez detectados, el software los clasifica de acuerdo a tamaño, forma y patrón de movimiento.

Posteriormente, el software predice cómo podrían comportarse los objetos a su alrededor. Con esto, el vehículo ya puede escoger una velocidad segura así como la trayectoria del carro. (Google, 2015)

**Figura 4. Nuevo prototipo del Google Driverless Car. Recuperado de: (Inman, 2015)**

El éxito de estos vehículos ha sido tanto que en varias ciudades de los Estados Unidos ya se ha aprobado su uso en carreteras. En muchos estados ya están trabajando en una legislación para que puedan comenzar a funcionar y hasta el momento únicamente en dos estados se ha mostrado oposición a su funcionamiento. (Weber, 2014)

A nivel de Brasil, en el *Laboratório de Robótica Móvel* de la USP, se han desarrollado dos proyectos que han llamado mucho la atención del público y han demostrado los avances que se están logrando en esta área.

El primero de ellos es el proyecto CaRINA 2 (*Carro Robótico Inteligente para Navegação Autônoma*), el cual consiste en vehículo autónomo inteligente capaz de navegar en ambientes urbanos sin la necesidad de un conductor humano. CaRINA 2, es un vehículo marca *Fiat Palio Adventure*, al cual se le realizaron modificaciones mecánicas y eléctricas para su correcto funcionamiento. Los objetivos de este proyecto están enfocados en la disminución del número de accidentes en las carreteras y el aumento de la eficiencia del tránsito. También se pretende desarrollar un sistema de ayuda al conductor, por medio del cual se le notificaría de situaciones de riesgo durante la conducción.

Para su funcionamiento, el vehículo cuenta con los siguientes sensores: LIDAR (*Light Detection and Ranging*), cámara estéreo, GPS (*Global Positioning System*), IMU (*Inertial Measurement Unit*). Otros tipos de cámaras y radares se encuentran en pruebas para posteriormente agregarlos al sistema de sensores existentes.

Este automóvil ya ha realizado varias pruebas de forma exitosa en el campus II de la USP de la ciudad de São Carlos. Una vez que se demostró su correcto funcionamiento en esta ubicación, se procedió a probarlo en las carreteras de la ciudad en el año 2013. Durante la demostración, el vehículo recorrió algunas avenidas de la ciudad, identificando peatones y obstáculos, y manteniéndose a una distancia segura de ellos. (Laboratório de Robótica Móvel, 2015)

**Figura 5. Proyecto CaRINA 2. Recuperado de: (Laboratório de Robótica Móvel, 2015)**

El éxito de este proyecto fue tanto que posteriormente se trabajó en conjunto con la empresa *Scania*, para que el LRM desarrollara el primer camión autónomo de América Latina. Al igual que en el proyecto CaRINA, al camión se le realizaron modificaciones mecánicas y eléctricas para que pudiera ser totalmente controlado por computadora. Se utilizó un GPS de alta precisión para la localización y planeamiento de trayectoria. La detección de obstáculos fue realizada a través de los datos obtenidos por un radar y una cámara estéreo. El software de control, desarrollado en el LRM, es ejecutado por un computador empotrado en el vehículo. (Laboratório de Robótica Móvel, 2015)

Esta tendencia muestra que la tecnología de la navegación autónoma y robots móviles está teniendo una gran influencia en la vida de las personas. Muestra que es un campo en el que innegablemente se tiene que trabajar cada día más.

## 2.2. Sensores importantes en robots móviles

Una parte esencial de todo robot son los sensores, ya que por medio de ellos se puede tener una noción de su estado así como del ambiente que lo rodea, se puede tener la información necesaria para planificar qué acciones se deben ejecutar. Son una parte crucial en el funcionamiento de un robot, especialmente si es un vehículo autónomo.

Según la *Instrument Society of America*, un sensor puede definirse como un dispositivo que brinda una salida útil en respuesta a una medición específica. La salida es definida como una "cantidad eléctrica", la cual representa la medición de una "cantidad física, propiedad o condición".

Esta definición puede generalizarse al extender el término de "cantidad eléctrica" a cualquier tipo de señal como mecánica y óptica, y al extender "cantidad física, propiedad o condición" a aquellas de naturaleza química y biológica, entre otros. (Patranabis, 2003)

Los sensores tienen un sinfin de aplicaciones. En la robótica móvil generalmente lo que se busca es obtener información de la ubicación en el espacio y cómo eso puede ayudar a tomar acciones para moverse de forma inteligente. Algunos de los principales sensores que se utilizan para llevar a cabo estas tareas se describen en las siguientes secciones.

### 2.2.1. Sensores 3D

Este tipo de sensores permite medir la profundidad o distancia a un objeto al iluminarlo con rayos de luz o patrones específicos que son emitidos por láser u otro tipo de fuente de iluminación. Posteriormente, los rayos o patrones dispersos sobre él se miden por medio de receptores. (Horaud, 2014)

En su mayoría, los sensores 3D generan "*point clouds*" o nubes de puntos. Un "*point cloud*" es una estructura de datos que permite representar colecciones de puntos multidimensionales, los cuales se usan generalmente para representar información en tres dimensiones. La mayoría de las veces, los puntos de un "*point cloud*" 3D representan las coordenadas geométricas X, Y, Z de una superficie generada. Cuando se cuenta con la información de color de cada uno de esos puntos, se habla de un "*point cloud*" 4D. (PointCloud.org, 2015)

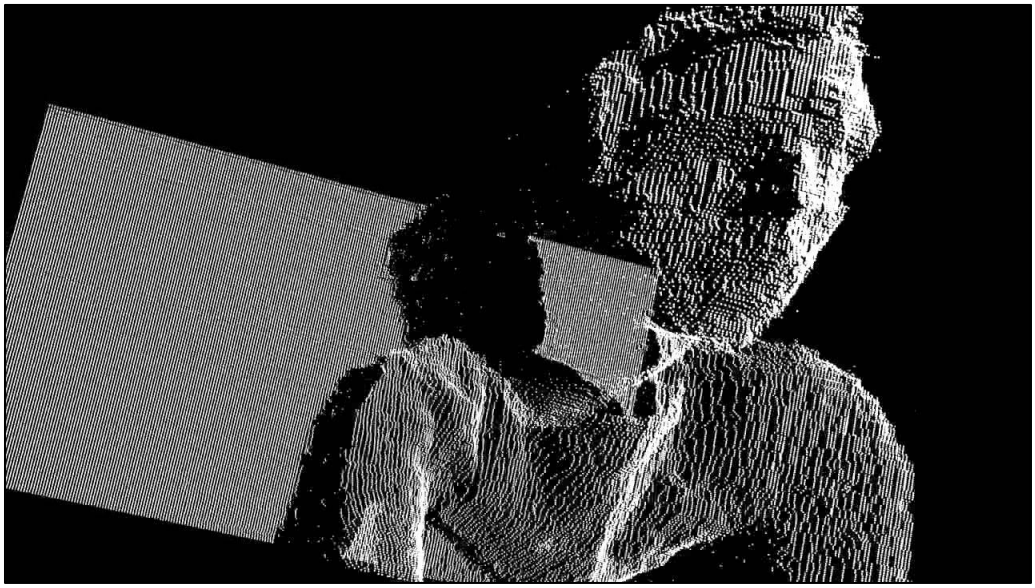


**Figura 6. Ejemplo de "point cloud" generado por un sensor Kinect. Recuperado de: (Gnecco, 2012)**

#### 2.2.1.1. Kinect

Al hablar acerca de este sensor, muchas personas pensarían que es solo una herramienta de juego para la consola de Xbox. Sin embargo, el Kinect es mucho más que eso. Se han desarrollado *drivers*, así como programas especializados que permiten recolectar los datos de este sensor, lo que ha permitido su adopción en la robótica.

Por medio de esos *drivers*, se obtienen los datos de las cámaras RGB e infrarrojas con las que cuenta el dispositivo. Para el cálculo de la profundidad de una imagen, el Kinect cuenta con un emisor de patrón cuadriculado NIR (*Near Infrared*). Posteriormente, este patrón es capturado por un sensor NIR que procesa la luz estructurada y así obtiene la profundidad de cada pixel. Por medio de una cámara normal se obtiene una imagen RGB. (Sociedade Brasilera de Computação, 2015)

El uso del Kinect tiene algunas limitaciones:

- El valor de profundidad de un píxel determinado tiene un margen de error: esto se debe a que el patrón de luz no cubre de forma continua algunas superficies, lo que conlleva a que algunos píxeles deban ser interpolados.

- Uso preferiblemente en ambientes internos: debido a que el sol emite luz infrarroja, los sensores de profundidad del Kinect pueden ver su funcionamiento afectado por interferencia de los rayos solares. (Salvatore, Osio, & Morales, 2014)

Estas son consideraciones de suma importancia que se deben tener en cuenta al trabajar con un sensor de este tipo.

#### 2.2.1.2. Cámaras estéreo

Este tipo de cámaras permite obtener la profundidad del ambiente al utilizar algoritmos que procesan las imágenes provenientes de dos cámaras desfasadas. La relación existente entre el mismo punto visto de diferentes posiciones permite calcular la profundidad de los elementos del ambiente. (Sociedade Brasilera de Computação, 2015)

#### 2.2.1.3. LIDAR

Los sensores LIDAR son aquellos que utilizan métodos de medición remota, donde se usa la luz en forma de un láser pulsante para medir distancias. Estos sistemas consisten principalmente de un láser, un escáner y un GPS especializado, sobre todo cuando se utilizan en aeronaves. (National Oceanic and Atmospheric Administration, 2015)

Los sistemas en tierra firme no requieren del GPS especializado y generalmente se encuentran montados en una base que gira 360°. Los pulsos del láser son reflejados por los objetos. Se registran aquellos que retornan y con eso se logra medir la distancia entre el objeto y el sensor. Un ejemplo de este tipo de sensor que se está utilizando mucho es el Velodyne. (LiDAR UK, 2015)


**Figura 7. Sensores tipo LiDAR marca Velodyne. Recuperado de: (Higgins, 2015)**

### 2.2.2. GPS

El *Global Positioning System* es una utilidad que permite saber a sus usuarios su ubicación en el globo, por medio de un sistema de veinticuatro satélites funcionales que orbitan el planeta y que continuamente transmiten información acerca de su ubicación.

Por medio de este sistema, el receptor de GPS puede calcular su ubicación al leer la información de los diferentes tiempos de llegada de la señal, con respecto a la distancia relativa a dos o más satélites. Gracias a esto, el receptor puede inferir su ubicación por medio de una triangulación.

Este tipo de sensor de posición es sumamente efectivo para aplicaciones de robots móviles al aire libre. Sin embargo, para aplicaciones en interiores no es tan efectivo, ya que en estos ambientes se presentan otros desafíos que en los exteriores no hay, como múltiples caminos y la dinámica del ambiente. (Siegwart & Nourbakhsh, 2004)

### 2.2.3. IMU

El *Inertial Measurement Unit* es un dispositivo electrónico que recopila información de la velocidad angular y la aceleración lineal para ser enviada a un procesador principal. El IMU en realidad contiene dos sensores separados. El primero es un acelerómetro en tres dimensiones. Este genera tres señales analógicas describiendo la aceleración que es producida en cada uno de los ejes. Generalmente, la aceleración más significativa medida por este sensor es la gravedad, debido a las limitaciones físicas del sistema.

El segundo sensor es el que mide el cambio angular en tres dimensiones. Este sensor tiene a su salida tres señales analógicas que describen el cambio angular en cada uno de los ejes del sensor. (University of Maryland, 2004)

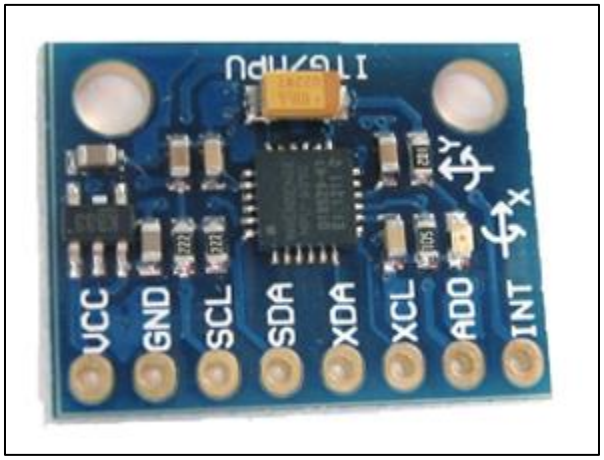



**Figura 8. Sensor MPU6050 tipo IMU de InvenSense. Recuperado de: (Goss, 2015)**

## 2.3. Métodos de navegación en vehículos móviles

### 2.3.1. Braitenberg

Como se mencionó anteriormente, en el año 1984 Braitenberg expuso, en su libro *Vehicles*, vehículos que navegaban de forma autónoma gracias al accionamiento de actuadores de forma proporcional a las mediciones realizadas por sensores.

El comportamiento de estos robots es reactivo; es decir, reaccionan en función de un estímulo, por lo que si este desaparece, la reacción también. Los comportamientos descritos por Braitenberg se pueden ejemplificar por medio de los siguientes vehículos:

- Vehículo 1: este vehículo cuenta con un motor (actuador) y un sensor únicamente, por lo que la fuerza del motor es proporcional a las mediciones del sensor. Por ejemplo, si se cuenta con un sensor de temperatura, el robot se desplazará con mayor velocidad en zonas calientes y con menor velocidad en zonas frías, y se detendrá en zonas donde la temperatura es cero.
- Vehículo 2a: este vehículo cuenta con dos sensores a cada lado y dos motores, uno izquierdo y otro derecho. Los sensores se encuentran conectados al motor de su respectivo lado. Esta conexión provocará que si el estímulo detectado por el sensor derecho es mayor, el motor de este lado girará con más velocidad haciendo que el robot se desplace a la izquierda. Debido a esto, el comportamiento de este robot es de escapar o alejarse del estímulo.
- Vehículo 2b: para este caso se tiene la misma configuración de sensores y motores que del vehículo 2a, solo que ahora los sensores se conectan a los motores de los lados opuestos. Esta nueva conexión provocará que si el estímulo detectado por el sensor derecho es mayor, el motor izquierdo se moverá con mayor velocidad provocando que gire hacia el lado contrario. Debido a esto, el comportamiento de este robot es de un seguidor del estímulo.

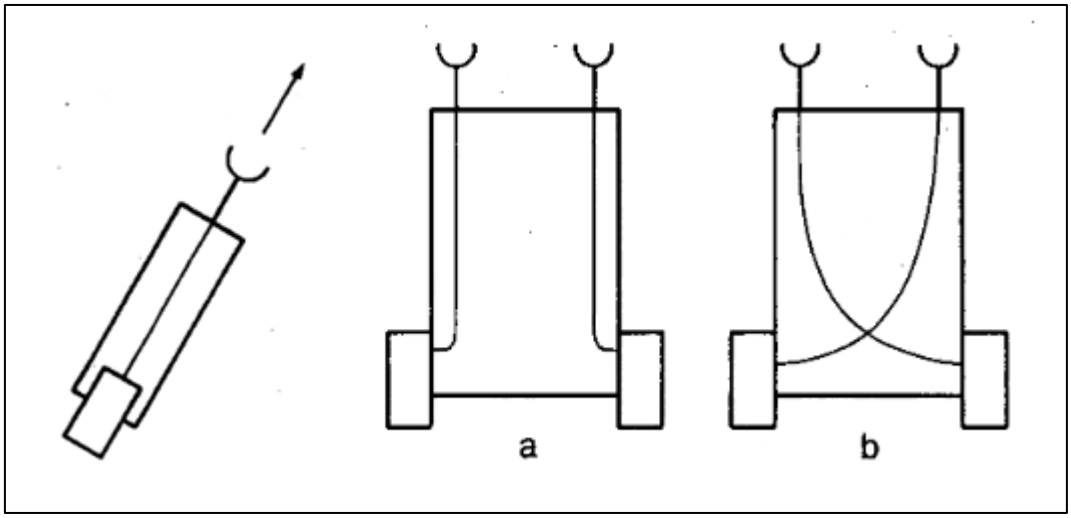



**Figura 9. Vehículos de Braitenberg, de izquierda a derecha: Vehículo 1, Vehículo 2a y Vehículo 2b. (Braitenberg, 1986)**

Los aportes de esta teoría de Braitenberg permiten conceptualizar mejor algunas aplicaciones de navegación. Por ejemplo, si se trabaja con robots móviles que tengan características similares de configuración de los motores y sensores, se pueden generar algoritmos para evitar obstáculos adoptando un comportamiento como el del vehículo 2a. También se podrían generar algoritmos para alcanzar una meta si se adopta un comportamiento como el del vehículo 2b. (Braitenberg, 1986)

### 2.3.2. Campos potenciales

Este método de navegación se basa en la idea de concebir al robot como una partícula que se ve influenciada por la fuerza de campos artificiales de atracción y repulsión. El espacio por el que se desplaza la partícula es el resultado de la suma de un campo potencial de atracción, el cual posee una fuerza que jala la partícula hacia el destino o meta, y de un campo de repulsión, que contiene fuerzas que la alejan de los obstáculos.

Si se define el campo de atracción como:

$$U_{atr}(q) = \frac{1}{2}\,\xi\,\rho^{2}_{meta}(q) \qquad (2.1)$$

Donde $U_{atr}(q)$ es el campo potencial de atracción de la meta que actúa sobre la partícula, la constante $\xi$ es un factor de escala positivo que permite modificar el alcance del campo potencial y $\rho_{meta}(q)$ denota la distancia euclidiana de la posición de la partícula a la posición meta $\| q - q_{\text{meta}} \|$.

La fuerza artificial de atracción se puede obtener del gradiente del campo de atracción descrito anteriormente, lo que da como resultado la siguiente ecuación:

$$\vec{F}_{\text{atr}}(q) = -\vec{\nabla} U_{atr}(q) \tag{2.2}$$

$$\vec{F}_{\text{atr}}(q) = -\xi (q - q_{meta}) \tag{2.3}$$

La función $U_{atr}(q)$ es positiva o nula, y alcanza su valor mínimo en $q_{meta}$, donde su valor es cero.

Por otro lado, el campo potencial de repulsión viene dado por la siguiente fórmula:

$$U_{rep}(q) = \begin{cases} \frac{1}{2}\,\eta \left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right)^2 & si\ \rho(q) \leq \rho_0 \\ 0 & si\ \rho(q) > \rho_0 \end{cases} \tag{2.4}$$

Donde $\eta$ es un factor de escala positivo que permite modificar el alcance del campo potencial. La función $\rho(q)$ es la distancia euclidiana que hay entre la partícula y el obstáculo más cercano, $||q - q_{obs}||$. La variable $\rho_0$ representa la distancia a partir de la cual comienza ejercer su efecto el campo de repulsión. La fuerza de repulsión respectiva a este campo se muestra en la siguiente ecuación:

$$\vec{F}_{\text{rep}}(q) = -\vec{\nabla} U_{rep}(q) \tag{2.5}$$

$$\vec{F}_{\text{rep}}(q) = \begin{cases} \eta \left(\frac{1}{\rho(q)} - \frac{1}{\rho_0}\right) \frac{1}{\rho^2(q)} \frac{q - qobs}{||q - qobs||} & si\ \rho(q) \leq \rho_0 \\ 0 & si\ \rho(q) > \rho_0 \end{cases} \tag{2.6}$$

Una vez que se obtienen los campos potenciales de atracción y repulsión y sus respectivas fuerzas, se puede obtener el campo potencial resultante por el cual se desplazará el robot o partícula. La fuerza resultante de este campo es producto de sumar la de atracción

y la de repulsión. Con este vector de fuerza se puede tener la dirección en la que hay que desplazarse así como una velocidad proporcional al módulo de este vector.

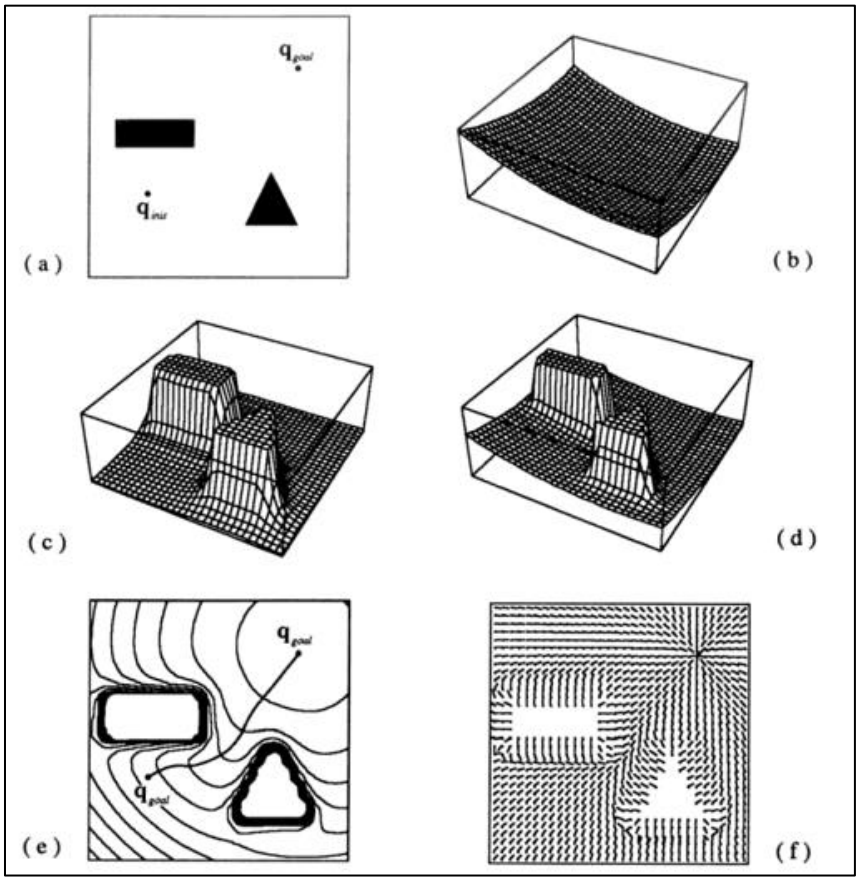



**Figura 10. En la Figura a, se muestra la configuración de un espacio bidimensional con obstáculos. En la Figura b y Figura c, se muestra el campo potencial de atracción de la meta y el de repulsión de los obstáculos, respectivamente. En la Figura d, el campo potencial resultante. En la Figura e, el camino generado por los campos y en la Figura f, el gradiente de vectores negados en el espacio. (Latombe, 1991)**

Este es un método de navegación elegante. Sin embargo, no es tan eficiente en situaciones en las que pueden presentarse mínimos locales. Los mínimos locales son zonas en las que la función potencial es igual a cero pero no es el punto meta, por lo que el robot se queda atrapado en ellos sin alcanzar su objetivo. (Latombe, 1991)

Siempre que se implemente este método, se debe tener en consideración la susceptibilidad a este problema. Si se combinan los campos potenciales con otras técnicas de navegación, se puede reducir significativamente el efecto de los mínimos locales.

### 2.3.3. Waypoints

Este método de navegación es muy popular en sistemas que cuentan con GPS. Los "*waypoints*" son una serie de puntos o coordenadas que definen una trayectoria de navegación. Por ejemplo, para desplazarse de un punto inicial A hasta un punto de destino B, se puede hacer por medio de una trayectoria directa entre estos dos puntos o se puede dividir el trayecto en trechos o etapas. De esta forma, se puede definir una ruta más eficiente y con un movimiento más suave entre el inicio y el final.

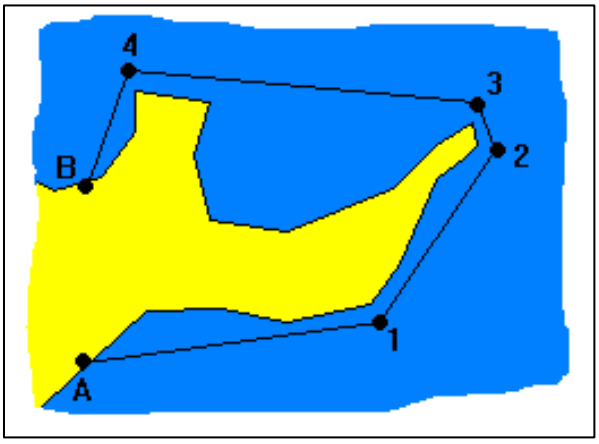



**Figura 11. Trayectoria entre los puntos A y B definida por una serie de "waypoints". Recuperado de: (Reis, 1996)**

En la imagen anterior se puede ver un ejemplo de "*waypoints*", donde la trayectoria entre A y B está definida por etapas menores entre puntos intermedios, por lo que para llegar desde A hasta B, se debe pasar primero por los puntos 1, 2, 3 y 4. Este método es de gran ayuda cuando se quiere tratar de evitar zonas que pueden ser conflictivas durante una navegación. (Reis, 1996)

## 2.4. Entornos de simulación

Antes de poner en práctica la teoría, cálculos y diseños que se han realizado, siempre es bueno contar con una herramienta o alternativa que permita validar el trabajo hecho. Por esto, los simuladores son una excelente opción en estos casos, ya que se pueden corregir situaciones que no se habían contemplado así como errores que pueden suceder.

En el campo de la robótica móvil es de suma importancia contar con este tipo de recursos. Llevar a cabo la implementación de un proyecto puede ser muy cara y si se puede saber de antemano o al menos tener una idea de si realmente el trabajo realizado se apegará a la teoría, se puede realizar una implementación más barata y eficiente. De igual forma, se pueden mejorar los algoritmos que procesan la información de los sensores y hacen actuar al robot.

Existen muchos programas para la simulación de robots, los cuales incluyen tanto robots móviles como robots de base fija, además de gran cantidad de sensores y herramientas para hacer cálculos cinemáticos y dinámicos. En las siguientes secciones se hablará un poco acerca de algunos simuladores, dándole mayor atención al simulador V-REP, ya que en este programa se llevó a cabo la simulación del proyecto.

### 2.4.1. ROS

ROS (*Robot Operation System*) es una colección de herramientas, librerías y convenciones que sirven para simplificar la escritura de software para robots. Su propósito es simplificar la tarea de crear los complejos y robustos comportamientos de un robot, siendo aplicable en distintas plataformas robóticas.

ROS es una herramienta de código abierto y de gran popularidad actualmente. Al ser de código libre, cuenta con una comunidad internacional trabajando constantemente en su mejoramiento y desarrollo. Se puede tener acceso a muchas librerías y paquetes que son de gran utilidad para el desarrollo de aplicaciones, principalmente en robots reales. Su uso es tanto en investigación y educación como en desarrollo de productos. (ROS, 2015)

### 2.4.2. Webots

Este simulador permite modelar, programar y simular robots móviles. Con este programa, el usuario puede diseñar complejas configuraciones robóticas, ya sea con uno o más robots similares o diferentes, en un ambiente compartido. Hasta el momento, Webots es utilizado en más de 1 200 centros de investigación y universidades.

El control del robot puede programarse directamente desde el ambiente de desarrollo integrado que posee o también pueden usarse otros ambientes de desarrollo compatibles con Webots. Los programas de control tienen la opción de que pueden transferirse a robots comerciales disponibles.

Este simulador no es de código libre, por lo que se tiene que comprar la licencia ya sea para uso comercial o educacional. Algunos modelos de robots móviles también se pueden comprar en caso de que sea necesario. (Cyberbotics, 2015)

### 2.4.3. Player/Stage/Gazebo

*Player* es un proyecto de código libre, donde su servidor probablemente es una de las interfaces para control de robots más usadas alrededor del mundo. Esta plataforma junto con sus ´*backends*´ *Stage* y *Gazebo* son ampliamente usados en la investigación de robots y sistemas de sensores. *Player* es desarrollado por un equipo internacional de investigadores en robótica y es usado en laboratorios en todo el globo.

*Player* brinda una interfaz en línea con variedad de robots y sensores. Su modelo permite escribir los programas de control del robot en cualquier lenguaje y posteriormente correrlos en cualquier computador que posea conexión en línea con el robot.

*Stage* permite simular poblaciones de robots en ambientes bidimensionales. Este simulador tiene una interfaz estándar de *Player*, por lo que no se requieren muchos cambios cuando se pasa de la simulación al hardware. Muchos controles diseñados en *Stage* han demostrado que funcionan en robots reales.

*Gazebo* es un simulador robótico 3D. Al igual que con *Stage*, con este programa se pueden simular poblaciones de robots, sensores, objetos, así como interacciones entre ellos, ya que incluye simulaciones de física de cuerpos rígidos. Aparte de dos interfaces nativas con las que cuenta, también tiene una interfaz de *Player*. (Player, 2014)

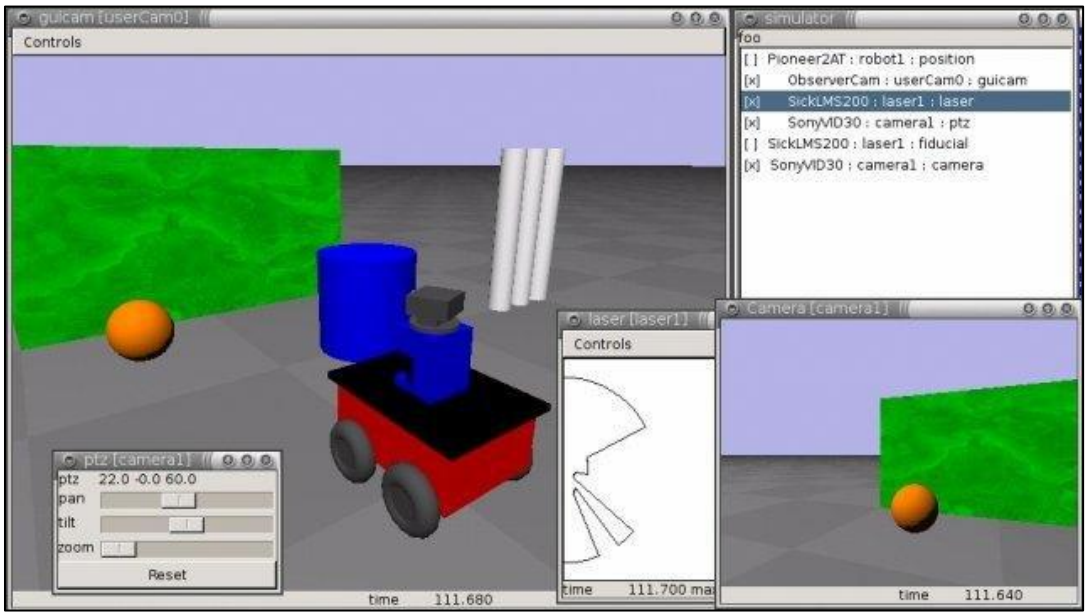



**Figura 12. Entorno de simulación de Gazebo. Recuperado de: (Player, 2014)**

### 2.4.4. V-REP

El simulador V-REP cuenta con un ambiente de desarrollo integrado. Este ambiente está basado en un control distribuido de arquitectura: cada objeto o modelo puede ser controlado individualmente por medio de un *script* o código propio, un *plugin* de ROS, un cliente remoto API, o una solución propia. Los controles pueden ser escritos en lenguajes como C/C++, Python, Java, Lua, Matlab, Octave o Urbi.

V-REP es usado para el desarrollado rápido de algoritmos, simulaciones de fábricas automatizadas, verificación rápida de prototipos, monitoreo remoto, entre otros.

Este simulador cuenta con una versión educacional que es de licencia gratuita. Sin embargo, también hay una versión *pro* que requiere una licencia comercial. En ambas versiones se cuenta con las siguientes características:

- Motores para cálculos físicos y dinámicos como *Bullet*, ODE, *Vortex* y Newton. Los cuales permiten simular condiciones físicas del mundo real así como la interacción entre objetos.
- Herramientas de cálculos de cinemática inversa y directa para cualquier tipo de mecanismo (Con extremidades, cerrado, redundante, entre otros).
- Detección de colisiones.
- Simulación de diversos sensores de proximidad y visión, así como herramientas para personalizarlos.
- Registro de información y visualización, interfaces de usuario personalizables, fácil importación y exportación de modelos, entre muchas otras funciones. (Coppelia Robotics, 2015)

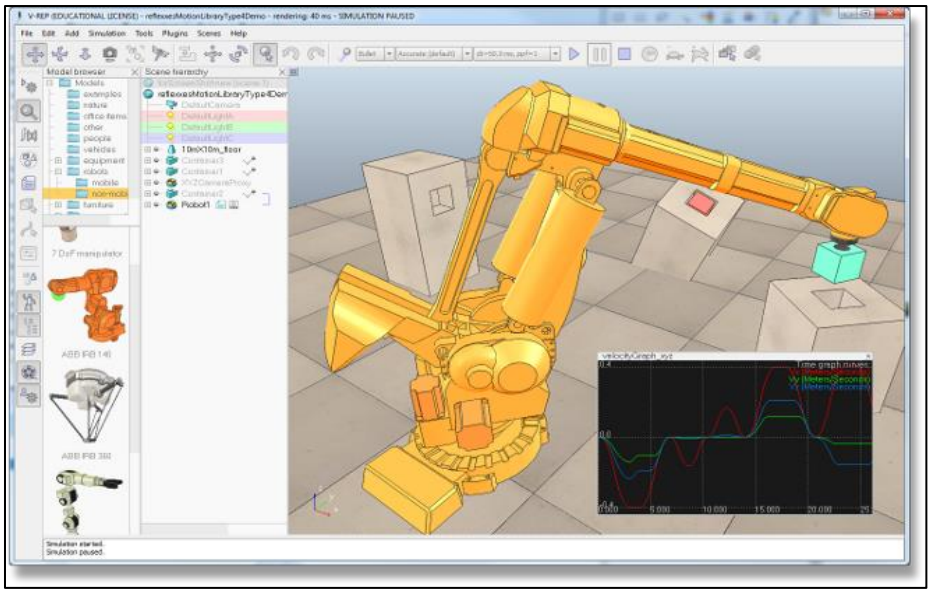


**Figura 13. Entorno de simulación de V-REP. Recuperado de: (Coppelia Robotics, 2015)**

## 2.5. Consideraciones finales

En el capítulo anterior se presentaron aspectos importantes de la teoría de la robótica móvil. Se dio a conocer un poco acerca de la historia de esta área, los avances que se han realizado, así como los proyectos que se están desarrollando actualmente, con el fin de demostrar las tendencias hacia las que se dirige está área de la robótica.

Se describieron tecnologías que permiten facilitar la tarea de un robot móvil para desplazarse en el ambiente y tener una noción de él. También, se dio una introducción a distintas técnicas de navegación que son de gran utilidad para llevar a cabo este proyecto.

Por último, se mostraron distintos ambientes de programación en los cuales se puede simular la aplicación que se desea implementar, así como las distintas funcionalidades que tienen. Toda la información brindada en este capítulo, pretende facilitar al lector la comprensión de la solución implementada en este proyecto.

Una vez que se cuenta con estos conceptos básicos y conocimiento del tema, se puede apreciar desde un mejor punto de vista el procedimiento para llevar a cabo este proyecto. En el siguiente capítulo se explicará a fondo el problema que se desea resolver así como el entorno en el que se llevará a cabo el proyecto.

# Capítulo 3: Generalidades del proyecto

En este capítulo se da una descripción más detallada del diseño de ingeniería, para definir los aspectos necesarios para solucionar el problema planteado. En cada sección se brinda al lector la información necesaria para que pueda entrar en contexto con el proceso de desarrollo del proyecto, así como las razones por las cuáles se seleccionó un determinado elemento o técnica como la mejor solución a un requerimiento.

## 3.1. Entorno del proyecto

Desde hace muchos años la Universidad de São Paulo ha destacado por la investigación y el desarrollo de proyectos, teniendo un gran impacto en áreas como la ciencia y la tecnología. Específicamente, el *Laboratorio de Robótica Móvel* de esta universidad en la sede São Carlos, está trabajando en el área de navegación autónoma e inteligente de vehículos y robots móviles. Para ello, se implementan sensores de muy alta tecnología, como sensores láser para monitoreo 3D de superficies, sensores de proximidad y ultrasónicos; así como sistemas de visión y algoritmos computacionales, en vehículos o robots ya existentes para que logren esa autonomía e inteligencia que antes no poseían.

Los propósitos de estos proyectos son varios, desde conducción autónoma en ambientes urbanos hasta aplicaciones en industria, agricultura y comercio. Uno de los proyectos más recocidos de este laboratorio es CaRINA, el cual es un automóvil al que se le han implementado sensores, así como modificaciones mecánicas y electrónicas para que maneje de forma autónoma sin ayuda alguna por las calles de São Carlos.

El proyecto desarrollado a continuación tiene como propósito utilizar sensores de tecnología 3D con el fin de detectar obstáculos y así poder navegar autónomamente evitando colisiones y llegar a un punto específico. La información o "*point clouds*" obtenidos de estos sensores será procesada y simulada para generar algoritmos de navegación eficaces. Con ello se pretende desarrollar una aplicación que logre que el robot desempeñe las funciones descritas anteriormente.

## 3.2. Descripción del problema

### 3.2.1. Generalidades

La navegación autónoma de vehículos así como el mapeo de zonas tiene un sinfín de aplicaciones. Una de gran peso e importancia es en zonas de riesgo o peligrosas, así como en las que han ocurrido desastres naturales. En estas situaciones, es muy peligroso enviar a una persona a realizar algún trabajo, ya que puede resultar herida o en el peor de los casos perder la vida. Por esta razón, un vehículo o robot autónomo que pueda llevar a cabo este trabajo es una excelente solución.

Por otro lado, este tipo de vehículos tiene aplicaciones muy importantes a nivel comercial e industrial. Por ejemplo, a nivel industrial, un vehículo capaz de llegar a una coordenada específica, con la capacidad de evitar obstáculos y evitar colisiones con objetos o personas, sería de gran provecho ya que se pueden mejorar los procesos productivos y la eficiencia de las fábricas. De igual forma, contribuiría con la seguridad de los trabajadores en caso de que se tengan que transportar materiales que representen un riesgo para la salud o la integridad física.

En el sector comercial, se pueden realizar aplicaciones para facilitar la vida cotidiana, donde un robot autónomo con las características descritas anteriormente, sea capaz de seguir a una persona con el fin de cargarle sus pertenencias. Por ejemplo, en un aeropuerto, se podrían usar como un servicio de transporte de maletas para los viajeros, en supermercados para hacer las compras y así con un sinfín de aplicaciones en distintas áreas. Se podría usar también para llevar las pertenencias de personas adultas mayores o de personas con alguna discapacidad y con esto evitar que sufran alguna lesión.

Con este proyecto, se tendrá una aplicación capaz de navegar autónomamente por distintos ambientes, evitando obstáculos de forma reactiva y con la que se podrá alcanzar la coordenada de un objetivo específico

El propósito del proyecto es implementar una aplicación que permita adecuar un robot móvil o vehículo para que sea capaz de hacer lo mencionado anteriormente.

### 3.2.2. Síntesis del problema

El problema consiste en implementar una aplicación por medio de sensores 3D para que un robot móvil o vehículo sea capaz de desplazarse autónomamente, evitando obstáculos, y sea capaz de llegar a una coordenada específica y con ello evitar problemas en la integridad física de las personas que requieran transportar materiales u otros objetos.

## 3.3. Enfoque de la solución

Para la solución del proyecto, la navegación se hizo con base en sensores 3D. Estos sensores generan "*point clouds*", los cuales, al ser procesados por diferentes algoritmos computacionales, permiten realizar un mapeo del lugar en el que se encuentran. Primero, se debió simular el funcionamiento de este sensor junto con el robot en un programa llamado V-REP, el cual es especial para aplicaciones de robótica móvil.

Una vez realizada la simulación, se obtuvieron los datos de ella para procesarlos y generar algoritmos de navegación autónoma. En otras palabras, para que el robot o vehículo fuera capaz de moverse, detectar obstáculos de un ambiente específico y evitarlos. Luego, los algoritmos debieron ser validados para realizar la implementación física con los sensores. Pero antes de proceder con la implementación de los sensores, se debió diseñar una aplicación para que se comunique adecuadamente. Al final, con todos los elementos integrados, se realizaron múltiples pruebas para verificar el correcto funcionamiento del proyecto.

## 3.4. Metodología de trabajo

### 3.4.1. Requerimientos

Para tener un mejor planeamiento del desarrollo del proyecto se definen los siguientes requerimientos:

1. La aplicación debe ser diseñada para un robot móvil con movimiento por medio de ruedas.
2. El sistema debe ser capaz de llegar a un punto específico determinado por el usuario.
3. El sistema debe usar un sensor de tecnología 3D como base para su navegación.
4. Conforme el robot se desplaza, este deberá ser capaz de detectar obstáculos.
5. Una vez detectado el obstáculo, el robot deberá ser capaz de evitar colisiones con él.
6. La aplicación debe tener la capacidad de reconocer la ubicación del robot en el ambiente.
7. La aplicación debe tener la capacidad de reconocer la orientación del vehículo conforme se mueve.
8. La aplicación debe poder seguir una trayectoria definida por el usuario.
9. La aplicación debe accionar el movimiento del vehículo de forma autónoma.
10. El sistema debe modificar la velocidad del robot, en función de la distancia a la que se encuentra de la meta y los obstáculos.

### 3.4.2. Tabla morfológica de requerimientos y análisis de opciones

**Tabla 1. Tabla morfológica para las posibles soluciones a cada requerimiento del sistema. Elaborado por el autor.**

| Requerimiento | | | | |
|---|---|---|---|---|
| **1** | Robot de tracción diferencial | Robot tipo triciclo | Robot Omnidireccional | |
| **2** | GPS | Seguidor de señal | Control remoto | |
| **3** | LIDAR | Kinect | Sonares 3D | |
| **4** | Sensor ultrasónico | Sensor láser | Kinect | Sensor capacitivo |
| **5** | Campos potenciales | Redes neuronales | Respuesta directa en función de la distancia a los obstáculos | |
| **6** | GPS | Red celular | Wifi | |
| **7** | IMU | Compás | Sensor de efecto hall | |
| **8** | Waypoints | Seguidor de línea | Mapa precargado | |
| **9** | Computador con programa | Microcontrolador | | |
| **10** | Campos potenciales | Respuesta directa en función de la distancia a los obstáculos | | |

Para mejorar el proceso de diseño de la aplicación, se elaboró la tabla morfológica mostrada, en relación a los requerimientos planteados en la sección anterior. Para cada uno de ellos, se enunciaron posibles alternativas con las cuales se podría desarrollar una solución. A continuación se analizarán dichas alternativas para saber cuál de ellas es la mejor o más adecuada para llevar a cabo el proyecto.

Para el primer requerimiento se presentan varias opciones de robots móviles con movimiento por medio de ruedas. La primera alternativa propuesta es el robot diferencial. Este tipo de robot es aquel que cuenta con dos ruedas accionadas por motores independientes y una tercera rueda que sirve de apoyo (en algunos casos pueden ser de cuatro ruedas). La diferencia de velocidad en las ruedas del robot permite que este gire en una dirección específica o que continúe en línea recta si se mantiene la misma velocidad en ambas.

La ventaja de este tipo de robot es que posee la cinemática más sencilla de los que se mueven por medio de ruedas. Esta característica permite que sea más fácil controlar el robot. Por otro lado en el LRM se cuenta con un robot de este tipo por lo que la aplicación también debe ir en función de los elementos con los que cuenta en el laboratorio.

La segunda opción es utilizar un robot tipo triciclo. Para este tipo de robot se cuenta con dos ruedas que brindan la tracción y una tercera que brinda el sentido de dirección en el que se debe mover el robot. El modelo cinemático de este robot es más complejo que el del robot anterior aunque su uso es muy común.

El robot omnidireccional presenta la ventaja de que puede moverse en cualquier dirección sin la necesidad de reorientarse. La desventaja de este sistema es la sincronización que se debe poseer entre las ruedas para un correcto desplazamiento del robot. Esto hace que el modelo cinemático sea más complejo y por ello también su control.

Al analizar las opciones que se tienen respecto al primer requerimiento, se determina que lo mejor es hacer la aplicación para un robot de tracción diferencial. Esto debido a que se cuenta con robot de este tipo en el LRM y se desarrollaría una aplicación para uno de los recursos con los que cuenta el laboratorio. Por otro lado, se presenta la ventaja de que el modelo cinemático de este sistema es el más simple de todas las opciones, por ello es que se tomó esta decisión.

Definido el tipo de plataforma para la que se desarrollará la aplicación, se define el método para determinar cómo llegar a un punto específico indicado por el usuario. La primera opción que se presenta es el GPS. La ventaja de usarlo es que se pueden obtener de forma sencilla los datos de un receptor GPS, por medio de una comunicación serial.

También presenta la ventaja de que, en caso de ser necesario, la técnica de recepción de datos es muy similar a otras técnicas cómo triangulación por señal de red celular o de wifi. Por esto, en caso de ser necesarias modificaciones para funcionamiento en otros ambientes, no se requerirían muchos cambios. La desventaja de este método es que el GPS presenta un error importante asociado a la señal, por lo que si no se combina con otros métodos puede afectar la precisión de la lectura de los datos.

La segunda opción es utilizar un seguidor de señal. Esto es un sistema que modifique su comportamiento conforme detecta una señal con mayor intensidad, como el caso de un seguidor de luz, una señal de radiofrecuencia, una señal infrarroja, entre otros. Sería un comportamiento seguidor de estímulo como el del vehículo 2b de Braitenberg (Braitenberg, 1986). La desventaja de este sistema es que no es un método muy preciso y se dependería del alcance de la señal, además de que se deberían colocar distintos emisores. Sin embargo, puede ser una opción muy barata de implementar.

Por último, se presenta la opción de ser controlado de forma remota. Como existe otro requerimiento de que el vehículo debe accionarse de forma autónoma y debido a la naturaleza del proyecto, esta opción queda descartada.

Analizando las dos opciones restantes, se puede ver que el que presenta más facilidad de uso es el GPS, ya que es un sistema muy generalizado. Aunque el seguidor de señal puede ser una opción más barata, en el laboratorio ya se cuenta con un receptor de GPS y por otro lado esta es la opción que se tiene para este tipo de función en el simulador. Por esta razón, se selecciona el GPS como el método para llegar a un punto específico, siempre teniendo en cuenta que su error asociado puede modificar el comportamiento del sistema.

Para el tercer requerimiento, se debe escoger un tipo de sensor 3D para trabajar la navegación. El primero que se presenta es el LIDAR. Como se habían mencionado anteriormente, este sistema funciona por medio de un láser pulsante. Este sistema es muy preciso y tiene un largo alcance. Sin embargo, su precio es muy elevado. Por ejemplo, un LIDAR modelo VPL-16 de Velodyne tiene un precio de $7999. (Higgins, 2015)

El Kinect es otra alternativa para este requerimiento. Aunque no es tan preciso ni tiene un alcance tan amplio como el sensor anterior, este sensor cuenta con una amplia documentación y acceso a su método de funcionamiento. Su alcance máximo es de 3.3 metros, por lo que brinda la posibilidad de detectar obstáculos dentro de un rango en el que se puede maniobrar con facilidad. Por otro lado, este sensor es mucho más barato, su modelo más nuevo se puede comprar por menos de $200 (Duarte, 2013). La principal desventaja de este sensor es que su funcionamiento puede verse afectado por una exposición directa a la luz solar.

Los sonares 3D son buenas opciones para mapeo, pero su uso se limita fundamentalmente para aplicaciones acuáticas. En robots terrestres no es muy común su uso y la información disponible no es mucha. Por otro lado, también presentan precios sumamente elevados.

De las tres opciones, se podría decir que la que presenta el mejor costo-beneficio es el Kinect, debido a su precio y versatilidad. Por otro lado, para esta aplicación no se requiere un grado de exactitud tan elevado como para usar alguno de los otros sensores mencionados. Por lo tanto, se escoge el Kinect como el sensor de tecnología 3D.

El cuarto requerimiento hace referencia a la detección de obstáculos. Para ello, existen varias opciones de uso de sensor, y se debe definir el tipo de tecnología que se utilizará. Si se trabaja con sensores capacitivos y ultrasónicos se requerirían varios sensores ya que sus mediciones son muy puntuales y no abarcan ángulos y distancias muy grandes. Por otro lado, al tener una mayor configuración de sensores el sistema se torna más complejo.

El sensor láser y el Kinect presentan la ventaja de que pueden abarcar un mayor ángulo de medición, así como mayores profundidades. Sin embargo, este requerimiento se encuentra sumamente ligado al anterior, por lo que se puede tener una aplicación más simplificada y eficiente al utilizar un único sensor que pueda cumplir con dos requerimientos. Por ello se define el Kinect como la solución adecuada.

Para el requerimiento de evitar obstáculos, se presentaron tres opciones. Una de ellas es utilizar redes neurales para reconocer obstáculos. Esta es una buena opción ya que

por medio de esta técnica de programación se pueden reconocer patrones así como elementos que el usuario puede definir cómo obstáculos. Esta sería una buena técnica, aunque el nivel de complejidad que posee es muy elevado y existen otros métodos que pueden presentar resultados igualmente buenos, por lo que se descarta esta opción.

La segunda opción es utilizar la teoría de campos potenciales. Como se había explicado en el capítulo anterior, esta técnica permite modelar al robot como una partícula que se ve afectada por campos de fuerzas imaginarios y con ello determinar el camino que debe tomar el robot. Aunque no es un método sencillo y pueden existir técnicas similares, esta opción destaca ya que permite al usuario determinar la trayectoria a seguir de forma reactiva, es decir, con mediciones de sensores como GPS y Kinect. La principal desventaja de esta teoría es que presenta mínimos locales, o puntos donde la suma de las fuerzan que actúan sobre el robot son cero pero en realidad no es el punto meta. Si este método se combina con diferentes técnicas, se puede reducir considerablemente este efecto.

Por último, se encuentra la opción de establecer una respuesta directa en función de la distancia a la que se encuentra de los obstáculos. Por ejemplo, se puede definir una función lineal que dentro de un rango de distancias modifique la velocidad de las ruedas para que gire en una dirección o gire sobre su propio eje. Este es un método sencillo. Sin embargo, se requiere conocer la cinemática del vehículo en el que se trabajará para obtener una respuesta adecuada.

Las dos últimas opciones brindan buenas soluciones para evitar obstáculos. Los campos potenciales son una solución más elegante y la respuesta directa es un método más sencillo. Estas dos opciones se pueden combinar para obtener un algoritmo más completo para evitar obstáculos y lograr una navegación más precisa y eficiente.

De igual forma, estas dos opciones pueden solventar el requerimiento número diez, ya que los campos potenciales brindan una velocidad proporcional a las fuerzas imaginarias que actúan sobre el robot, ya sea que se encuentre cerca de un obstáculo, la meta o ambos. Con la respuesta directa en función de la distancia a los objetos, también se puede lograr esto. Como se había mencionado anteriormente, combinar estas dos opciones puede presentar mejores resultados de navegación y de control de velocidad del robot.

El requerimiento número seis está muy relacionado con el requerimiento número dos, ya que para ello se pueden utilizar los mismos sensores. Las tres opciones presentadas tienen un funcionamiento similar, donde la ubicación es determinada por medio de la triangulación de distintas señales. El principal aspecto a considerar es el ambiente en el que se utilizará la aplicación. Por ejemplo, el GPS funciona muy bien para aplicaciones al aire libre, el wifi y la red celular funcionan mejor que el GPS para ambientes internos.

Sin embargo, para fines de simulación, se tiene que utilizar un GPS, ya que es el único sensor que se tiene de este tipo para determinar la posición del robot. Se debe tener en consideración que esta técnica es escalable en caso de que para la aplicación se quiera trabajar en ambientes internos. Tanto para wifi como para red celular, la forma de trabajar los datos es muy similar por lo que en caso de ser necesario utilizarlos, el trabajo requerido no sería tan exhaustivo. Como se había mencionado anteriormente, cuando se trabaja con este tipo de sensores, se debe tener en cuenta el error asociado con las mediciones que realizan.

Reconocer la orientación que posee el vehículo es una tarea que se puede realizar por medio de varios sensores. El IMU es una opción muy generalizada, ya que se tiene acceso a los códigos para interpretar sus datos, y posee un precio relativamente bajo. Además de medir la orientación que posee el robot, también permite medir la velocidad angular en cualquier momento, lo cual sería muy útil para futuras aplicaciones. El IMU también cuenta con un microprocesador y gracias a ello se pueden obtener los datos por medio de una compuerta serial.

La brújula digital o magnetómetro es un elemento simple de utilizar, podría verse como el giroscopio del IMU. Al igual que el IMU, es un sensor barato. Sin embargo, se requiere realizar adaptaciones para obtener los datos por medio de un puerto serial o por medio de computador.

En cuanto al sensor de efecto Hall, este podría constituirse como la opción más precisa de las tres. El problema es que esta precisión viene acompañada de una mayor complejidad tanto para leer datos como para instarlo adecuadamente en el robot, ya que generalmente se utilizan para medir el ángulo de giro en el eje de un motor.

Al analizar las tres opciones, se puede ver que la más versátil y de fácil acceso es el IMU. Por ello, se selecciona para llevar a cabo la aplicación, ya que con un solo dispositivo se pueden medir tanto la orientación como la velocidad angular. Por otro lado, al contar con un microprocesador, el acceso a su información es más sencillo.

Para seguir una trayectoria se debe tener en cuenta los requerimientos anteriores, ya que dependiendo de los sensores con los que se cuenta, la implementación puede ser más compleja según el método. Por ejemplo, con la técnica del seguidor de línea no se cuenta con el sensor más adecuado para detectarla, ya que por medio del Kinect se requiere un mayor procesamiento computacional para hacerlo. Además se requerirían hacer modificaciones físicas al ambiente para llevarlo a cabo.

El mapa precargado con una ruta permitiría definir fácilmente el camino que debe seguir el robot. La desventaja de este método es que se requiere conocer muy bien el ambiente en el que se trabaja o tener un mapa de él y por otro lado también se limita la libertad que tiene el robot para desplazarse.

Utilizar "*waypoints*" sería una buena opción ya que en conjunto con el método de campos potenciales, se podría definir la trayectoria que se quiere que siga el robot y a la vez se minimizaría el problema de los mínimos locales. La combinación de estos dos métodos es una excelente simbiosis de las herramientas con las que se puede trabajar, ya que se satisface un requerimiento y se reduce la desventaja de una técnica seleccionada anteriormente (mínimos locales). Por esta razón, se decide utilizar la técnica de "*waypoints*".

El requerimiento número nueve hace referencia al movimiento del vehículo de forma autónoma. Para ello se presenta la opción de utilizar un computador portátil o utilizar un microcontrolador. El microcontrolador tiene la ventaja de que su tamaño es menor que el del computador portátil, haciéndolo más fácil de acoplar a un robot. También permite ser programado para una única tarea específica, lo que lo puede hacer más eficiente en el control.

Por otro lado, el computador portátil tiene la posibilidad de recibir información de una mayor cantidad de sensores, así como una mayor versatilidad a la hora de programar la

aplicación. También, el computador portátil cuenta con una batería integrada, por lo que no requiere ser alimentado por medio de cables o una configuración extra.

Al comparar las dos opciones se puede ver que una de ellas es más pequeña y fácil de acoplar al vehículo móvil, mientras que la otra es más grande pero posee mayor facilidad para procesar la información de los sensores. Si se utiliza el microcontrolador, se requeriría un mayor trabajo para interpretar la información de los sensores y muchos microcontroladores no cuentan con la cantidad de puertos seriales necesarios para esta tarea. Por otro lado, aunque el computador portátil tenga un mayor tamaño, este tiene más ventajas para procesamiento de información y comunicación con el robot, por lo que se decide trabajar con el computador portátil.

### 3.4.3. Especificaciones

1. La aplicación será diseñada para un robot móvil de tracción diferencial.
2. Para llegar a un punto específico determinado por el usuario se usará un GPS, así como para reconocer la ubicación del robot en el espacio.
3. El sensor de tecnología 3D que se usará como base para la navegación y para detectar obstáculos será un Kinect.
4. Para evitar colisiones y modificar la velocidad del vehículo en función de la distancia a la que se encuentre de la meta o los obstáculos se implementará la teoría de campos potenciales.
5. Para reconocer la orientación del vehículo en el espacio se utilizará un sensor IMU.
6. El método a utilizar para seguir una trayectoria definida por el usuario será el de "*waypoints*".
7. El accionamiento del vehículo de forma autónoma se realizará por medio de un computador portátil.

Con las especificaciones ya definidas, se puede proceder a explicar el proceso que se realizó para solucionar el problema en función de dichas especificaciones. Este proceso

se dividió en dos etapas: una de simulación y otra de implementación. En el siguiente capítulo se explica la primera de ellas.

# Capítulo 4: Simulación

El desarrollo de este proyecto lleva consigo una fuerte parte de investigación. Las herramientas que se utilizaron para simular tanto la aplicación, así como los sensores y distintos elementos del ambiente significaron una parte relevante durante la ejecución del proyecto.

Luego de aprender a utilizar la herramienta y a familiarizarse con ella, se procedió a estudiar la teoría que permitiera llevar a cabo un correcto funcionamiento de la aplicación y dar una solución al problema planteado. Una introducción a los conceptos y teoremas necesarios para la implementación se presentaron en el Capítulo 2.

En general, el proceso de la simulación se puede resumir en tres secciones. La primera de ellas fue más que todo para conocer el simulador y la forma en que trabajan los distintos elementos que tiene y a la vez desarrollar algoritmos para que el robot pueda navegar sin un objetivo, únicamente evitando obstáculos.

Posteriormente, en la segunda sección o etapa se hace la navegación inteligente. En otras palabras, se le da una meta al robot y a la vez se conserva la etapa anterior de evitar obstáculos. Por último, se combinan distintas técnicas de navegación para obtener la aplicación final.

El programa utilizado para la simulación es V-REP, el cual se ha venido utilizando bastante en el LRM debido a su versatilidad. El simulador cuenta con *scripts* o códigos base para cada modelo. Por ejemplo, para el Kinect, el programa ya cuenta con un código base de su funcionamiento. Sin embargo, si se desea que trabaje en conjunto con otros objetos u elementos o que realice otras funciones, éstas se deben programar.

La explicación de cómo se llevó a cabo cada una de las etapas se explica en las siguientes secciones.

## 4.1. Evitar obstáculos

En esta etapa se implementó un ambiente sencillo para mostrar el funcionamiento del robot junto con el sensor Kinect. La información procedente de este sensor influencia directamente la velocidad en cada uno de los motores del robot.

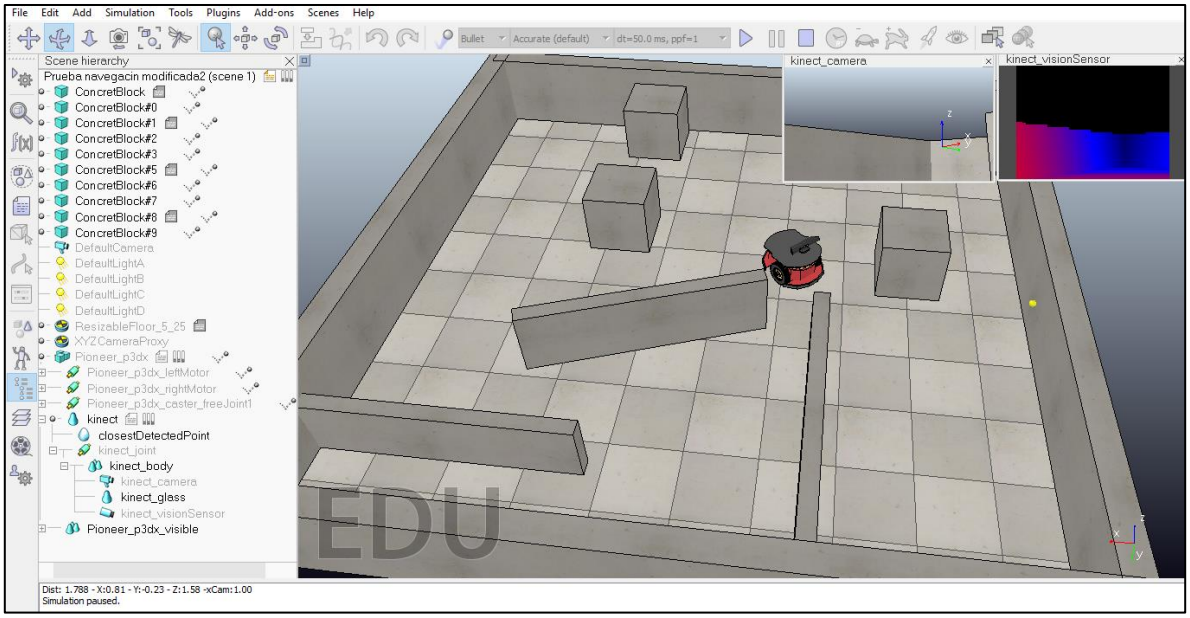



**Figura 14. Ambiente de simulación para probar evitar obstáculos. Elaborado por el autor en VRep.**

El ambiente consiste en un terreno plano de 5m x 5m, donde hay diferentes tipos de obstáculos como paredes y bloques de diversos tamaños. En este ambiente también se pueden ver dos ventanas, una en la que se muestra lo que ve la cámara RGB y otra ventana

en la que se muestra la profundidad en una escala de pseudo-color. Para esta escala, mientras más cálido sea un color, más cerca se encuentra esa zona del sensor. A continuación se explicará el funcionamiento del sensor principal y su interacción con el robot.

### 4.1.1. Uso del Kinect

#### 4.1.1.1. Características de simulación del Kinect

El fin de utilizar este sensor es analizar los datos del "*point cloud*" generado para detectar los obstáculos que se encuentran en el camino y con ello accionar el movimiento del robot. Para fines de la simulación, se trabaja con una resolución de la cámara de profundidad de 64x48 pixeles, aunque la resolución real del Kinect es de 640x480. El hecho de no utilizar la resolución real permite mejorar la velocidad de simulación y la variación en los resultados es mínima.

Para entender mejor las características de la resolución del Kinect, así como los ángulos de visión que posee, se puede observar la siguiente imagen:

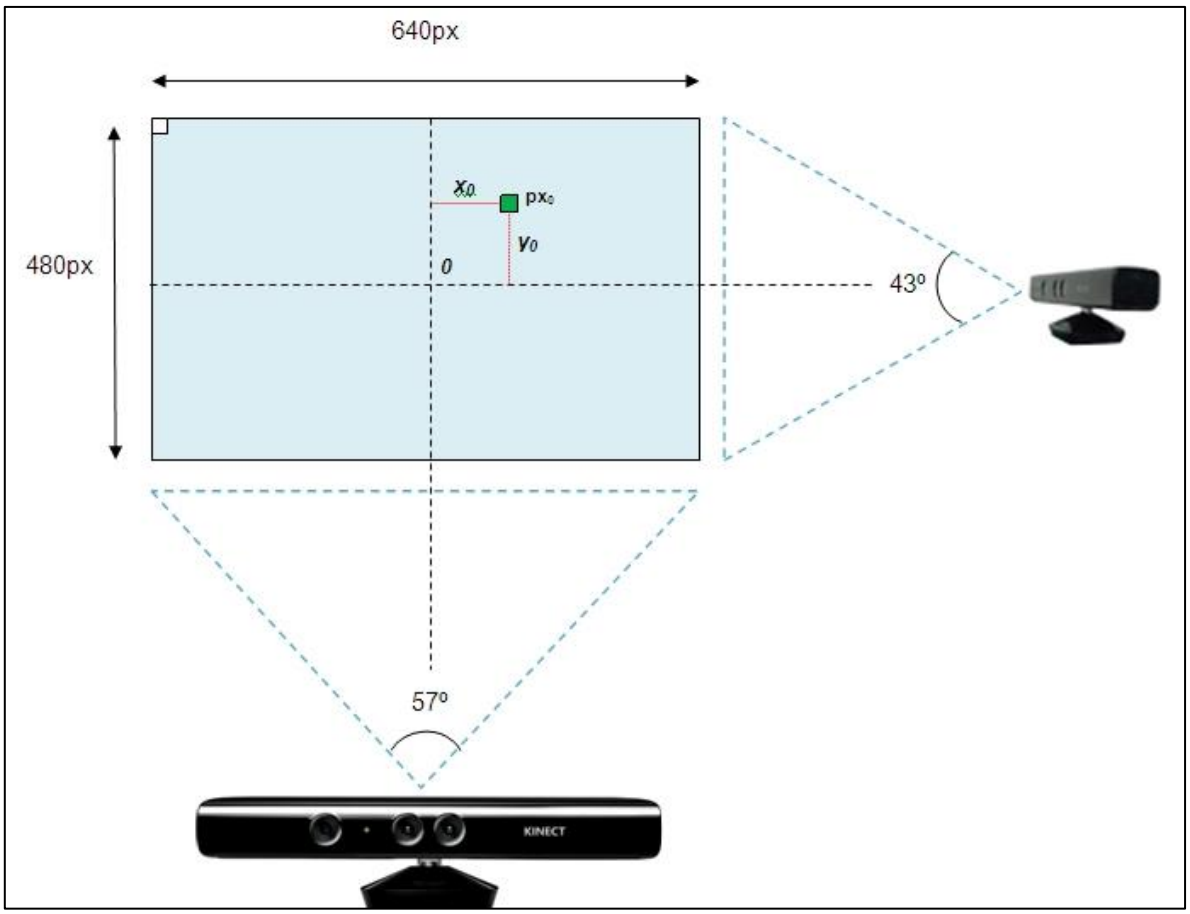



**Figura 15. Características del sensor Kinect. Recuperado de: (Grupo de Investigación en Robótica Autónoma, 2015)**

Según las características que posee el Kinect, los puntos del "*point cloud*" se pueden encontrar en una región delimitada por dos planos. El plano más cercano se conoce como "*Near Clipping Plane*", que es la distancia mínima que se puede medir y el segundo plano es el de la máxima profundidad que se puede medir.

Al observar la figura 15, se puede ver que al tener un punto en una posición $p_{x0}$, este tendrá una respectiva coordenada $x_0$ y una respectiva coordenada $y_0$ con respecto al marco de referencia del Kinect que se observa en la imagen, donde el cero u origen es en el centro del plano bidimensional.

#### 4.1.1.2. Algoritmo del Kinect

Con la información anterior se puede entender mejor el funcionamiento del sensor Kinect y con ello el algoritmo para detectar el punto más cercano del "*point cloud*". A continuación se presenta una tabla con las variables usadas en el algoritmo.

**Tabla 2. Variables utilizadas en el algoritmo de detección de punto más cercano en el "point cloud". Elaborado por el autor.**

| Variable | Significado |
|---|---|
| i | Contador para los pixeles en X |
| j | Contador para los pixeles en Y |
| xAngle | Ángulo entre el punto analizado y el eje X del Kinect |
| yAngle | Ángulo entre el punto analizado y el eje Y del Kinect |
| depthValue | Valor de profundidad para la coordenada i,j dentro del "*point cloud*" |
| zCoord | Coordenada en el eje Z del Kinect del punto analizado |
| xCoord | Coordenada en el eje X del Kinect del punto analizado |
| yCoord | Coordenada en el eje Y del Kinect del punto analizado |
| dist | Distancia entre el punto analizado y el Kinect |
| closestDist | Distancia más cercana entre todos los puntos del "*point cloud*" y el Kinect |
| closestXYZ | Vector que contiene las coordenadas X,Y,Z del punto más cercano |

El algoritmo inició con la definición de constantes que se requieren para calcular algunas variables de la tabla anterior. Estas constantes se definieron anteriormente y son los ángulos del sensor en el eje X y el eje Y, el "*NearClippingPlane*", la profundidad máxima que se puede medir y la resolución en cada eje.

Una vez definido esto, se inicia el análisis del "*point cloud*" para detectar cuál de todos los puntos que lo conforman se encuentra más cercano al sensor, así como calcular cuál es la distancia a la que está y las coordenadas en las que se ubica. Este proceso se representa en la figura 16 por medio de un diagrama de flujo. Las variables que se utilizan son las mismas de la tabla 2.

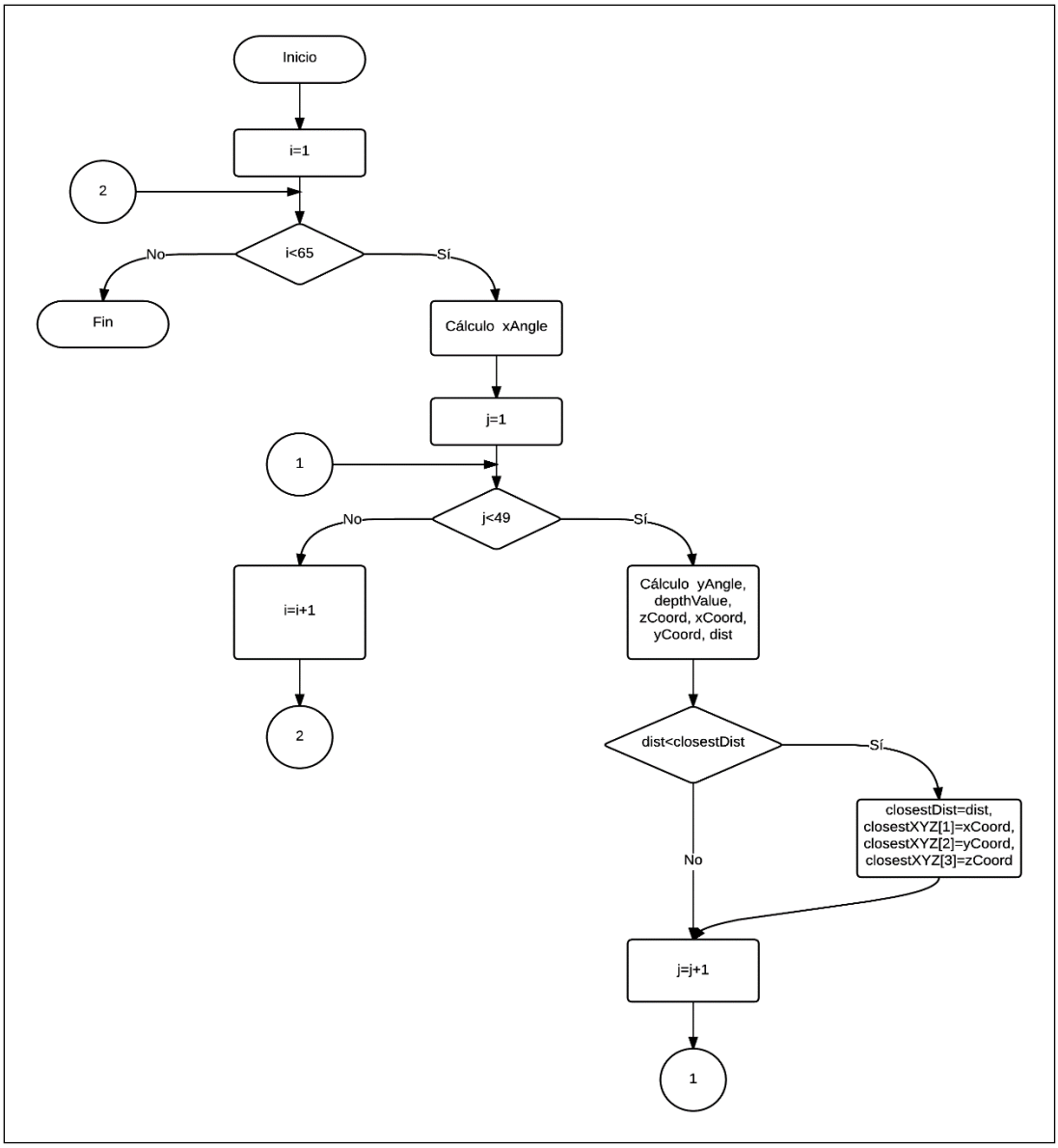



**Figura 16. Diagrama de flujo para lectura del "point cloud" para encontrar el punto más cercano al Kinect. Elaborado por el autor en LucidChart.**

El algoritmo para detectar el punto más cercano se inicia con una sentencia "*for*" que inicia en uno y acaba en sesenta y cuatro. Se hace esto para recorrer los 64 pixeles de resolución del eje X del sensor. Posteriormente, se calcula el *xAngle* para el respectivo punto i, para saber el ángulo existente con respecto al eje X del sensor. Este ángulo viene dado por la siguiente fórmula:

$$xAngle = ((32 - i - 0{,}5)/32)) * camXHalfAngle \qquad (4.1)$$

De la fórmula anterior se puede observar que para obtener el ángulo con respecto al eje X en el que se encuentra el punto analizado, se emplea una regla de tres. Primero, se obtiene la distancia a la que se encuentra ese punto con respecto al centro del eje (32-i). Luego, esa distancia se mide con respecto al centro del pixel, por ello es que al resultado anterior se le resta 0,5. Una vez que se tiene esta distancia, se multiplica por la mitad del ángulo de visión del eje X y se divide entre la distancia en pixeles que le corresponde a este ángulo, que es 32.

El resultado de esta fórmula genera que para todos aquellos pixeles menores a 32, se obtengan valores de ángulos positivos, mientras que para pixeles mayores a 32, los ángulos resultantes son negativos.

Una vez que se realiza esto, se establece una segunda sentencia "*for*", la cual tiene como objetivo analizar los pixeles del eje Y. Es por ello que el ciclo inicia 1 y acaba en 48. Dentro de este segundo ciclo, se calcula *yAngle* para el respectivo punto j, el cual corresponde al ángulo con respecto al eje Y del Kinect. Este ángulo viene dado por la siguiente fórmula:

$$yAngle = ((j - 24 + 0{,}5)/24)) * camYHalfAngle \qquad (4.2)$$

El cálculo de este ángulo responde a la misma lógica de *xAngle*, se aplica una regla de tres. Para una respectiva posición desde el centro del pixel analizado hasta el eje Y del Kinect, se calcula el ángulo correspondiente en función de la mitad del ángulo de visión en el eje Y, y la respectiva longitud de pixeles asociada a él.

El resultado de la fórmula genera que para todos aquellos pixeles menores a 24 se obtengan valores de ángulos negativos, mientras que para pixeles mayores a 24, los ángulos resultantes son positivos.

Dentro de este ciclo también es necesario calcular el *depthValue*, el cual corresponde a un valor entre 0 y 1 que representa la profundidad de la coordenada i,j dentro del "*point cloud*", el cual es un valor proporcional a la profundidad real. Dichos valores de profundidad de cada pixel se encuentran guardados en un vector de 3072 elementos (64x48), llamado *depthBuffer*. Lo cual viene descrito por la siguiente fórmula:

$$depthValue = depthBuffer[i + (j - 1) * 64] \quad (4.3)$$

Luego, la variable *depthValue* se multiplica por la constante de la máxima distancia que se puede medir y se le suma el "*NearClippingPlane*" para obtener el valor de la coordenada en el eje Z del punto i,j.

Una vez que se tiene la coordenada en Z, se puede obtener por medio de trigonometría las coordenadas en X y en Y, ya que anteriormente se habían calculado las variables *yAngle* y *xAngle*. Dichos cálculos se pueden ver a continuación:

$$xCoord = math.tan(xAngle) * zCoord \quad (4.4)$$

$$yCoord = math.tan(yAngle) * zCoord \quad (4.5)$$

Finalmente, se obtiene la distancia euclidiana del punto i,j del "*point cloud*" con las coordenadas tridemensionales *xCoord, yCoord* y *zCoord* y se almacena en la variable *dist*.

El siguiente paso es determinar sila variable *dist* es la distancia más cercana al sensor del Kinect (*closestDist*). Para ello, se realiza una simple comparación. En caso de que *dist* sea menor que el valor anterior guardado en *closestDist*, se actualiza la distancia más cercana y en el vector *closestXYZ* se guardan las coordenadas tridimensionales de dicho punto.

Al acabar de analizar todos los puntos i,j del "*point cloud*", los valores guardados en *closestXYZ* se almacenan en variables globales con el fin de que se puedan leer desde otros *scripts*, como el del robot al que está conectado el Kinect. Como una ayuda visual, el código también grafica una esfera amarilla en el ambiente de simulación, en las coordenadas tridimensionales donde se encuentra el punto más cercano al sensor.

#### 4.1.1.3. Modificaciones al código

Realmente al código no se le debieron realizar muchas modificaciones para que el robot tuviera acceso a los datos del sensor. Una de las primeras modificaciones que se hizo fue declarar las variables de las coordenadas tridimensionales como variables globales. Como se había mencionado anteriormente, esta pequeña modificación permite leer los datos desde cualquier otro script del entorno de simulación.

Una modificación inicial que se realizó para entender mejor el comportamiento del "*point cloud*", fue leer únicamente un espacio bidimensional en vez de leer un espacio tridimensional. En otras palabras, se analiza una línea horizontal y la respectiva profundidad para cada uno de los puntos de la línea.

Para hacer esto, el cambio que se realizó fue establecer un j fijo, lo que haría que ya no se recorran los 48 pixeles del eje Y, únicamente los 64 pixeles del eje X para esa posición j. Esto hace que variables como *yAngle* y *yCoord* se tornen constantes y que el cálculo de *depthValue* dependa de una única variable i.

Analizar solo una línea permite entender mejor las coordenadas de referencia del Kinect, así como la forma en la que se interpreta la información del "*point cloud*". Permite entender mejor los cálculos necesarios para triangular la posición específica en la que se desea encontrar el punto más cercano al sensor. Una vez realizado esto, es mucho más fácil trabajar con el espacio tridimensional conformado por los distintos puntos.

Por último, el código se modificó para que guardara la posición i (1-64) en la que se encuentra el punto más cercano, en una variable global. Esto se hizo para facilitar el desarrollo del algoritmo para evitar obstáculos. Lo cual se explicará en la siguiente sección.

### 4.1.2. Pioneer P3-DX

#### 4.1.2.1. Características de simulación del Pioneer P3-DX

El Pioneer P3-DX es el robot utilizado para simular la aplicación de navegación en el simulador. Este es uno de los robots más populares para investigación en robótica debido

a su versatilidad. El robot cuenta con un controlador de movimiento con retroalimentación de *encoders*.

Este es un robot móvil diferencial que cuenta con ocho sonares en la parte delantera y ocho en la parte trasera. Sin embargo, no se hizo uso de ellos porque todas las mediciones para detectar obstáculos se hicieron en base al Kinect.

El Pioneer cuenta únicamente con dos ruedas motorizadas y una tercera que sirve como punto de apoyo. De aquí que su movimiento sea diferencial, lo que significa que si se desea girar, una rueda debe girar más rápido que la otra. La velocidad máxima de este robot es de 1,2 m/s y tiene un radio de giro de 26,7 cm. (Adept MobileRobots, 2015)

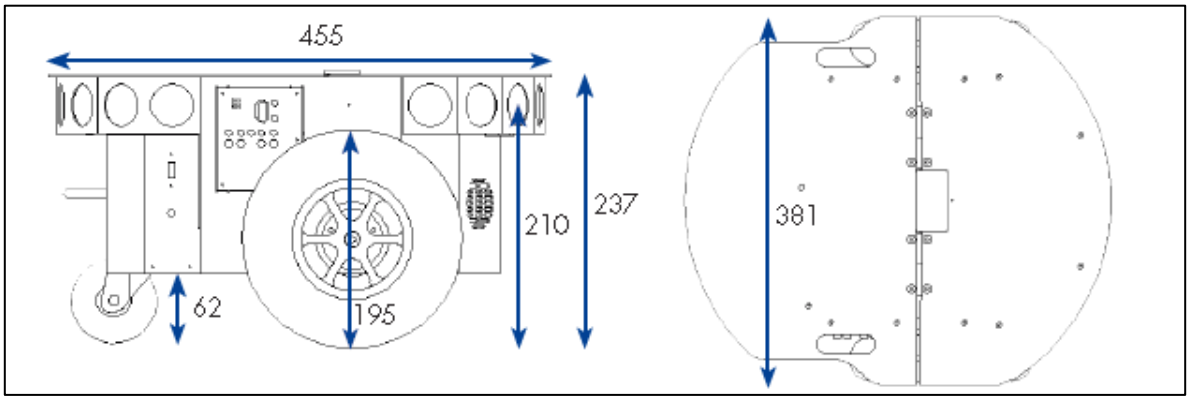



**Figura 17. Dimensiones del Pioneer P3-DX. Recuperado de: (Adept MobileRobots, 2015)**

En la figura anterior se muestran las dimensiones del robot móvil, lo cual es muy importante tener en cuenta ya que con esto podemos saber el espacio mínimo por el que puede navegar el robot. Por ejemplo, para un movimiento en línea recta se requiere al menos un espacio de 40 cm de ancho y en caso de que el robot gire se debe tomar en cuenta el radio de giro que tiene.

Para efectos de la simulación, como se mencionó anteriormente, aunque el robot cuenta con 16 sensores ultrasónicos para ayudarlo en la navegación, no se hizo uso de ellos ya que uno de los objetivos de este proyecto es investigar los "*points clouds*" generados por el Kinect y las aplicaciones que pueden tener en el área de la robótica móvil.

Aspectos como la velocidad máxima se establecieron igual en el simulador para que fuera lo más similar a la realidad. Por otro lado, el ambiente en el que navega el robot presenta diferentes configuraciones para que el robot pase por pasillos pequeños, tome decisiones en esquinas y determine si puede pasar por ciertos espacios.

Cabe resaltar que la selección del robot Pioneer P3-DX no es pura casualidad, sino que se hace con el fin de desarrollar una aplicación para un robot Pioneer con el que cuenta el *Laboratório de Robótica Móvel*.

#### 4.1.2.2. Algoritmo desarrollado para que el robot evite obstáculos

Para desarrollar este algoritmo, se utilizó como base la teoría de Braitenberg expuesta con anterioridad en el capítulo 2. Basado en la configuración del vehículo 2a, se desarrolló un comportamiento para alejarse del estímulo, es decir, alejarse del punto más cercano que se detecta en el "*point cloud*".

Dicho esto y con la información de la sección anterior, se espera que se facilite la comprensión del algoritmo para que el robot móvil evite obstáculos. De forma general, el algoritmo lo que hace es modificar la velocidad de los motores de las ruedas del robot en función de la distancia a la que se encuentra del obstáculo más cercano detectado. Se presenta en la tabla 3 las variables utilizadas para desarrollarlo.

**Tabla 3. Variables utilizadas en el algoritmo de control de movimiento del Pioneer P3-DX. Elaborado por el autor.**

| Variable | Significado |
|---|---|
| TCx | Coordenada en el eje X del punto más cercano al sensor |
| TCy | Coordenada en el eje Y del punto más cercano al sensor |
| TCz | Coordenada en el eje Z del punto más cercano al sensor |

| | |
|---|---|
| TDst | Distancia total a la que se encuentra el punto más cercano |
| PosX | Pixel del eje X en el que se encontró el punto más cercano |
| v0 | Velocidad máxima de los motores del robot |
| pOld | Valor anterior de PosX |
| vLeft | Velocidad que se asignará al motor izquierdo |
| vRight | Velocidad que se asignará al motor derecho |

El algoritmo inicia definiendo tres variables en las que se guardaran los valores globales de las coordenadas X, Y, Z, del punto más cercano al robot que se habían definido como valores globales desde el código del Kinect. También, se obtienen los valores de la distancia total a la que se encuentra ese punto y el número de pixel en el eje X del Kinect en el que se encontró.

Una vez que se guardan estos valores, se hace una primera comparación para determinar si el obstáculo detectado se encuentra a una distancia igual o menor que 0,5 metros. En caso de que sí se encuentre a esta distancia, el robot se detiene y para ello se asigna una velocidad cero a cada uno de los motores del robot.

Cuando el robot se encuentra detenido, se realiza otra comparación para determinar hacia dónde debe doblar el robot. Esto se hizo con la intención de que, cuando el robot tome la decisión de doblar en una dirección y se encuentra a una distancia menor de 0,5 metros, no sea de una forma aleatoria o hacia una dirección predilecta siempre.

En esta etapa es donde el número de pixel del eje X del Kinect que se había guardado en el script anterior se torna relevante, ya que con él se toman decisiones para el accionamiento de los motores. Lo que se hace es comparar el valor anterior del pixel con el valor actual. Entonces, si el valor actual es mayor que el anterior, o en otras palabras este valor aumenta, es porque el robot debe girar a la derecha, caso contrario es porque debe girar a la izquierda.

Como se había mencionado anteriormente, el eje X del Kinect tiene valores entre 1 y 64, por lo que los valores de los pixeles siempre estarán en ese rango y es más fácil tomar las decisiones con este método. Este método se decidió implementar con el fin de tomar una mejor decisión de giro cuando el robot se encuentra en esquinas.

Cuando el robot decide hacia qué lado girar, se accionan los motores con la misma magnitud de velocidad pero con sentidos opuestos, con el fin de que pueda girar sobre su propio eje y así evite colisiones contra algún objeto.

Una vez que se termina de asignar la velocidad de cada motor, se actualiza el valor de la variable *pOld* para que en la siguiente iteración del código se tenga el valor antiguo de la posición. Finalmente, se procede a accionar los motores con la velocidad que se asignó a cada uno.

Retomando la comparación de si el obstáculo se encuentra a una distancia menor a 0,5 metros, está el caso en el que esta condición no se cumple y ahora el robot puede encontrarse a un intervalo de entre 0,5 y 0,75 metros del objeto o a una distancia mayor a 0,75 metros.

Para el caso en el que es mayor a 0,75 metros, a la velocidad de cada motor se le asigna un mismo valor, el cual corresponde a la máxima velocidad que puede alcanzar el robot que es de 1,2 m/s. Luego de asignarse, se procede al accionamiento de los motores.

Cuando la distancia a la que se encuentra el Pioneer de los obstáculos tiene un valor en el intervalo entre 0,5 y 0,75 metros, el robot toma la decisión de girar a la izquierda o a la derecha. Es aquí donde la variable *PosX* se vuelve a tornar importante, ya que si el obstáculo se encuentra a la derecha del sensor, esta variable tendrá un valor menor a 32 y el robot girará a la izquierda, caso que sea mayor, el obstáculo se encontrará a la izquierda del sensor y el robot girará en el sentido contrario. Cuando el obstáculo se encuentra totalmente al frente del sensor, es decir en la posición de pixel 32, este girará a la izquierda.

Para este intervalo, la velocidad de las ruedas viene dada por la siguiente fórmula:

$$vel = 2{,}4(TDst) - 0{,}6 \tag{4.6}$$

La fórmula anterior es una función lineal de la velocidad donde su valor máximo alcanza una velocidad de 1,2 m/s y en su valor mínimo 0,6 m/s. Dicha función da una velocidad proporcional a la distancia que se ubica el sensor del obstáculo.

Por lo tanto, si el robot debe girar a la izquierda, al motor izquierdo se le asigna esa función y el motor derecho se mantiene con la velocidad máxima. En el caso de que deba

girar a la derecha, el motor que se mantiene con la velocidad máxima es el izquierdo y al que se le asigna la función es el derecho.

El resumen de las velocidades para cada intervalo se presenta en la siguiente tabla:

**Tabla 4. Velocidad en cada motor en función de la distancia a la que se encuentra el objeto más cercano. Elaborado por el autor.**

| | Motor | Velocidad para girar a la izquierda (m/s) | Velocidad para girar a la derecha (m/s) |
|---|---|---|---|
| **TDst≤0,5** | Izquierdo | -0,6 | 0,6 |
| | Derecho | 0,6 | -0,6 |
| **0,5<TDst≤0,75** | Izquierdo | 2,4(TDst)-0,6 | 1,2 |
| | Derecho | 1,2 | 2,4(TDst)-0,6 |
| **0,75<TDst** | Izquierdo | 1,2 | 1,2 |
| | Derecho | 1,2 | 1,2 |

Para facilitar la comprensión del algoritmo, en la siguiente imagen se presenta el diagrama de flujo correspondiente:

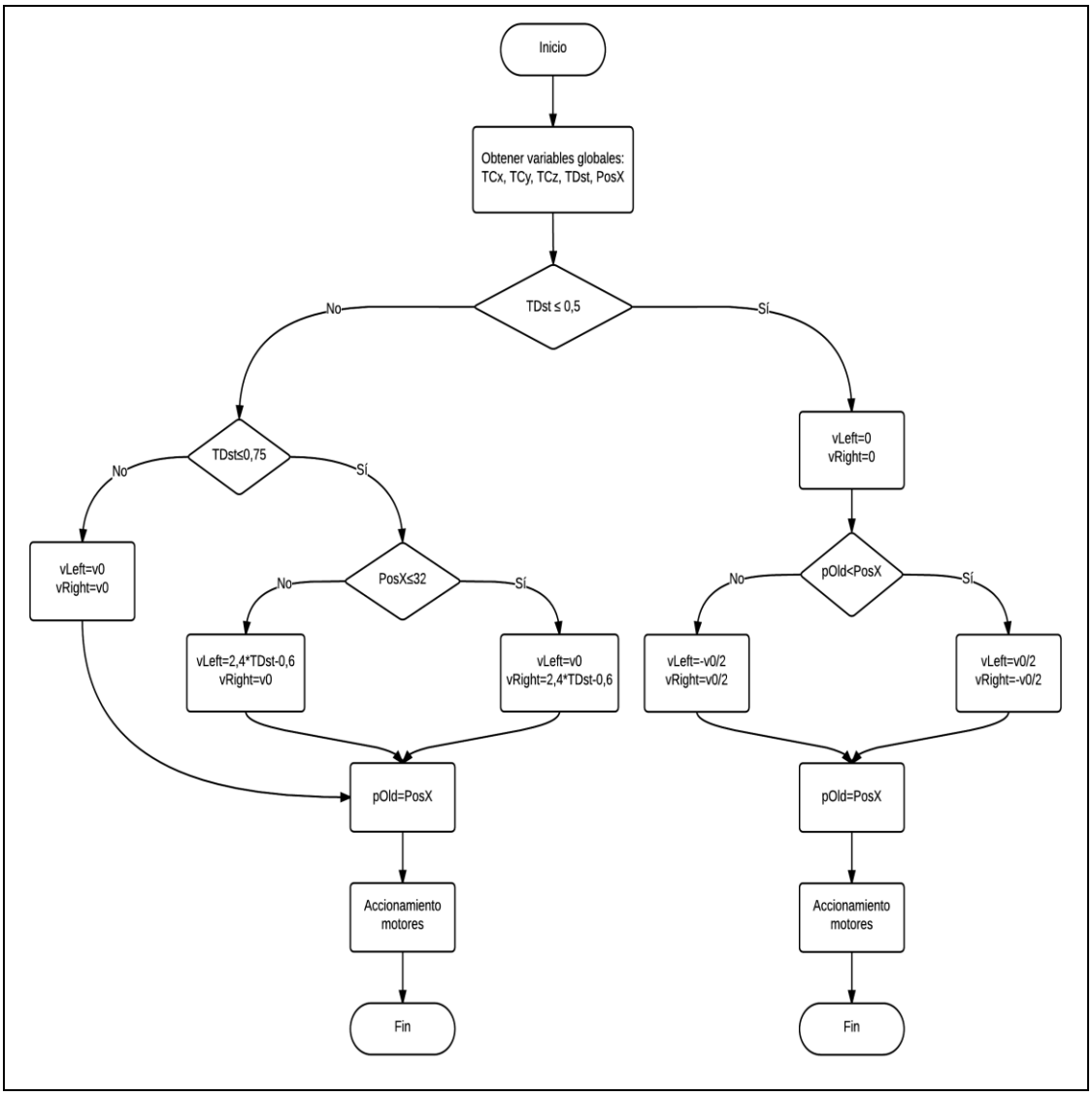



**Figura 18. Diagrama de flujo del algoritmo de control de movimiento del Pioneer P3-DX. Elaborado por el autor en LucidChart.**

Cabe resaltar que los algoritmos que se presentaron anteriormente, como los que se presentarán en los próximos apartados se ejecutan de forma cíclica por el simulador. Es por ello que no se especifica esa condición en los diagramas de flujo.

Descrito el procedimiento para evitar obstáculos, se puede continuar hacia la siguiente etapa de la simulación, la navegación inteligente. En la siguiente sección se puede observar el impacto del trabajo descrito anteriormente y cómo estas técnicas influenciaron el desarrollo de los nuevos algoritmos y la combinación del uso de sensores.

## 4.2. Definir una trayectoria

Una vez que ya se tiene el conocimiento del funcionamiento del robot, el entorno de simulación y un primer método para evitar obstáculos, se puede proceder a mejorar la aplicación para que el robot realice propiamente la navegación autónoma.

Para llevar esto a cabo, se utilizó la teoría de campos potenciales con el fin de definir una trayectoria hasta el punto que se desea alcanzar y que a la vez se puedan evitar obstáculos. Como se había mencionado en el Capítulo 3, esta técnica requiere combinarse con otros métodos debido a la presencia de mínimos locales. El proceso para disminuir este efecto se explicará en las próximas secciones.

Por otro lado, para implementar esta técnica también se requiere utilizar otros sensores aparte del Kinect, los cuales permiten determinar la ubicación del robot en el espacio así como la orientación que este posee. En las siguientes subsecciones se explica el funcionamiento de los nuevos sensores que se emplearon, así como de la teoría aplicada para lograr el objetivo del proyecto.

### 4.2.1. Giroscopio

Del análisis de opciones para determinar la orientación del robot se concluyó que el mejor sensor para esta tarea es el giroscopio. Otra ventaja es que en el simulador se encuentra un modelo de este sensor.

El *script* de este sensor está determinado para brindar mediciones de la velocidad angular del objeto al que se encuentre asociado, que para este caso es el robot Pioneer P3-DX. Sin embargo, la medición que se requiere es la de la posición angular, no la de la velocidad. Por esta razón, se debió profundizar en el código del sensor para modificar los datos que se obtienen, y que se pueda medir la variable deseada.

El código del sensor se inicializa obteniendo la matriz de transformación del sensor con respecto al marco de referencia universal y el tiempo de simulación. Esta matriz se obtiene por medio de una función específica (*simGetObjectMatrix*), la cual recibe como

parámetros el objeto del cual se quiere obtener la matriz de transformación, que es el sensor; y el marco de referencia con respecto al cual se quiere obtener la matriz, que es el universal. El resultado de esta función da una matriz de 4x3, donde la primera fila son los componentes de la orientación del eje X, la segunda fila corresponde a los componentes de la orientación del eje Y, la tercera a los componentes de la orientación del eje Z y la última fila corresponde al componente de traslación.

Una vez que se inicializa el código, se obtiene una matriz de transformación actualizada. También se calcula la matriz inversa de la primera matriz de transformación que se obtuvo. Estas dos matrices se multiplican y a partir del resultado se utiliza la función *simGetEulerAnglesFromMatrix* para obtener los ángulos de Euler que describen la orientación del objeto con respecto al marco universal.

Originalmente, el código toma estos ángulos y los divide entre un delta de tiempo para calcular la velocidad angular. Esta parte del código fue la que se modificó, ya que esos cálculos estarían sobrando y con anterioridad ya se habían obtenido los ángulos de Euler, los cuales corresponden a la orientación del robot en el espacio.

Posteriormente, las variables de los ángulos se definen como variables globales para que sus valores puedan ser leídos por otros *scripts*. Por último, se actualizan los valores de las matrices de transformación y las variables del delta de tiempo.

Un aspecto importante de esta parte, es que el único ángulo de importancia para el funcionamiento del proyecto, es aquel alrededor del eje Z, el cual se conoce como *Yaw*. Esto es debido a que se supone que el robot trabaja en un plano bidimensional y por otro lado la orientación de los demás ejes no afectan los cálculos que se realizarán posteriormente en otros algoritmos.

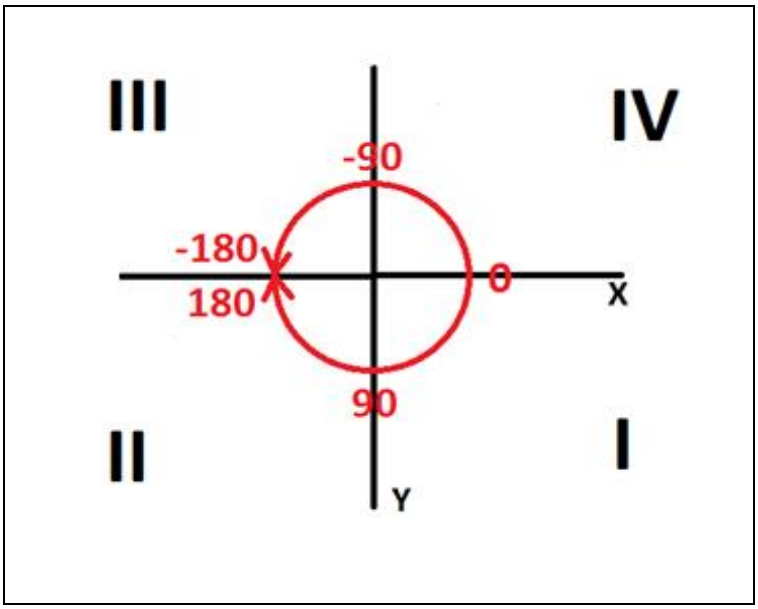



**Figura 19. En rojo, ejemplo de las mediciones del giroscopio, donde los ángulos son valores positivos o negativos en 0 y 180°. Recuperado de (28IM, 2015)**

En la figura 19, se pueden observar en color rojo los valores de ángulos que mide el giroscopio. Generalmente los ángulos se suelen medir de 0 a 360°. Sin embargo, este sensor mide el ángulo existente hasta un valor máximo de 180° con respecto al eje X que tiene predeterminado, que para nuestro caso sería el eje X del marco de referencia global. Por lo tanto, la orientación del robot será expresada en un ángulo positivo o negativo de 0 y 180° con respecto al eje X. Este comportamiento es igual al resultado que daría la función *atan2* de dos argumentos X, Y. Los ángulos medidos por el sensor se encuentran en radianes.

### 4.2.2. GPS

El GPS es el segundo sensor que se debió utilizar para implementar la teoría de los campos potenciales. Este se coloca en el robot para obtener las coordenadas en las que se ubica en el ambiente de simulación. También se utiliza en algunas simulaciones como punto de referencia al que debe llegar el robot en su navegación autónoma.

El *script* del GPS no es muy complicado pero siempre es importante entender su funcionamiento a nivel de simulación, con el fin de saber si el modelo se ajusta al

comportamiento real que puede tener un GPS. Uno de los aspectos que hay que destacar de este sensor es que se simula con un error asociado a la señal, lo cual es de gran importancia porque la mayoría de los receptores GPS presentan esta característica. De no tomarse en cuenta esto, los resultados esperados podrían verse afectados considerablemente. Más adelante se explicará el método utilizado para minimizar este efecto.

Para obtener la posición absoluta, es decir con respecto al marco de referencia global, en el *script* del GPS únicamente se precisa de una función (*simGetObjectPosition*), la cual devuelve las coordenadas X, Y, Z de donde se encuentre el sensor. Para fines de la simulación, únicamente se requiere de las coordenadas X,y Y, ya que como se mencionó anteriormente, se supone que el robot se desplaza en un plano bidimensional.

Si el *script* se mantuviera así, el funcionamiento del GPS no sería como el de la gran mayoría de modelos existentes en el mercado de bajo precio, sino que sería como el de los modelos de extrema precisión cuyo precio es de miles de dólares. Por ello, se trabaja con el error que tiene, ya que en la etapa de implementación se trabajará con un modelo comercial de bajo costo.

Por lo tanto, antes de declarar las coordenadas de la ubicación como variables globales, a los valores absolutos iniciales se les agrega un error de valor aleatorio que puede ser tanto positivo como negativo. El error de valor aleatorio puede tener un valor máximo de 5 metros.

### 4.2.3. Campos Potenciales

Una vez que se ha explicado el funcionamiento de todos los sensores que se requieren para la navegación autónoma del robot, se procede a explicar la implementación de la teoría de campos potenciales. Esta permite que la navegación se efectúe sin colisiones, llegando hasta un punto deseado. A la vez, esta teoría permite controlar la velocidad y orientación del robot en función de la distancia a la que se encuentra de la meta o los obstáculos.

Como se mencionó anteriormente, la teoría de campos potenciales se aplicó para un espacio bidimensional. Esto se debe a que la coordenada de la altura a la que se encuentra el robot no influye en los cálculos de los campos imaginarios y la navegación que realiza el robot se puede decir que es en un espacio 2D. Los campos resultantes actúan en el plano bidimensional XY.

En la sección 4.1 se explicó el desarrollo del algoritmo básico para que el robot evite obstáculos. A partir de la idea utilizada en ese algoritmo, en la cual la velocidad se ve influenciada por la distancia a la que se encuentra el robot de los objetos, se aplicó el mismo concepto para variar la influencia del campo potencial de atracción en relación a la distancia a la que se encuentra el robot de la meta.

De acuerdo a lo expuesto anteriormente y al acercamiento de campos potenciales expuesto por Goodrich (2008), donde se define el gradiente del campo potencial en función del vector de posición de la partícula., se obtienen las funciones del gradiente del campo de atracción en función de distintos radios de influencia:

$$\Delta x = \begin{cases} 0, \ d < r \\ \alpha(d-r)\cos\theta \, , r \leq d \leq (s+r) \\ \alpha * \mathrm{s} * \cos\theta, \ (s+r) < d \end{cases} \tag{4.7}$$

$$\Delta y = \begin{cases} 0, \ d < r \\ \alpha(d-r)\,\mathrm{sen}\,\theta \, , r \leq d \leq (s+r) \\ \alpha * s * \mathrm{sen}\,\theta, \ (s+r) < d \end{cases} \tag{4.8}$$

Las fórmulas anteriores dan como resultado los componentes X, Y, del gradiente del campo de atracción. Donde *d* es la distancia euclidiana entre el robot y la meta. La variable *r* es la distancia, con respecto a la meta, a la cual se desea que la influencia del campo potencial sea nula o deje de actuar. Se puede ver esta variable como la distancia de la meta a la que se desea que el robot se detenga. Por lo tanto, si la distancia entre el robot y la meta es menor al radio definido anteriormente, los componentes del gradiente serán cero y el robot se detendrá.

La variable *s* es la distancia sobre la cual se extiende el campo de atracción. Por lo tanto si el robot se encuentra a una distancia de la meta definida entre *r* y *(s+r)*, el campo de atracción tendrá un valor variable, donde su valor mínimo se alcanza en *r* y su valor

máximo en *(s+r)*. La variable $\alpha$ es una constante con la cual se puede ajustar la magnitud del campo. La variable $\theta$ es el ángulo existente entre la posición del robot y la posición de la meta, con respecto al marco universal. Por lo tanto, el valor del gradiente del campo potencial para ese intervalo, estará definido por el seno o el coseno de $\theta$, multiplicado por la constante $\alpha$ y el factor *(d-r)* que hace que el valor del campo varíe entre su valor máximo y mínimo.

Para el intervalo donde la distancia *d* es mayor a *(s+r)*, a los componentes del gradiente se les asigna el máximo valor que puede tener el campo potencial que estará definido por el seno o el coseno de $\theta$, las constantes $\alpha$ y *s*.

Basado en la misma idea para obtener los campos potenciales de atracción, se obtienen los de repulsión que están descritos por las siguientes fórmulas:

$$\Delta x = \begin{cases} -sign(\cos\theta)\gamma, \ d < r \\ -\beta(s + r - d)\cos\theta\,, r \leq d \leq (s + r) \\ 0, \ (s + r) < d \end{cases} \tag{4.9}$$

$$\Delta y = \begin{cases} -sign(\operatorname{sen}\theta)\gamma, \ d < r \\ -\beta(s + r - d)\operatorname{sen}\theta\,, r \leq d \leq (s + r) \\ 0, \ (s + r) < d \end{cases} \tag{4.10}$$

Se puede observar que para este caso se mantienen las mismas condiciones de los intervalos. La diferencia es que ahora la variable *d* denota la distancia euclidiana entre el robot y el obstáculo, *r* denota la distancia dentro de la cual se presenta la mayor influencia del campo de repulsión y la variable *s* ahora representa la distancia sobre la cual se extiende el campo de repulsión. Se puede observar que se tienen dos nuevas variables $\beta$ y $\gamma$. La primera viene a cumplir la función que anteriormente cumplía $\alpha$ y la segunda es una constante para definir el máximo valor que se desea que posea el gradiente.

Para distancias menores que *r*, el robot experimentará la mayor repulsión posible. El gradiente tendrá un valor definido, y su signo será determinado por la función *sign( )*, esto con el fin de que apunte hacia afuera del obstáculo. Cuando el robot se encuentra entre *r* y *(s+r)*, experimentará los efectos de un campo de repulsión variable, donde alcanza su valor mínimo en *(s+r)* y su valor máximo para el intervalo en *r*.

Una vez explicadas las fórmulas que se implementaran en la simulación, se procede a explicar cómo se seleccionaron las distintas variables descritas anteriormente, con el fin de entrar en detalle del funcionamiento del algoritmo.

Para el campo potencial de atracción, se deben seleccionar los valores de $s$ y $r$, básicamente a criterio del usuario, considerando las distancias a las que se quiere que la influencia del campo sea menor y la distancia a la que se desea que el robot se detenga de la meta. Por lo tanto, se seleccionó para $s$ un valor de 1 metro y para $r$ un valor de 0,5 metros, ya que con esto el robot comenzaría a disminuir su velocidad a partir de 1,5 metros del objetivo y así no sería tan abrupto el frenado cuando se acerque a la zona de velocidad cero que es en 0,5 metros.

Por otro lado, para la variable $\alpha$, su valor se determina de forma totalmente empírica. Luego de probar varios escenarios se concluyó que el valor que mejor se ajusta es 1, ya que con esto el gradiente final tendrá un valor proporcional a la velocidad con la que debe navegar el robot. En la próxima sección se afondará en este detalle.

Para el campo potencial de repulsión pasó lo mismo con las variables $s$ y $r$. Por lo tanto se escogió un valor 0,5 metros para ambos. Esto hace que el campo de influencia de los obstáculos tenga un radio de alcance de un metro. Por lo tanto, si el robot sobrepasa esta distancia, comenzará a desviar su trayectoria de él.

El caso de las variables $\beta$ y $\gamma$, es similar al de $\alpha$. Su valor se determina totalmente de forma empírica. Para $\beta$ se seleccionó un valor de 1, ya que basado en la misma premisa de $\alpha$ se obtuvieron muy buenos resultados, donde de igual forma el gradiente tendrá un valor proporcional a la velocidad de navegación del robot. Finalmente para $\gamma$, se determinó que el valor que daba mejores resultados es 0,7 metros. Valores mayores provocaban que la repulsión fuera excesiva y valores menores provocaban que su efecto no fuera el deseado.

A modo de resumen se presenta la siguiente tabla con los valores de cada una de las variables para los campos de repulsión y atracción.

**Tabla 5. Valores de las variables usadas en las fórmulas de campos potenciales.**

| Variable | Campo de atracción | Campo de repulsión |
|---|---|---|
| s | 1 | 0,5 |
| r | 0,5 | 0,5 |
| α | 1 | - |
| γ | - | 0,7 |
| β | - | 1 |

Con toda esta información se hace más fácil comprender el algoritmo para calcular los campos potenciales que actúan sobre el robot. A continuación se describe el algoritmo para calcular el campo potencial de atracción entre el robot y la meta. Primero se muestra la tabla con las diferentes variables que se utilizaron.

**Tabla 6. Variables utilizadas para el cálculo del campo potencial de atracción entre el robot y la meta. Elaborado por el autor.**

| Variable | Significado |
|---|---|
| px0,px1,px2 | Variables que poseen el valor actual y dos anteriores de la coordenada en X dada por el GPS |
| py0,py1,py2 | Variables que poseen el valor actual y dos anteriores de la coordenada en Y dada por el GPS |
| robotX | Valor corregido de la ubicación del robot para la coordenada X |
| robotY | Valor corregido de la ubicación del robot para la coordenada Y |
| goalX | Coordenada X del punto al que debe llegar el robot |
| goalY | Coordenada Y del punto al que debe llegar el robot |
| dGoalR | Distancia entre el robot y el punto meta |
| angleGR | Valor anterior de PosX |
| deltaX | Componente en X de la fuerza resultante de atracción |
| deltaY | Componente en Y de la fuerza resultante de atracción |

El código se inicia obteniendo los valores de las coordenadas X, Y, de la ubicación del robot en el ambiente, que son dadas por GPS. En la sección en la cual que se describió

el funcionamiento de este sensor, se explicó que para realizar una simulación más exacta se trabaja con un error en la señal. Al inicio del proyecto, estos datos se trabajaron sin ninguna corrección, lo que hacía que la trayectoria del robot no fuera la mejor o que no lograra ubicar la posición de la meta correctamente.

Conforme se fue avanzando con el desarrollo del proyecto, se decidió realizar un promedio de las tres últimas mediciones del sensor para disminuir ese error asociado. Aunque es una técnica sencilla, se vieron buenos resultados y por esta razón, se decidió trabajar con ella.

Para hacer esto, se utilizan las variables px0, px1, px2 para la coordenada X y las variables py0, py1, py2 para la coordenada Y. Primero, se obtiene el valor de las coordenadas dado por el GPS, el cual sería el valor actual de la posición, que se guarda en las variables que terminan en px2 y py2. Conforme avanza la simulación, en las otras variables se guardan los valores de las dos mediciones anteriores realizadas por el sensor. Una vez que se tienen los tres valores actualizados, se realiza el promedio de las tres coordenadas y se guarda el resultado en las variables robotX y robotY. Para los dos primeros ciclos de simulación, las dos últimas variables no tendrán datos. Por esta razón, a robotX y robotY se les asigna directamente el valor actual medido por el GPS mientras se presente esa condición.

Luego de asignar estas variables, se calcula la distancia entre el robot y la meta, y el resultado se guarda en la variable dGoalR. Luego, se calcula el ángulo entre el robot y la meta con respecto al marco de referencia global y se guarda su valor en angleGR. Un aspecto importante de esta parte es que se trabaja con la función *atan2( )* para obtener el ángulo, ya que así se obtiene en el cuadrante correcto y se trabajaría de igual forma a como se había explicado con el IMU (ángulos positivos o negativos entre 0 y 180°).

Una vez calculadas estas dos variables, se realizan las comparaciones respectivas para determinar en qué intervalo de distancia se encuentra el robot con respecto a la meta y así poder asignarle el valor del gradiente del campo potencial, en función de esta condición. Para visualizar mejor este proceso, se puede observar la siguiente figura.

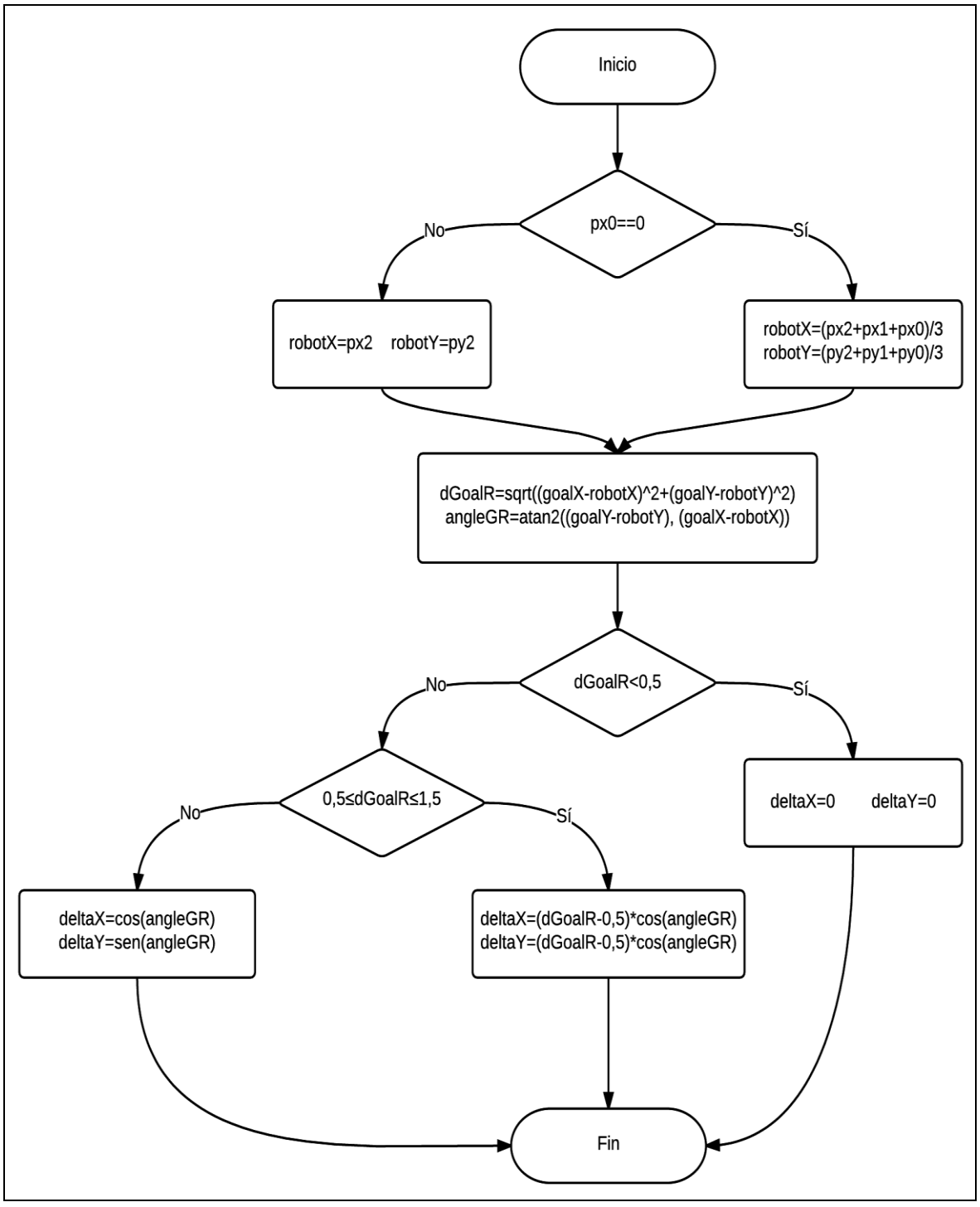



**Figura 20. Diagrama de flujo para calcular el campo potencial de atracción entre el robot y la meta. Elaborado por el autor en LucidChart.**

Una vez que se calcula el campo de atracción, se calcula el de repulsión entre el robot y los obstáculos. Es importante recalcar que el cálculo de estos campos es totalmente reactivo; es decir, se calcula para el punto más cercano detectado en ese instante. La distancia más cercana no se requiere calcular ya que anteriormente en el *script* del Kinect se había hecho esto. Lo único que se hace es asignar el valor global de esa distancia en la variable Txy.

Las variables usadas para esta parte del código se muestran a continuación:

**Tabla 7. Variables utilizadas para el cálculo del campo potencial de repulsión entre el robot y el obstáculo. Elaborado por el autor.**

| Variable | Significado |
|---|---|
| angleZ | Ángulo de orientación del robot medido por giroscopio |
| angleOb | Ángulo del obstáculo con respecto al sensor Kinect |
| angleObR | Valor corregido de la ubicación del robot para la coordenada X |
| TCx | Coordenada en el eje X del Kinect del punto más cercano detectado |
| TCz | Coordenada en el eje Z del Kinect del punto más cercano detectado |
| deltaX1 | Componente en X de la fuerza resultante de atracción |
| deltaY1 | Componente en Y de la fuerza resultante de atracción |

Para calcular el ángulo existente entre el robot y el obstáculo, se utilizan las coordenadas del punto más cercano detectado por el Kinect. Se trabaja con la coordenada Z del Kinect ya que el plano bidimensional XY por el que navega el robot coincide con el plano XZ del Kinect. Por lo tanto, la coordenada Z estaría funcionando como la coordenada Y en el sistema de referencia global. Otro punto importante con respecto al cálculo del ángulo es que también se utiliza la función *atan2( ),* ya que con ello se obtiene el ángulo en el cuadrante correcto y se trabajaría con el mismo formato del IMU, cuyos datos también se requieren para los cálculos en esta etapa.

Como el ángulo descrito anteriormente se obtiene con respecto al marco de referencia del Kinect, que también coincide con el del robot, se debe realizar un procedimiento extra para que se encuentre referenciado con respecto al marco global. Primero, el ángulo se guarda en la variable *angleOb*. Luego, se obtiene el ángulo de orientación del robot dado por el IMU y se guarda en *angleZ*. Si la suma de estos dos ángulos es mayor a π o menor a -π, el ángulo resultante referenciado con respecto al marco

universal viene dado por la resta de *angleOb* a *angleZ*; caso contrario, ambos ángulos se suman. Este resultado se guarda en *angleObR.*

Esto se realiza de esta forma ya que *angleOb* se mide con respecto al eje Z del Kinect y los valores máximos que puede tener son ±28,5°. Entonces, si se quiere una referencia con respecto al marco universal, solo se deben sumar ambos ángulos en el caso de que el resultado no sobrepase el valor de ±π, sino el ángulo se encontrará en otro cuadrante y el resultado correcto viene dado por la resta de *angleOb* y *angleZ.*

Una vez que se termina de trabajar con el ángulo, se hacen las comparaciones necesarias para saber en qué intervalo de distancia con respecto al objeto se encuentra el robot y así asignarle el gradiente correspondiente.

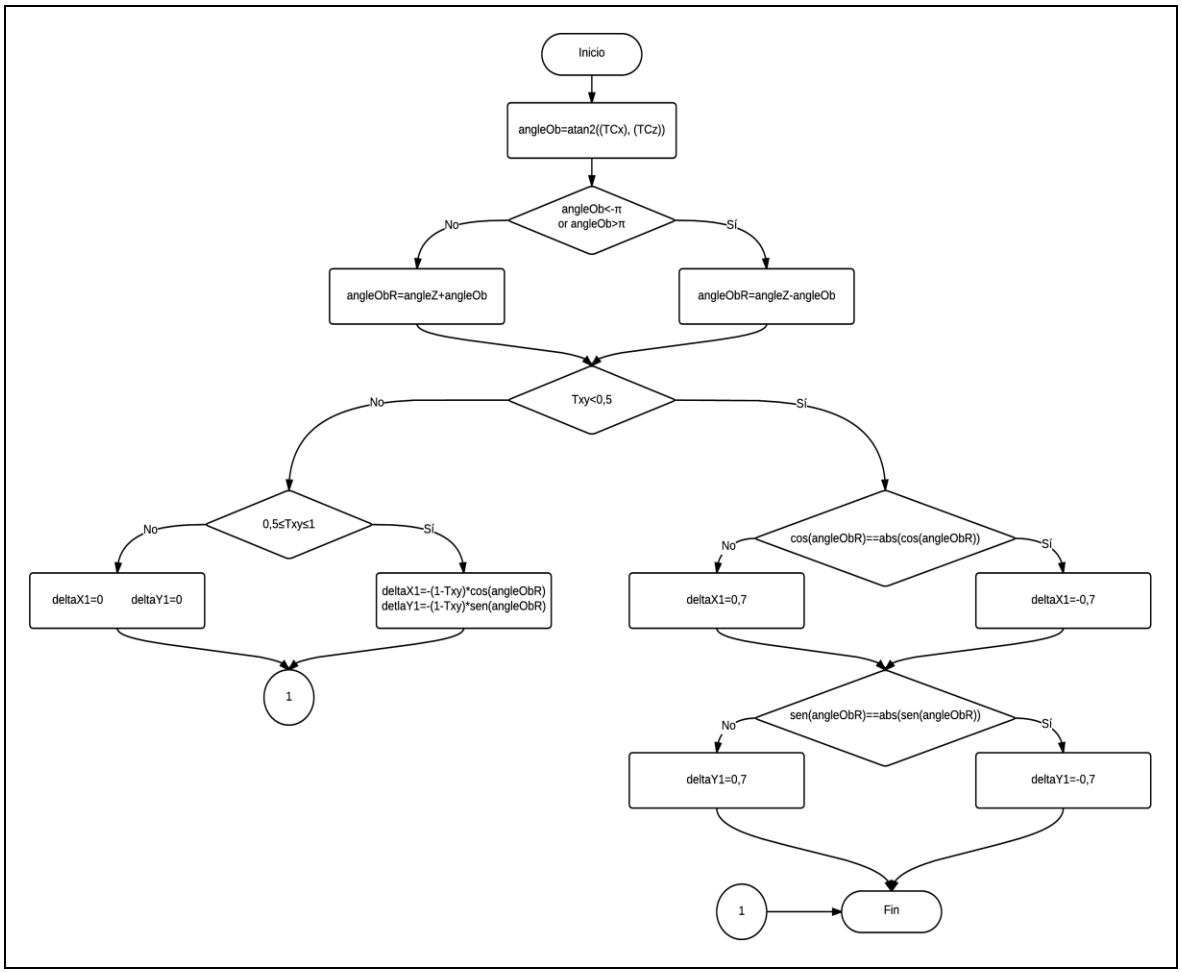



**Figura 21. Diagrama de flujo para calcular el campo potencial de repulsión entre el robot y el obstáculo. Elaborado por el autor en LucidChart.**

Cuando se terminan de calcular los gradientes de ambos campos, los componentes se suman para obtener el vector final que da la dirección de la trayectoria que debe seguir el robot para evitar obstáculos y llegar a la meta. Este vector también da una velocidad proporcional a la que debe navegar el robot de acuerdo a las distancias a las que se encuentre de la meta y los objetos.

### 4.2.4. Control de velocidad y ángulo resultante de la trayectoria del robot

Una vez que se calcula el vector resultante por medio de campos potenciales, se procede a realizar el control de la velocidad y el ángulo resultante de la trayectoria. Para obtener el ángulo final con respecto al cual debe girar el robot, se debe tener en cuenta el ángulo en el que se encuentra girado el robot y el ángulo resultante de la trayectoria dado por los campos potenciales; todo esto referenciado con respecto al marco universal.

Esto se hace con el fin de que el robot tenga que girar el menor ángulo posible y determinar si debe girar en sentido horario o anti horario y que así la navegación sea lo más eficiente posible. Para entender mejor esto, se puede observar la figura 22. Aquí se presenta una situación en la que la orientación del robot y la del ángulo de la trayectoria no coinciden. El *angleZ*, que es el ángulo medido por el IMU que brinda la orientación del robot, tiene un valor de 110°, mientras que el generado por los campos potenciales tiene un valor de 45°. Como se había mencionado anteriormente, todos los ángulos se trabajan con valores positivos o negativos entre 0 y 180° con respecto al eje X del marco de referencia global.

Para el caso de la imagen, ambos ángulos son positivos y se requiere que el robot gire en sentido horario para que la orientación de *angleZ* coincida con la de *anglePath*. Para lograr esto simplemente se resta *angleZ* de *anglePath*. El valor que se obtiene indica cuánto debe girar el robot para alcanzar la posición deseada y el signo que posee indica el sentido en el que debe girar. El signo positivo indica sentido horario y el signo negativo sentido anti horario. Para este caso, el resultado es de 65°, por lo tanto el robot debe girar en sentido horario.

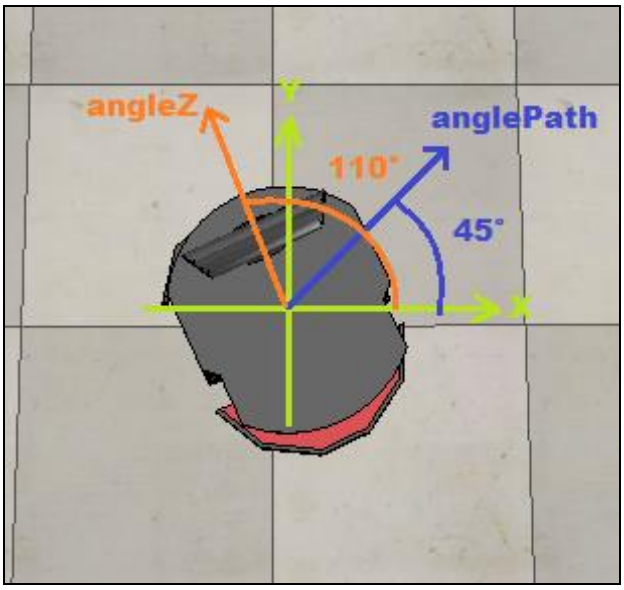



**Figura 22. Ángulos de orientación con mismo signo con respecto al marco de referencia global. Elaborado por el autor en Paint.**

El procedimiento aplicado anteriormente funciona para los casos en los que los ángulos poseen el mismo signo, es decir ambos son negativos o ambos son positivos. Por otro lado, dentro de este caso específico puede presentarse la situación de que los ángulos son iguales. Si esto sucede, tanto el ángulo resultante como el sentido de giro serán cero, lo que quiere decir que el robot no debe girar y debe mantener su trayectoria.

A nivel del código, se trabajan con dos variables, una que representa el sentido en el que se debe girar y otra que almacena el ángulo resultante. La primera se llama *sentido* y puede tener tres valores: 1 en caso de que el robot tenga que girar en sentido horario, -1 en caso de girar anti horario y 0 en caso de que no se deba girar. La segunda variable, se llama *angle* y guarda el valor absoluto de la resta de *angleZ* y *anglePath*.

El diagrama de flujo de cómo se implementó este caso específico del código se presenta en la siguiente figura:

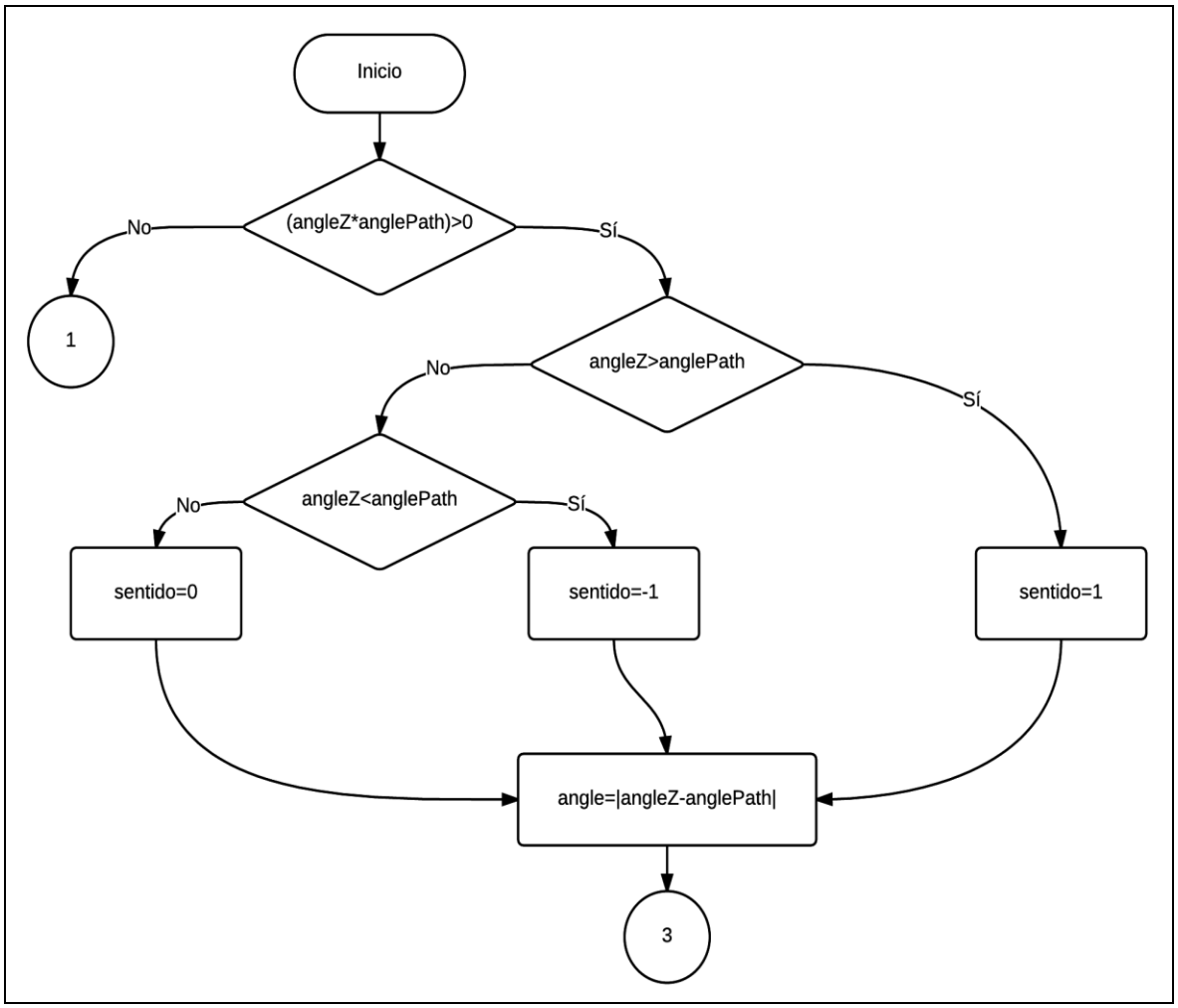



**Figura 23. Cálculo del ángulo y el sentido de giro para el caso de angleZ y anglePath con el mismo signo. Elaborado por el autor en LucidChart.**

El siguiente caso que se puede presentar es cuando los signos de *angleZ* y *anglePath* son diferentes. Ahora la diferencia que se tiene es que no siempre se puede restar el ángulo de la trayectoria dada por los campos potenciales del ángulo del robot para obtener el ángulo resultante, ya que si se hace de esta forma, no se estaría trabajando con el menor ángulo que debe girar el robot y la navegación se tornaría ineficiente.

Para ejemplificar esto, se puede observar la figura 24. Se puede ver que ahora se tiene un ángulo positivo y otro negativo. Si se aplica el criterio anterior, el resultado de restar ambos ángulos daría 270° en sentido horario. Este es un resultado válido pero no es

el más eficiente, ya que el ángulo que se requiere para que se gire lo menor posible es el complemento de 270°, o sea 90°.

La condición con la cual sucede esto es cuando la resta de ambos ángulos es mayor a 180° o menor a -180°. De ser así, se debe restar el valor absoluto de *anglePath* del valor absoluto de *angleZ*, lo cual dará como resultado la magnitud que debe girar el robot. Para determinar el sentido de giro, simplemente se compara si *angleZ* es mayor a *anglePath* y de ser así se gira de forma anti horaria, caso contrario se gira de forma horaria.

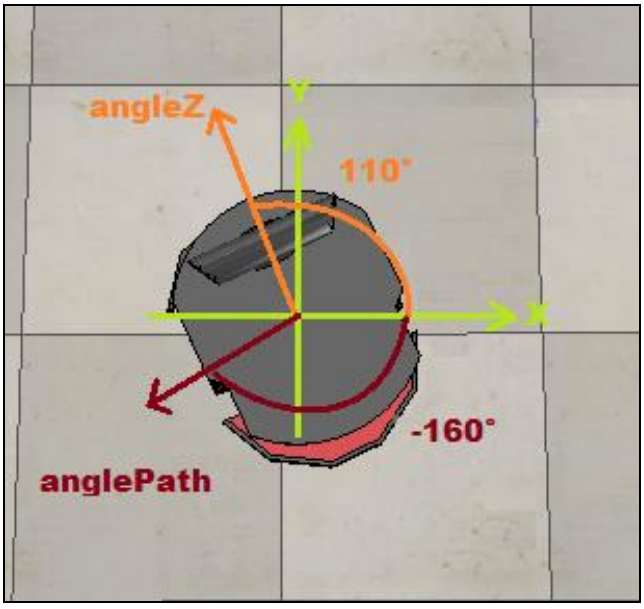



**Figura 24. Ángulos de orientación con diferente signo con respecto al marco de referencia global. Elaborado por el autor en Paint.**

Cuando la resta de los dos ángulos de diferente signo no es mayor a 180° o menor a -180°, se puede seguir trabajando con el criterio de restar *anglePath* de *angleZ* y determinar el sentido de giro por el signo del resultado.

A nivel de código, este proceso se trabaja por medio de comparaciones, donde primero se determina cuál de los dos ángulos es positivo. Esto permite saber que el robot debe girar en sentido horario en caso de cumplirse la condición o en sentido contrario en caso de que no se cumpla. Antes de asignar las variables *angle* y *sentido*, se compara si la suma de los valores absolutos de los ángulos sobrepasa 180°, ya que de ser así se aplica el procedimiento descrito anteriormente.

A continuación se muestra el diagrama de flujo que resume esta etapa del código:

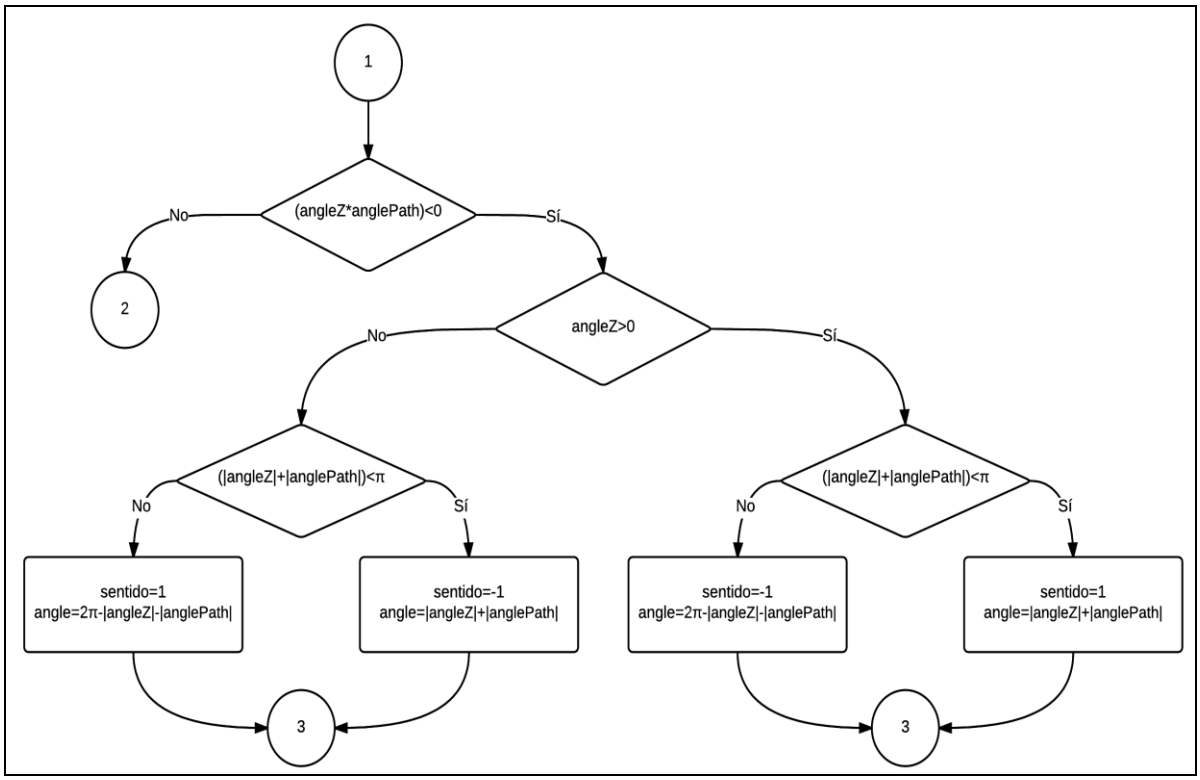



**Figura 25. Cálculo del ángulo y el sentido de giro para el caso de angleZ y anglePath con diferente signo. Elaborado por el autor en LucidChart.**

Si ninguna de las dos condiciones generales descritas anteriormente se cumple, esto quiere decir que al menos uno de los dos ángulos es igual a cero. Esta parte del código se inicia comparando si los dos ángulos son iguales, lo que quiere decir que ambos son cero. Esto implicaría que el robot no tendría que modificar su trayectoria por lo que los valores de *sentido* y *angle* serían 0.

Si uno de los dos es diferente de cero, se determina cuál de ellos lo es y se vuelve a comparar si son positivos o negativos. Esto permite saber que si el ángulo positivo es *angleZ*, el robot debe girar en sentido horario y en caso de ser *anglePath* se girará en sentido contrario. En caso de que sean negativos, se realiza lo contrario. Este proceso se puede visualizar en la figura 26.

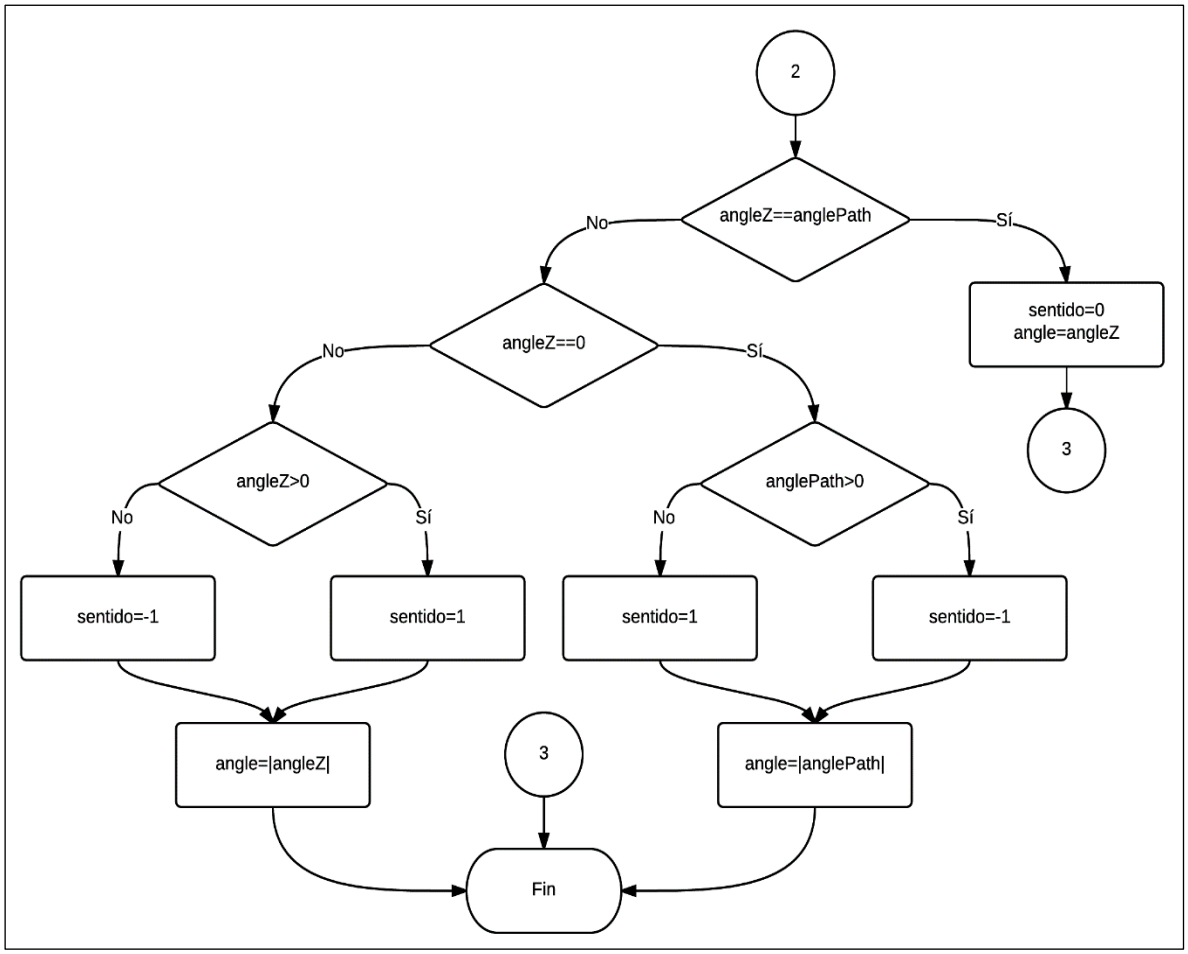



**Figura 26. Cálculo del ángulo y el sentido de giro para el caso en el que uno o ambos ángulos de orientación son cero. Elaborado por el autor en LucidChart.**

Como comentario final de este algoritmo para calcular el ángulo resultante, se puede observar que los valores resultantes guardados en la variable *angle*, siempre serán menores o iguales que 180°, independientemente del sentido de giro que se le asigne al robot.

Explicado el algoritmo para determinar el menor ángulo entre el robot y la trayectoria definida por los campos potenciales, así como el sentido de giro, se procede a explicar el control de velocidad del robot.

Primero, se recuerda que por medio del vector resultante de la suma de los campos de atracción y de repulsión, se puede obtener un valor proporcional a la velocidad a la cual se desea navegar. Este valor proporcional se calcula como la norma de dicho vector, lo cual es igual a la raíz cuadrada de la suma de cada uno de sus componentes al cuadrado.

Para obtener la fórmula por medio de la cual se regirá la velocidad del robot, se utilizó como base el modelo cinemático de un robot de tracción diferencial de dos ruedas. El esquema que lo describe se muestra en la figura siguiente.

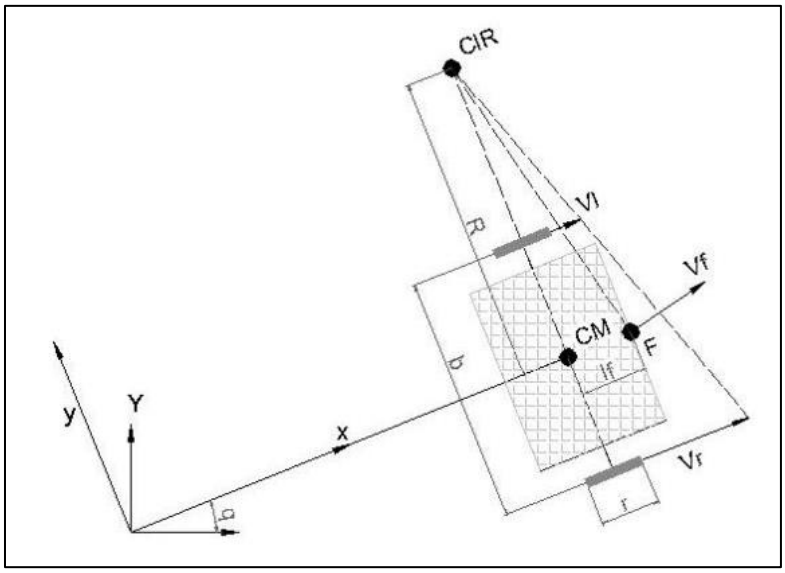



**Figura 27. Modelo cinemático de un robot móvil de tracción diferencial de dos ruedas. (Bañó, 2003)**

Las ecuaciones de velocidad lineal y angular que describen este modelo se presentan a continuación:

$$v = \frac{vr+vl}{2} \tag{4.11}$$

$$\omega = \frac{vr-vl}{b} \tag{4.12}$$

La variable $v$ corresponde a la velocidad lineal del robot, $\omega$ a la velocidad angular, $vr$ es la velocidad lineal de la llanta derecha, $vl$ es la velocidad lineal de la llanta izquierda y $b$ es la distancia entre el centro de una llanta y el centro de la otra (Bañó, 2003). El siguiente paso es reescribir estas ecuaciones en función de la velocidad angular y la velocidad lineal, ya que lo se requiere saber es la velocidad que se le debe asignar a cada llanta del robot.

También se reescriben las fórmulas porque con los campos potenciales se obtuvo un valor proporcional que modificará la velocidad máxima del robot, en función de la distancia

a la que se encuentre el robot de los obstáculos o de la meta. Las fórmulas reescritas en función de la velocidad lineal y angular del robot quedan de la siguiente forma:

$$vr = v + \frac{\omega * b}{2} \tag{4.13}$$

$$vl = v - \frac{\omega * b}{2} \tag{4.14}$$

Estas fórmulas describen la velocidad de las ruedas por medio de un componente lineal y otro componente angular. Estas fórmulas están referenciadas con respecto al marco de la figura 27, donde el robot está girando en sentido anti horario. Para el caso donde el robot gira en sentido horario, el único cambio que ocurre es que a la velocidad de la derecha se le restará el componente de la velocidad angular y a la velocidad de la izquierda se le sumará este componente.

Una vez que se comprende que las velocidades de las ruedas están compuestas por un componente de velocidad lineal y angular, se desarrolla una ecuación específica de velocidad basada en las ecuaciones recién vistas y la metodología utilizada para evitar obstáculos. Ya que esta técnica presentó buenos resultados anteriormente, se decidió combinarla con el modelo del robot, esperando obtener mejores resultados y así una ecuación adaptada a la aplicación.

Uno de los puntos que se decidió mantener del algoritmo de evitar obstáculos que se diseñó con anterioridad, fue el de trabajar por medio de intervalos de distancias. El valor del primer intervalo se mantuvo igual, que es cuando la distancia del robot al obstáculo (guardado en la variable Txy) es menor o igual a 0,5 metros.

Si la distancia se encuentra en este intervalo, el robot únicamente tendrá velocidad angular (gira sobre su propio eje) con el fin de que pueda evitar el obstáculo y así alejarse de una posible colisión con él. El sentido de giro está determinado por el cálculo del ángulo resultante que se hizo anteriormente. En este punto el robot también girará con la máxima velocidad posible, por lo que no se usa el valor proporcional dado por los campos potenciales.

Este valor proporcional se utiliza para el siguiente intervalo, que es cuando la distancia del robot a los obstáculos es mayor a 0,5 metros. La diferencia con respecto al

algoritmo de evitar obstáculos es que ahora no se precisan de más intervalos ya que el valor proporcional se encarga de definir la velocidad en función de la distancia a la que se encuentra de los obstáculos y la meta.

Para este caso, se definió también una velocidad angular proporcional al ángulo resultante que tiene que girar el robot. Con anterioridad se mencionó que este ángulo nunca iba a superar 180° (π radianes). Por lo tanto, si se quiere girar ese ángulo máximo, se debe establecer la velocidad máxima (1,2 m/s), y para un ángulo específico (variable *angle*) que se quiera girar, se obtiene la siguiente fórmula de velocidad que vendría a reemplazar el término $^{(\omega * b)}/_{2}$ de las ecuaciones (4.13) y (4.14):

$$v2 = \frac{1{,}2 * angle}{\pi} \tag{4.15}$$

Por otro lado, el término de la velocidad lineal de las ecuaciones anteriores se sustituiría por el valor proporcional dado por los campos potenciales multiplicado por la velocidad máxima del robot. Esto hace que las ecuaciones de velocidad específicas para la aplicación basadas en el modelo cinemático del robot y el algoritmo para evitar obstáculos diseñado anteriormente, sean las siguientes:

$$vr = 1{,}2(v + \frac{angle}{\pi}) \tag{4.16}$$

$$vl = 1{,}2(v - \frac{angle}{\pi}) \tag{4.17}$$

Como en ambos términos se multiplica la constante 1,2, ésta se saca a factor común. Se puede ver que estas nuevas ecuaciones poseen tanto el componente angular como el lineal, únicamente que ahora el lineal es proporcional al factor *v* y la velocidad angular es producto de la relación del ángulo que se debe girar y la velocidad máxima del robot. De igual forma, estas ecuaciones son para girar en sentido anti horario, caso contrario el componente angular se suma a *vl* y viceversa para *vr*.

Para este intervalo, también se trabaja con la variable *sentido*, para saber si el robot debe girar en un sentido o debe mantener su trayectoria. En caso de que el robot deba mantener su trayectoria, el componente de la velocidad angular se torna cero.

La combinación de estos métodos significó una solución muy eficaz para solucionar el problema planteado.

## 4.3. WayPoints

Los "*waypoints*" se utilizaron con el fin de que el robot pueda navegar mayores distancias y así no tenga problemas con los mínimos locales que pueden presentarse durante la navegación del robot. Una consideración importante de utilizar esta técnica, es que para fines de la aplicación, el usuario debe tener un conocimiento básico de dónde podrían originarse estos puntos conflictivos.

Una vez que se definen los puntos por los cuales navegará el robot, se genera una tabla en la que se guardan las coordenadas X, Y de cada uno de ellos. En el código, esta tabla se llama *waypoint*. También, se cuenta con una variable llamada *iway* la cual se utiliza para recorrer dicha tabla y por último se tiene la variable *waypointpos* en la cual se almacena el "*waypoint*" respectivo al que se desea llegar.

El código comienza con la inicialización de la variable *waypointpos,* a la cual se le asigna la primera posición guardada en la tabla de coordenadas. Posteriormente, se compara si ya se ha llegado al último punto de navegación almacenado en la tabla. Si esta condición se cumple, se acaba el algoritmo.

Si todavía no se llega al último punto de navegación, el valor de *waypointpos* se debe actualizar hasta llegar al punto final. Sin embargo, este valor no se actualiza hasta que el robot se ubique a una distancia menor de 0,7 metros del punto de navegación al que se debe ir. Una vez que se cumple esa condición, se actualiza el valor y el robot continúa su trayectoria hasta el siguiente "*waypoint*". Este procedimiento se puede observar en la figura 29.

Es importante mencionar que, conforme el robot se acerca a cada uno de los "*waypoints*", este irá disminuyendo su velocidad de acuerdo a los parámetros que se definieron anteriormente en la sección de campos potenciales. Por esta razón, se define el valor de 0,7 metros de distancia al "*waypoint*" objetivo, para que se actualice su valor con las coordenadas del siguiente, porque si se coloca a una distancia más cercana, el robot irá

muy despacio hasta alcanzar el punto y durará más tiempo en su trayectoria. Si se coloca más lejos el robot, no seguirá la ruta tan bien como se desea.

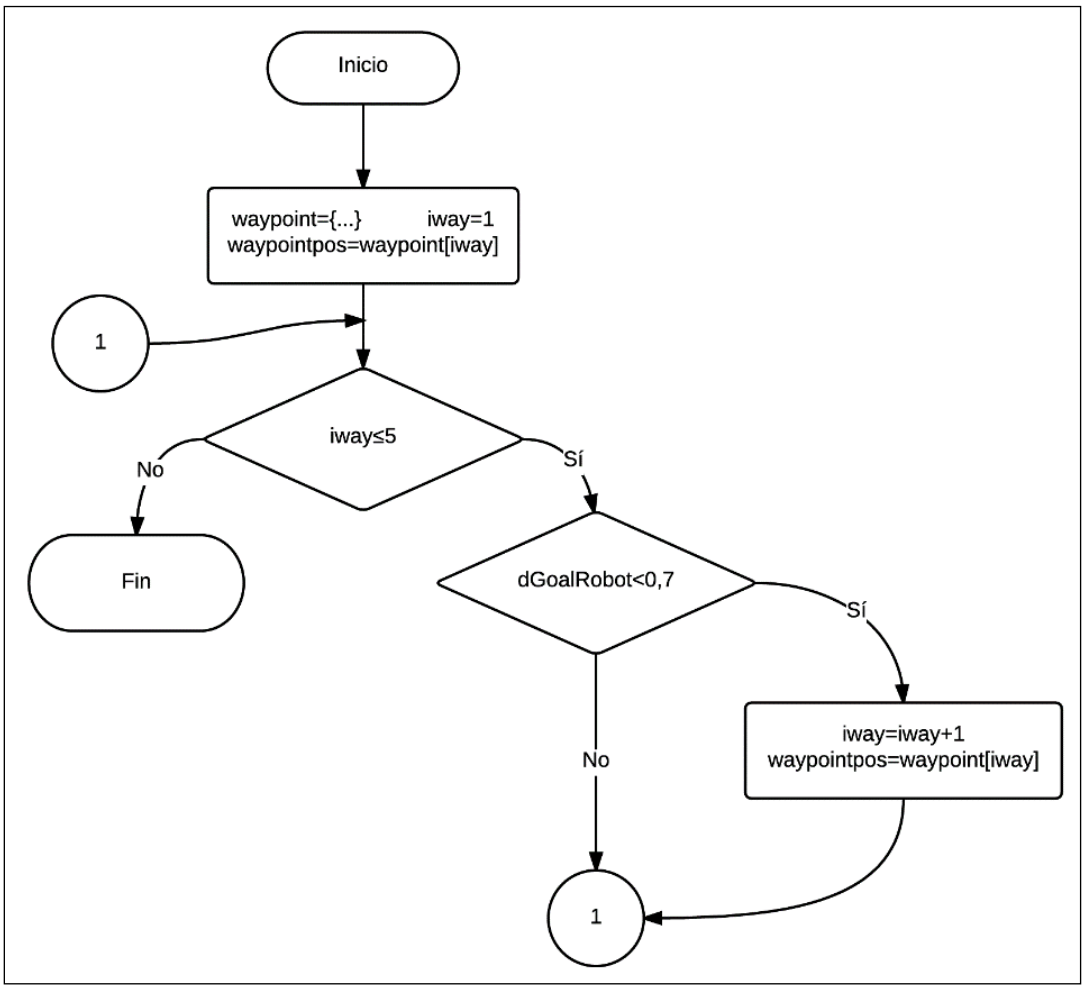



**Figura 28. Algoritmo de los "waypoints". Elaborado por el autor en LucidChart.**

Otro punto importante de utilizar esta técnica, es que durante la simulación se pudo observar que mejoró la trayectoria seguida por el robot, haciendo que el error del GPS cada vez sea menos significativo.

## 4.4. Descripción de la simulación realizada

Una vez que ya se explicó el funcionamiento de cada una de las partes que componen el sistema y los algoritmos de control respectivos, se requiere probar todo ello en conjunto en un ambiente con condiciones similares a las que se podrían tener en un espacio real.

Para ello, se modeló un espacio que podría verse como un conjunto de oficinas, el cual tiene un área de 400 $m^2$ (20x20). Los pasillos de los extremos tienen un ancho de 1.70 metros, mientras que el del medio posee un ancho de 2 metros. Los espacios de todas las puertas tienen un ancho de un metro.

Se puede observar que en los pasillos y en las diferentes oficinas se encuentran objetos cotidianos como sillas, mesas, sillones, entre otros. Esto con el fin de que se pareciera lo más posible a un ambiente real.

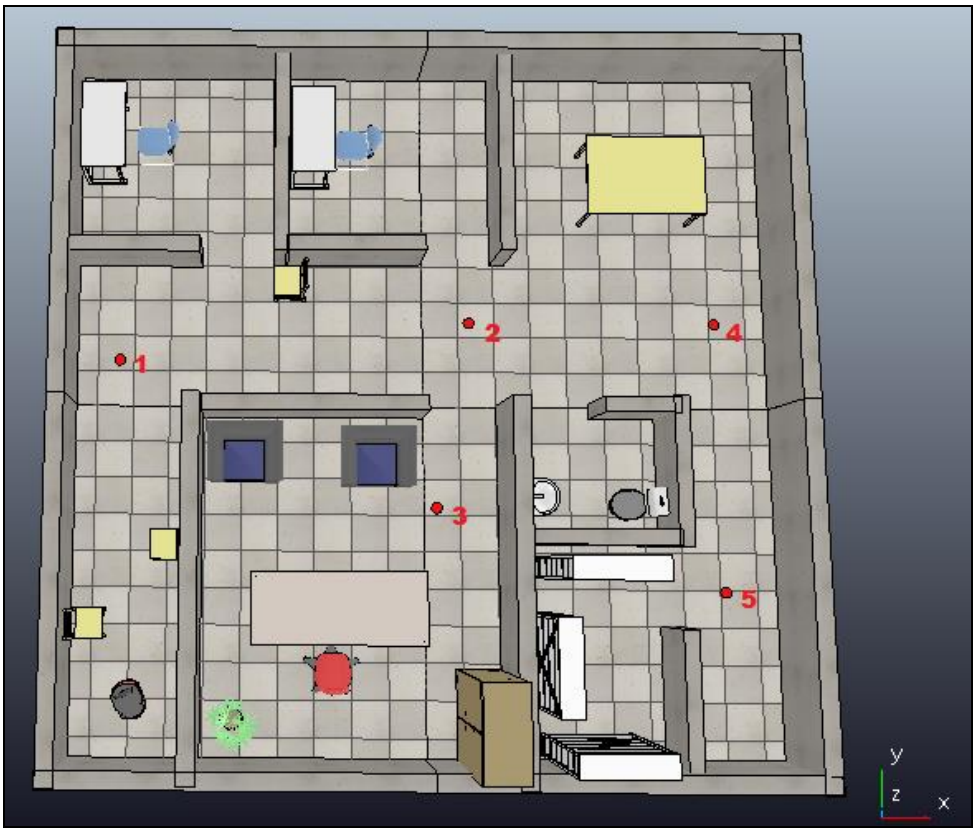



**Figura 29. Ambiente modelado para probar la aplicación. Elaborado por el autor en V-REP.**

En la imagen se puede observar que en distintas zonas se encuentran puntos rojos en el piso, marcados con números. Estos no tienen ninguna influencia sobre la navegación del robot, simplemente sirven como una marca visual para señalar donde se ubican los "*waypoints*" seleccionados para que el robot navegue por una trayectoria específica. También se puede observar en la esquina inferior el marco de referencia global.

En la siguiente tabla se muestran las coordenadas de cada uno de los "*waypoints*" y de la posición inicial del robot, con el fin de que se tenga una noción de las distancias que navega el robot en el ambiente modelado.

**Tabla 8. Coordenadas de los "waypoints" y la ubicación inicial del robot. Elaborado por el autor.**

| Objeto | Coordenada X | Coordenada Y |
|---|---|---|
| **Waypoint 1** | -4,2 | 1,0 |

| | | |
|---|---|---|
| **Waypoint 2** | 0,65 | 1,2 |
| **Waypoint 3** | 0,1750 | -1,4 |
| **Waypoint 4** | 4,075 | 1,2 |
| **Waypoint 5** | 4,075 | -2,525 |
| **Posición inicial robot** | -3,97 | -4 |

El primer trayecto se hace desde la posición inicial hasta el "*waypoint1*". La intención de este tramo es probar el comportamiento del robot en un pasillo con obstáculos y analizar que tan bien se define la trayectoria del robot y la velocidad por medio de los campos potenciales y las mediciones de los sensores.

El segundo tramo que va del punto 1 al 2 se hizo para probar qué tan bien giraba el robot en un esquina para llegar el siguiente punto y con ello verificar el funcionamiento de la función de velocidad desarrollada especialmente para la aplicación. Para el trayecto del punto 2 al 3, se quería probar la capacidad del robot para entrar en espacios más reducidos y con mayor cantidad de objetos.

El tramo de vuelta del punto 3 al 2 es para analizar que tan bien responde el robot cuando su orientación es casi opuesta al punto al que debe llegar y puede encontrarse con objetos de frente mientras gira casi 180° grados para reorientarse hacia la coordenada de destino.

Por último, los tramos de los puntos 3 al 4 y del 4 al 5, son para corroborar el funcionamiento de la aplicación en pasillos sin obstáculos donde solo se ve influenciado por las paredes que los delimitan.

Una vez que se ejecutó el algoritmo, se pudo observar que los resultados fueron los deseados. En el capítulo 6, esto se presenta con detalle.

# Capítulo 5: Implementación con sensores en físico

Lo que se pretende con esta parte del proyecto es corroborar los datos que se obtuvieron en la simulación. En otras palabras, se quiere verificar si las señales de actuación sobre el robot serán las mismas al trabajar con los datos de los sensores en físico y saber si con ello se tendrá el mismo comportamiento que el que se obtuvo a nivel de la simulación.

En esta etapa, se trabajó con dos programas para procesar la información de los sensores. Para las mediciones provenientes del IMU y el GPS se siguió trabajando en V-REP, ya que este programa tiene la ventaja de leer datos seriales con mucha facilidad. Por otro lado, los algoritmos desarrollados anteriormente se implementaron en el lenguaje de programación de este simulador, por lo que no se requiere trasladar el código a otro lenguaje.

Al trabajar con los datos reales de estos sensores, ya no se requiere utilizar los *scripts* de sus modelos de simulación. Lo único que se requiere es leer los datos de cada

uno desde el puerto serial y asignarlos a las variables globales que anteriormente almacenaban los valores brindados por esos *scripts*.

Para el caso del Kinect, el programa utilizado fue Octave. Esto es debido a que los datos que se obtienen de este sensor se trabajan como imágenes RGB, donde la información de profundidad se encuentra en trece bits de un pixel, repartidos entre el canal R y el canal G. Utilizar Octave permite el procesamiento de imágenes por medio de cada uno de sus canales por separado, además de que cuenta con un paquete de procesamiento de imágenes muy versátil y útil para la aplicación que se está trabajando; complemento con el que no cuenta V-REP. Otro punto importante es que V-REP tiene la posibilidad de comunicarse con Octave por medio de un API remoto.

Definidos los programas en los que se procesará la información de cada sensor, se procede a explicar la configuración requerida para el correcto funcionamiento de cada uno y la interacción en los programas.

## 5.1. Configuración de los sensores

A partir de este punto se puede ver que el funcionamiento de este proyecto se basa en tres sensores: un GPS, un IMU y un Kinect. Obtener los datos de ellos varía un poco con respecto a lo que se hace en el simulador.

Para el caso del GPS, se utilizó un receptor con el que cuenta el laboratorio, cuyo modelo es un XC-GD75. La ventaja de este receptor es que tiene salida USB, por lo que se pueden leer sus datos por medio de un puerto serial de este tipo. Las especificaciones más relevantes de este receptor son las siguientes (Sunsky, 2015):

- Rastreo de hasta 20 satélites.
- Tasa de actualización de datos de 1 Hz.
- Precisión de las mediciones de 5 metros.
- "*Baud Rate*" de 4800 bps.


**Figura 30. Receptor GPS, modelo XC-GD75. Recuperado de: (Sunsky, 2015)**

De las características anteriores se puede observar que la que puede generar mayores problemas para el funcionamiento de la aplicación real es la de la tasa de actualización de datos. Sin embargo, como en esta etapa del proyecto lo que se quiere es verificar el comportamiento del robot para situaciones específicas (robot con una orientación y posición fija), su uso es válido y no representa ninguna desventaja para fines de validación.

Por otro lado, se puede observar que este receptor puede rastrear una buena cantidad de satélites que componen este sistema de localización (20 de 24). Esto permite tener mejores mediciones, ya que se pueden realizar más triangulaciones de posicionamiento. A la vez, se puede ver que la precisión de 5 metros de este sensor es la misma que la que se utilizó a nivel de simulación, por lo que los resultados con el sensor real deberían responder de forma muy similar. Otro dato importante que se brinda es el del "*Baud Rate*", ya que si no se contara con él, no se podría inicializar correctamente la lectura de la información que llega al puerto serial.

La información que brinda este sensor se encuentra en padrón NMEA. Este padrón define aspectos como: características eléctricas de la señal, protocolo de la transmisión de datos, sincronismo y formatos específicos de las sentencias de transmisión (Carvalho, 2004). De los formatos específicos que puede brindar el GPS, el que realmente interesa es el GGA, ya que este contiene los datos de latitud y longitud de la posición medida. En la sección de anexos se explica con más detalle el padrón NMEA, así como los elementos del formato GGA.

Para el caso del IMU, también se trabaja con un modelo presente en el laboratorio, el 9DOF *Razor* IMU. Este incorpora un giroscopio de triple eje ITG-3200, un acelerómetro de triple eje ADXL345 y un magnetómetro de triple eje HMC583L. De esos tres, únicamente se trabaja con los datos del ITG-3200. Algunas de las especificaciones más relevantes de este sensor son las siguientes (Sparkfun, 2015):

- Salida de datos de los tres sensores procesada por un ATmega328, con salida a interfaz serial.
- "*Baud Rate*" de 57600 bps.
- Pines de salida compatibles para "FTDI *Basic Breakout*".

Las características anteriores muestran que este sensor tiene prestaciones muy favorables para el tipo de aplicación que se está desarrollando. Primero, se tiene la ventaja de que tiene un gran valor de "*Baud Rate*", lo que permite enviar más información por medio de la comunicación serial y con mayor velocidad. Por otro lado, se puede ver que la salida de datos es por comunicación serial. Sin embargo esta salida no es para un puerto USB. Es aquí donde la última especificación es muy útil, ya que al tener pines de salida compatibles para un "FTDI *Basic Breakout*", se puede convertir la información serial proveniente del ATmega328 al formato serial para USB. La configuración de conexión entre estos dos elementos se puede observar en la sección de anexos.

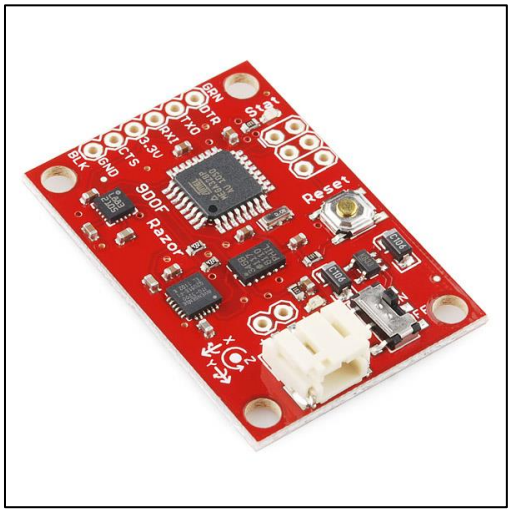



**Figura 31. 9DOF Razor IMU. Recuperado de: (Sparkfun, 2015)**

Para el último sensor, el Kinect, se trabajó con el modelo más nuevo, el Kinect II. El motivo de utilizar este nuevo modelo es porque posee una mayor resolución y más detalle, así como un mayor ángulo de visión. Las especificaciones más relevantes de este sensor son las siguientes (Montserrat, 2015):

- Ángulo de visión en el eje X de 70° y de 60° en el eje Y.
- Mayor alcance de profundidad hasta 4,5 metros.
- Mayor resolución de la calidad de las imágenes.

Como se puede ver, las especificaciones que se mostraron son diferentes con respecto a las que se presentaron en el Capítulo 4 para el Kinect I. Sin embargo, esto no representa ningún problema porque el funcionamiento que se describió anteriormente para calcular la profundidad y analizar el "*point cloud*" se mantiene. Por eso a nivel del código simplemente se requieren escalar los valores de los ángulos de visión y aplicar una regla de tres para la resolución de la imagen obtenida. Una vez realizados estos cambios, la validez del código usado en la simulación para calcular la profundidad en el "*point cloud*" sigue siendo correcta para esta aplicación.

**Figura 32. Kinect II. Recuperado de: (Duarte, 2013)**

Descritas las características de cada uno de los sensores que se usaron para corroborar los datos de la simulación, se procede a explicar la solución realizada para

procesar la información de cada uno de los sensores y obtener los resultados de las variables que posteriormente accionarán el movimiento del robot.

## 5.2. Descripción de la solución

Al inicio del capítulo se explicaron las razones por las que se decidió utilizar V-REP para procesar los datos del IMU y el GPS, y Octave para los datos del Kinect. De forma general se podría decir que el procesamiento de imágenes y análisis del "*point cloud*" se realiza en Octave y en V-REP se realiza el procesamiento de datos de la posición del robot.

Para hacer esto, se guardaron los "*point clouds*" generados por el Kinect en diferentes situaciones; cada una de ellas a una distancia diferente de la meta y los objetos. La orientación del vehículo también varía de una situación a otra. Por lo tanto, las mediciones se realizan para un instante dado, para una situación específica en la que el robot se ubica en una cierta posición y los sensores están brindando la información de ese instante. Los casos específicos de cada configuración se muestran en la siguiente sección.

El ambiente en el que se tomaron las imágenes del Kinect fue un pasillo de la escuela de computación. Este tiene un ancho de 1,80 metros, al igual que uno de los

pasillos del ambiente simulado y presenta objetos en las paredes, los cuales podrían verse como obstáculos. En la figura 33 se puede observar dicho pasillo.

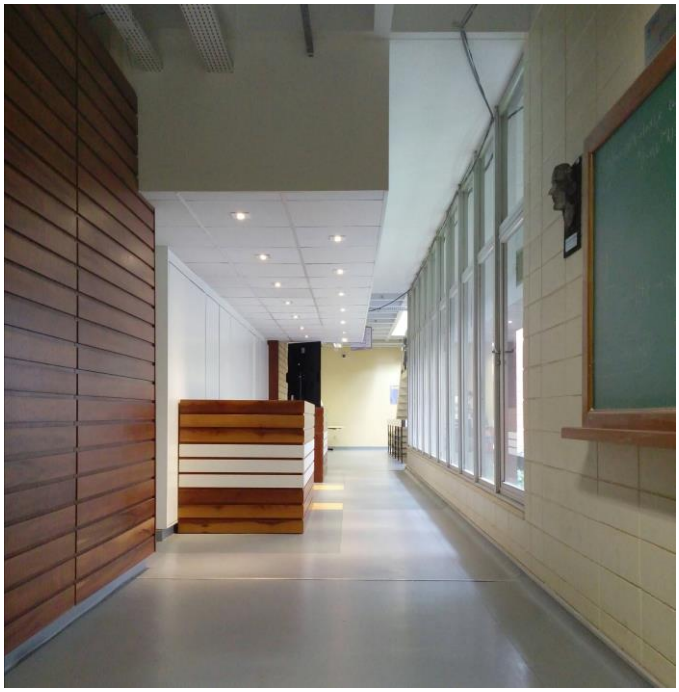


**Figura 33. Ambiente para comprobar los datos de simulación. Realizado por el autor.**

Dichas imágenes se cargaron en Octave y se guardaron como una matriz de dimensiones iguales a las de la imagen. El valor de cada pixel se guarda en la matriz en la misma posición i, j del archivo cargado. El procesamiento de las imágenes se inicia al aplicarles un filtro por medio del cual se realiza un suavizado. Esto se hace con el motivo de disminuir el efecto del ruido presente en ellas y así posteriormente obtener mejores resultados.

Específicamente, la función que se utilizó para esto se llama *imsmooth( )*, la cual por diferentes métodos puede realizar el suavizado de la imagen. El método que presentó mejores resultados fue el de *median*, en donde el valor de cada pixel es reemplazado por la media de los pixeles de un área local. Se utilizó un área de 3x3 pixeles para esta operación. Posteriormente, se guarda el canal R y G de la imagen suavizada, en nuevas variables.

Luego de realizar el suavizado, se puede proceder a aplicar el código del Kinect que ya se había trabajado en el Capítulo 4. De las especificaciones que se brindaron del sensor,

se sabe que se debe modificar el ángulo de visión en ambos ejes. Por ello se modifica el ángulo en el eje de visión X por 70° y el del eje Y por 60°.

Posteriormente, cuando se comienza a analizar el "*point cloud*", uno de los cambios más grandes que se presentaron fue el cálculo del *depthValue*. Esto porque la imagen de profundidades generada por este nuevo modelo del Kinect tiene una resolución de 512x424 pixeles, donde el canal R posee cinco bits de la información del valor de profundidad y el G posee ocho. Esto hace que la fórmula del *depthValue* cambie, así como su resultado, ya que no tendrá valores entre 0 y 1, sino que ahora estarán entre 0 y 8191 ($2^{13}$-1). Esto se debe a que la nueva fórmula está definida de la siguiente forma:

$$depthValue = 256 \times pixelR + pixelG \tag{5.1}$$

Esta fórmula se determinó al analizar los valores máximos que existen en cada canal para cada valor de los pixeles. De los 13 bits en los que se configura la profundidad, los del canal rojo representan los más significativos. Por ello, se multiplica su valor por 256 para que se mantenga esa condición. Al sumarse los bits de los pixeles verdes, que son los menos significativos, se obtiene el valor total de profundidad. Mayores valores de *depthValue* indican que el pixel está más lejos del sensor.

Posteriormente, como las demás fórmulas se realizaron para una resolución de 64x48, los valores que se obtienen de la posición i, j de la matriz analizada se deben escalar por un factor. En el caso de las coordenadas, se multiplica por 64/512 si son en el eje X, y por 48/424 sin son en el eje Y. Para el caso de los ángulos, se multiplica por 57/70 si son en el eje de visión X, y por 43/60 si son en el eje de visión Y. Una vez que se realizan estos procedimientos, se obtienen los valores respectivos de distancias en los tres ejes y la distancia mínima al punto más cercano. Luego, estos datos se pasan a V-REP para terminar de realizar los cálculos, junto con las mediciones de los otros sensores.

Para leer los datos de posición y orientación del robot, dados por el IMU y el GPS, se deben utilizar las siguientes funciones en V-REP:

- *simSerialOpen*( ): esta función recibe como parámetros un *string* del nombre del puerto serial del que se leerán los datos y el "*baud rate*" al que se recibe la información en dicho puerto. Siempre se utiliza en la inicialización para abrir y

guardar en una variable el identificador del puerto serial del que se recibirán datos.

- *simSerialCheck*( ): esta función lee cuántos bytes están esperando para ser leídos en el puerto serial. Únicamente recibe un parámetro y es el identificador del puerto, lo cual se obtiene por medio de la función anterior.
- *simSerialRead*( ): se utiliza para leer los datos de un puerto serial que se haya abierto con anterioridad y da como resultado el *string* con la información requerida.

Para el caso del GPS, se realizaron dos mediciones, una del primer '*waypoint*' al que se desea llegar, la cual se guardó como un dato estático en el programa, y otra que es la de la posición del robot, la cual se lee constantemente desde el puerto serial.

Como el GPS no funciona muy bien en ambientes interiores, ambas mediciones se realizaron en las afueras del edificio. Los datos que se reciben de las lecturas constantes desde el puerto serial corresponden a la posición en la que se ubica el robot en el pasillo. La otra medición, que corresponde a la del "*waypoint*" al que se desea llegar, se ubicó y midió a cinco metros de distancia del robot, sobre la latitud. Este dato se ingresa directamente en V-REP.

Para leer los datos del GPS, se utilizan las funciones anteriores. Primero, se abre el puerto serial correspondiente con la velocidad de 4800 bps que tiene este sensor. Una vez que se tiene el identificador de puerto, se utiliza la función *simSerialCheck( )* para saber si el resultado es mayor a diez bytes y así utilizar la función que lee el *string*. Si el resultado es menor que diez bytes, esto significa que el *string* no contiene toda la información necesaria y por ello no se lee. El *string* resultante que envía el GPS es el siguiente:

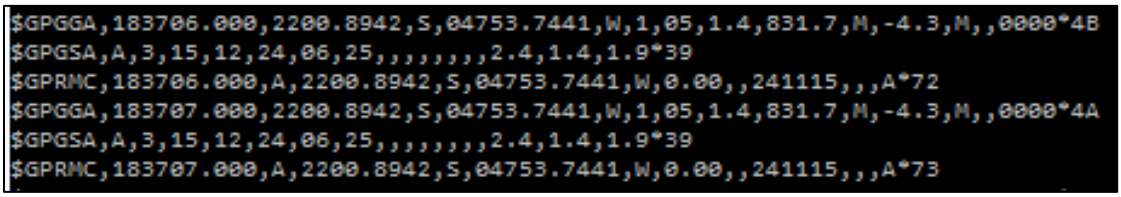



**Figura 34. Datos de posición leídos en V-REP por medio de comunicación serial con el GPS. Elaborado por el autor en V-REP.**

Como se puede observar, en el *string* se envían tres formatos distintos del padrón NMEA. Sin embargo, anteriormente se mencionó que la sección de interés de estos datos enviados es aquella en formato GGA. Una vez que se localiza esta línea, se deben substraer los valores de latitud y longitud. Estos datos son los que se ubican al lado izquierdo de las letras S y W, respectivamente.

Para substraerlos se usa la función *string.find( )*. Ésta recibe el *string* que se quiere analizar y un patrón específico que se quiere buscar en él. La función da como resultado el número de índice en el que se encuentra el primer y último caracter del patrón. Esta función se utiliza tres veces. La primera es para segmentar del *string* completo la línea con el formato GGA, por lo tanto se busca este patrón. Luego, se utiliza para buscar en esa línea las letras S y W. Una vez que se poseen los índices de dónde se ubican esas letras se pueden substraer los valores de latitud y longitud. El último paso consiste en convertir esa cadena de caracteres a un número. Finalizado este proceso, se pueden realizar los cálculos respectivos con dichas variables.

El proceso para leer los datos en el IMU es casi el mismo que para el GPS. Se usan las mismas funciones y se recibe un *string* del cual se requiere substraer un valor específico. Para este sensor, los datos se reciben a una velocidad de 57600 bps y se ven de la siguiente forma:

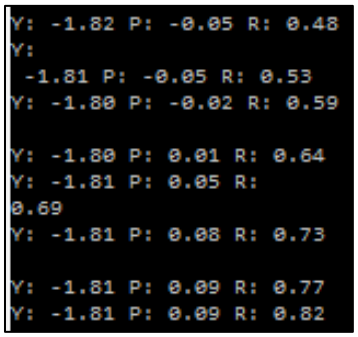



**Figura 35. Datos de orientación leídos en V-REP por medio de comunicación serial con el IMU. Elaborado por el autor en V-REP.**

La ventaja de este *string* es que es más pequeño que el del GPS y únicamente se requiere uno de los valores presentes en él, que es el ángulo de *Yaw*. Por ello, cuando se usa

la función de *string.find( )*, se busca el patrón "Y:", ya que a su lado está el valor que se requiere. De igual forma, una vez segmentada la parte del *string* deseada, esta se convierte a número para poder utilizarlo en los cálculos siguientes.

A forma de resumen, la verificación de datos se realizó para distintas situaciones determinadas, donde lo que varió entre cada una de ellas fue la distancia con respecto a los obstáculos, la orientación del robot y la posición del mismo. El ambiente de prueba fue un pasillo de la escuela de computación, ya que presentaba condiciones muy similares a las que se modelaron en la simulación. Los datos de los sensores se midieron para cada una de esas situaciones específicas y posteriormente se procesó la información de ellos en los programas respectivos. Una vez procesada esta información, se realizan los cálculos respectivos para obtener las señales de actuación sobre el robot.

En la siguiente sección se explican las situaciones que se seleccionaron y el análisis de cada una de ellas.

## 5.3. Análisis de casos

A continuación se presentan tres casos diferentes en los que se varía la distancia a los obstáculos, así como la orientación del robot. Se pretende analizar cómo éste se comportaría en un ambiente real, con los datos reales de los sensores.

Las imágenes que se muestran para cada uno de ellos presentan a la derecha el "*point cloud*" tomado desde el Kinect y a la izquierda el mismo "*point cloud*" luego de ser procesado en Octave. Como ayuda visual, se graficó en la imagen de la derecha un punto rojo en la coordenada en la que se ubica la distancia más cercana la sensor.

Para el primer caso, lo que se quiere analizar es cómo se comportaría el robot con obstáculos muy cercanos a él. En esta situación, una persona se posicionó a medio metro del robot. La distancia entre la posición del robot y el "*waypoint*" es de 5 metros y el ángulo de orientación del robot es de 1,61 radianes (92,24°). Se recuerda que todos los ángulos son referenciados respecto al marco universal.

Para esta situación, se puede observar que la posición del robot con respecto al "*waypoint*" hace que la influencia de la fuerza de atracción sobre el robot sea la máxima, ya que está por encima del valor de 1,5 metros donde su efecto empieza a disminuir. Por otro lado se puede observar que el ángulo de orientación del robot está apuntando casi directamente hacia el "*waypoint*", lo que hace que el ángulo existente entre la orientación del robot y el ángulo de la posición del mismo con respecto a la meta, sea muy pequeño.

A continuación, se muestran los resultados del procesamiento del "*point cloud*" en Octave y seguidamente los resultados de las señales de actuación, luego de procesar la información de todos los sensores en conjunto en V-REP.

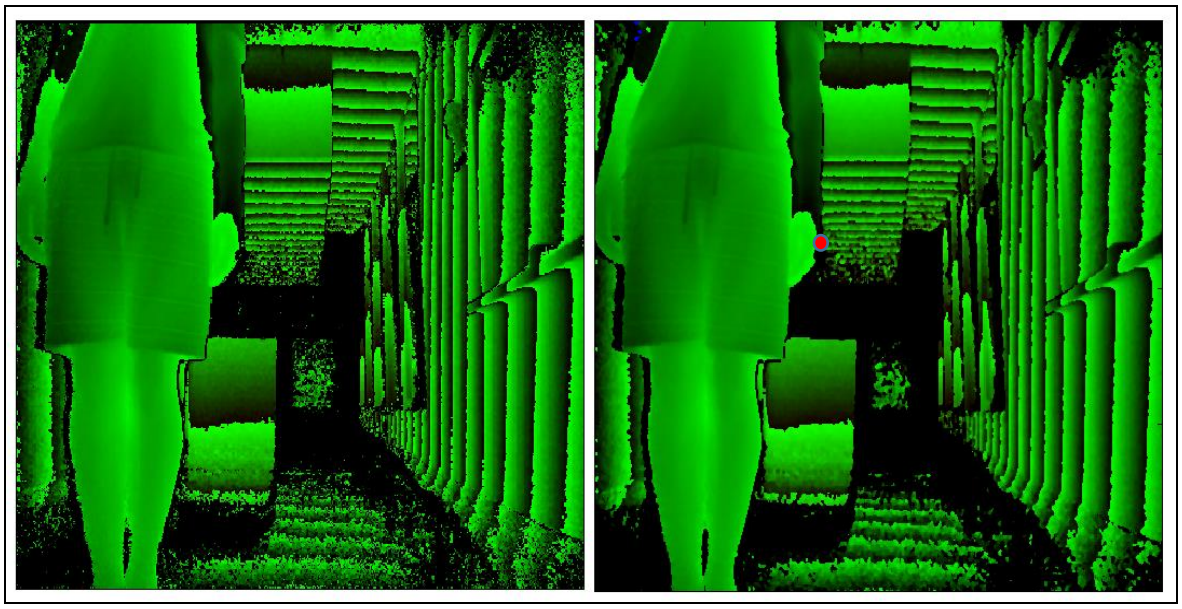


**Figura 36. Primer caso de análisis. Pasillo con obstáculo principal a 0,5 m. Elaborado por el autor en Octave.**

Datos Kinect (Sistema de coordenadas del sensor):

- Coordenada X: 0,043658 m.
- Coordenada Y: 0,043915 m.
- Coordenada Z: 0,50941 m.
- Distancia al punto más cercano: 0,51128 m.

Señales de actuación en el robot (Sistema de coordenadas universal):

- vLeft: 0,68 m/s.
- vRight: 0,56 m/s.
- Ángulo resultante de la trayectoria: 1,45 radianes (83,08°).
- Ángulo entre el objeto y el robot: 1,70 radianes (97,41°)
- Ángulo entre la meta y posición del robot: 1,57 radianes (90°).

En la imagen de la derecha se puede observar que el punto más cercano se detectó en el borde de la mano que está casi en la mitad de la imagen. Lo cual tiene sentido, ya que conforme los puntos se encuentran más hacia los extremos, estos se ubican a una mayor distancia. Por otro lado, la persona también se ubicó muy próxima al robot.

De los datos brindados por el Kinect, se puede ver que la distancia a la que se detectó este punto fue de 0,51128 metros, lo cual concuerda con la distancia a la que se posicionó la persona con respecto al robot. Estos dos datos no son iguales ya que el punto más cercano no coincide con el origen de los ejes de visión del Kinect (centro de la imagen), haciendo que la distancia a la que se encuentra este punto, sea realmente la tangente del ángulo formado por los catetos de la coordenada en Z y la coordenada en X del sistema de coordenadas del Kinect.

De las señales de actuación del robot, se puede observar que los ángulos resultantes tienen valores coherentes y se puede decir que era lo que se esperaba. Por ejemplo, el ángulo entre la posición del robot y la meta está correcto, ya que se colocó el robot en línea recta con la meta, sobre una línea paralela al eje Y. De ahí que su valor sea 1,57 radianes (90°).

Con respecto al ángulo entre el objeto (punto más cercano) y el robot, se podría decir que este dato también es correcto. Esto es debido a que, si se referencia con respecto al sistema de coordenadas del Kinect, lo cual sería restar el ángulo de la orientación del

robot del ángulo entre el objeto y el robot, se obtiene un ángulo de 0,09 radianes (5,16°), el cual para fines de la aplicación es pequeño. Esto tiene sentido, porque como se mencionaba anteriormente, el punto rojo no coincide con el origen de los ejes de visión del Kinect, pero se puede apreciar que aun así está muy cerca de ellos, haciendo que el resultado sea como este.

El ángulo resultante de la trayectoria también presenta un buen resultado, ya que se aleja considerablemente del objeto pero no tanto del ángulo entre la posición del robot y la meta. De igual forma, este ángulo indica que el robot debe girar en sentido horario, lo cual tiene sentido porque hacia ese lado evitaría el obstáculo y se dirigiría hacia la zona del pasillo que está libre. Esto también se puede corroborar en las señales de actuación de velocidad de las ruedas del motor. Al girar más rápido la rueda izquierda que la derecha, el robot girará en este sentido.

También se puede observar que, al detectar un obstáculo tan cercano, la velocidad del robot se redujo casi a la mitad de su valor máximo. Esto se debe a que el obstáculo está ejerciendo una repulsión muy fuerte, cercana al umbral donde es máximo su valor (0,5 m), lo que hace que la velocidad se reduzca. Sin embargo, este valor no es menor porque el "*waypoint*" está ejerciendo la atracción máxima, con lo cual también se corrobora el correcto funcionamiento de los campos potenciales.

El segundo caso presenta las mismas condiciones iniciales que el primero. La única diferencia es que ahora la persona se encuentra a un metro de distancia del robot, por lo que ahora el obstáculo más cercano deja de ser la persona y pasa a ser la baranda de madera en la pared de la derecha. El motivo por el que se hizo así, es para determinar si se están detectando correctamente los puntos más cercanos en el "*point cloud*". De igual forma, se quiere corroborar el comportamiento del robot donde los puntos más cercanos son aquellos en los extremos de la imagen, es decir, aquellos puntos donde se tienen los mayores ángulos posibles entre el robot y el obstáculo. En la siguiente figura se muestra el "*point cloud*" luego de ser procesado.

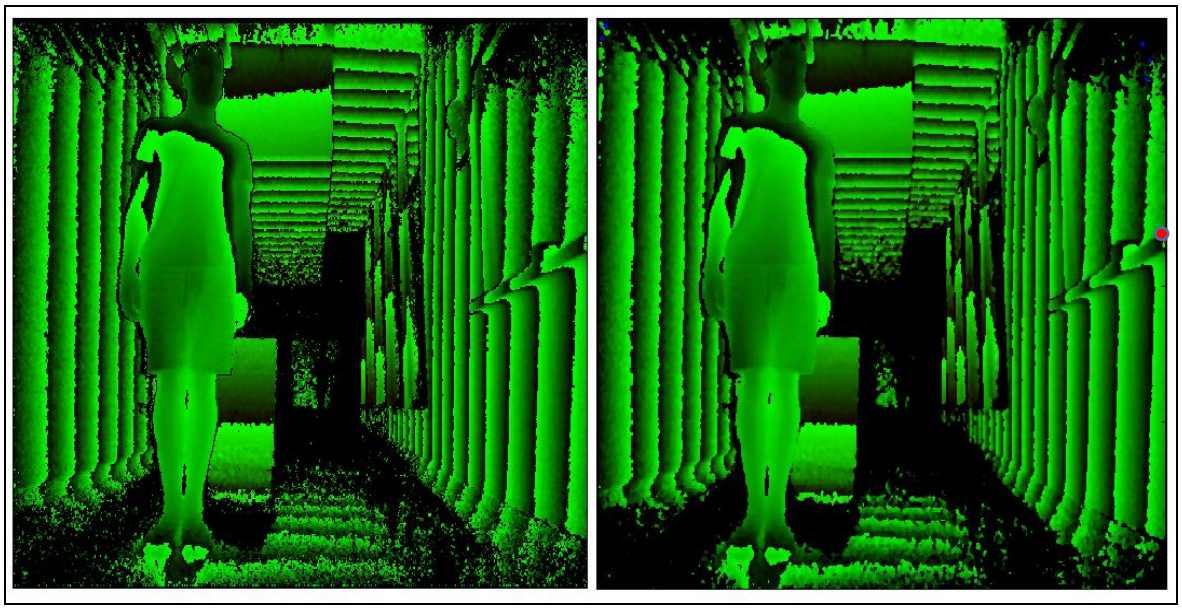


**Figura 37. Segundo caso de análisis. Pasillo con obstáculo más cercano en la pared. Elaborado por el autor en Octave.**

Datos Kinect (Sistema de coordenadas del sensor):

- Coordenada X: -0,50519 m.
- Coordenada Y: 0,085994 m.
- Coordenada Z: 0,61336 m.
- Distancia al punto más cercano: 0,70091 m.

Señales de actuación en el robot (Sistema de coordenadas universal):

- vLeft: 0,84 m/s.
- vRight: 0,95 m/s.
- Ángulo resultante de la trayectoria: 1,75 radianes (100,26°).
- Ángulo entre el objeto y el robot: 1,10 radianes (63,03°).
- Ángulo entre la meta y la posición del robot: 1,57 radianes (90°).

Se puede observar que el punto más cercano se detectó en la región que se esperaba, la cual corresponde a la parte de la baranda de madera que se ve primero en la imagen. Esto hace que el punto se ubique en uno de los extremos de la imagen. La distancia más cercana

detectada tiene sentido, ya que el robot se ubica un poco más cerca de la pared izquierda y también porque la baranda sobresale de la pared unos 10 cm, haciendo que se valor se aproxime a 0,7 metros.

Con las señales de actuación sobre el robot, se pueden ver comportamientos similares a los del caso anterior. Por ejemplo, el ángulo entre el objeto y el robot, concuerda con el hecho de que el punto más cercano se detectó en uno de los extremos de la imagen. Esto es debido a que, si se expresa con referencia al sistema de coordenadas Kinect, lo que es igual a restarle el ángulo anterior al ángulo de orientación del robot, se obtiene un resultado de 0,51 radianes (29,21°), lo cual corresponde a la mitad del máximo ángulo de visión del Kinect en el eje X. El hecho de que el resultado no sea exactamente 28,5° se debe al manejo de las decimas de los radianes.

Al igual que en el caso anterior, el ángulo entre la meta y la posición del robot está correcto ya que la posición entre el robot y la meta se colocó sobre una línea paralela al eje Y. Para el caso del ángulo de la trayectoria, se puede ver que se tiene un buen resultado y a la vez coherente. Esto es debido a que, con respecto a la orientación del robot, el ángulo que se debe alcanzar para llegar a esa trayectoria no es muy grande. De igual forma, este ángulo indica que el robot debe girar en sentido anti horario para llegar a su trayectoria y así ubicarse en una mejor posición para navegar. Este resultado se reafirma con las velocidades de las ruedas, donde la derecha gira más rápido que la izquierda por el sentido de giro que debe seguir.

Por otro lado, si se comparan las velocidades de las ruedas con respecto al ejemplo anterior, se puede volver a reafirmar el correcto funcionamiento de los campos potenciales, ya que para este caso, el robot se encuentra más alejado del obstáculo y el efecto de reducción de la velocidad de las ruedas no es tan grande como ocurrió anteriormente.

Las condiciones para el tercer caso varían con respecto a lo que se venía realizando anteriormente. Ahora se trabaja con una orientación del robot de 2,20 radianes (125,05°). La posición del robot se desplazó unos 20 cm hacia la pared izquierda. Esto se realizó con el fin de corroborar el comportamiento del robot cuando se encuentra bastante próximo a una pared u obstáculo y su orientación más perpendicular a ellos.

A pesar de haber desplazado la posición del robot, este cambio no pudo ser medido por el GPS y la lectura de datos siguió brindando el mismo valor con el que se trabajó en los casos anteriores.

Al igual que en el caso anterior, se puede observar que el punto más próximo al sensor se detectó en uno de los extremos de la imagen. Esto se debe a que no hay otros obstáculos en el ángulo de visión del robot, solo está la pared; y también porque la orientación del robot hace que el sensor apunte más perpendicularmente hacia la pared. Por lo tanto, el punto detectado se encuentra en esa zona.

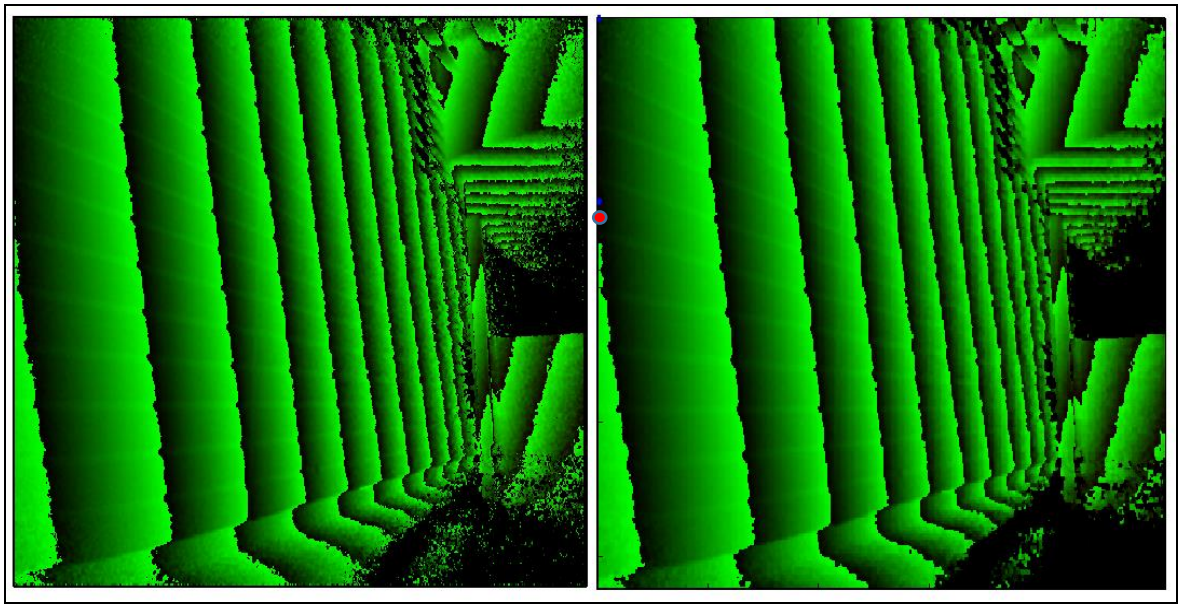


**Figura 38. Tercer caso de análisis. Robot muy cercano a la pared. Elaborado por el autor en Octave.**

Datos Kinect (Sistema de coordenadas del sensor):

- Coordenada X: 0,26140 m.
- Coordenada Y: 0,060011 m.
- Coordenada Z: 0,50941 m.
- Distancia al punto más cercano: 0,57257 m.

Señales de actuación en el robot (Sistema de coordenadas universal):

- vLeft: 1,20 m/s.
- vRight: 0,68 m/s.

- Ángulo resultante de la trayectoria: 1,03 radianes (59,01°).
- Ángulo entre el objeto y el robot: 2,68 radianes (153,55°).
- Ángulo entre la meta y la posición del robot: 1,57 radianes (90°).

De los datos del Kinect se puede observar que la medición de distancia del punto más cercano da un resultado coherente, ya que el robot se encuentra a 70 cm de la pared en línea recta y en combinación con el ángulo de orientación del robot, se debería obtener un valor como el anterior.

De las señales de actuación en el robot, se puede observar que el ángulo entre el robot y la meta se mantiene igual que para los dos casos anteriores. Como se había mencionado anteriormente, la resolución del sensor no fue suficiente para determinar este pequeño cambio por lo que su valor no varía. La ventaja es que a nivel de simulación, este error asociado al GPS se había considerado por lo que no debería dar problemas en los siguientes resultados.

El ángulo entre el robot y el objeto es correcto, ya que corresponde al máximo ángulo en el eje de visión X. Esto se comprueba si se resta la orientación del robot del ángulo anterior, lo cual da como resultado 28,51°.

El ángulo resultante de la trayectoria es muy acertado, ya que como el robot se encuentra muy próximo a la pared y el ángulo con respecto al punto más cercano es muy grande, además de que se debe modificar mucho su orientación para que logre igualar el ángulo generado por el campo de atracción de la meta; se obtiene un valor que lo hace alejarse bastante de la pared, evitando una colisión.

Al igual que en los otros casos, esto se refleja en las velocidad de las ruedas, donde al girar la izquierda más rápido se logra navegar en sentido horario evitando una colisión. También se puede analizar que, al requerir girar un ángulo tan grande, se posee una velocidad de giro mucho mayor que en los casos anteriores, donde la velocidad de la rueda derecha es casi la mitad del valor de la izquierda.

En forma de resumen, se podría decir que los resultados que se obtuvieron en cada caso fueron los esperados. Esto porque se pudo observar cómo se modificó cada señal de actuación en función de la distancia a la que el robot se encuentra de los objetos y de la

orientación y posición del mismo. También se pudo observar que se realizó un correcto procesamiento de los "*point clouds*", siendo todas las mediciones empleadas para ello, provenientes de sensores reales.

# Capítulo 6: Resultados y limitaciones

En los Capítulos 4 y 5 se explicó el trabajo realizado para desarrollar los algoritmos y la aplicación para que el robot móvil pueda manejar de forma autónoma y cumpliendo los requerimientos definidos en el Capítulo 3.

En el Capítulo 4 se mostró el proceso para llegar hasta una solución inteligente, en la que el robot tiene la capacidad de evitar obstáculos, controlar su velocidad y trayectoria en función de la distancia de la que se encuentra de los objetos y llegar hasta coordenadas especificadas por el usuario.

En la primera sección de este capítulo, se presentó el primer algoritmo desarrollado para que el robot evitara obstáculos. Luego de realizar varias simulaciones, se determinó que los rangos definidos para que la velocidad del robot se modificara en función de la distancia a los obstáculos, fueron los más adecuados.

Por ejemplo, utilizar un valor de 0,75 metros para el segundo intervalo (0,75>TDst>0,5) donde la velocidad se comienza a modificar, permite que el robot pueda navegar a máxima velocidad por más tiempo. Por otro lado, también permite que el robot tenga el tiempo suficiente para alejarse de los obstáculos y modificar su trayectoria para evitar colisiones.

El valor de 0,5 metros utilizado para el segundo intervalo (TDst<0,5), permitió que el robot pudiera girar sobre su propio eje sin que colisionara contra algún objeto u

obstáculo mientras lo hacía. Valores menores hubieran implicado que el robot se acercara demasiado a ellos o en efecto los golpeara. Valores mayores hubieran ocasionado que la navegación fuera menos eficiente y se tuviera un menor rango de acción.

La fórmula que se implementó para modificar la velocidad en función de la distancia presentó excelentes resultados. El robot navegaba con trayectorias suaves y no se observó ningún comportamiento oscilatorio a menos que el robot estuviera pasando por un pasaje estrecho como el que se observa en la figura 39, en el cuadro número seis. Parte del recorrido realizado por el robot en esta etapa se presenta en la figura mencionada anteriormente.

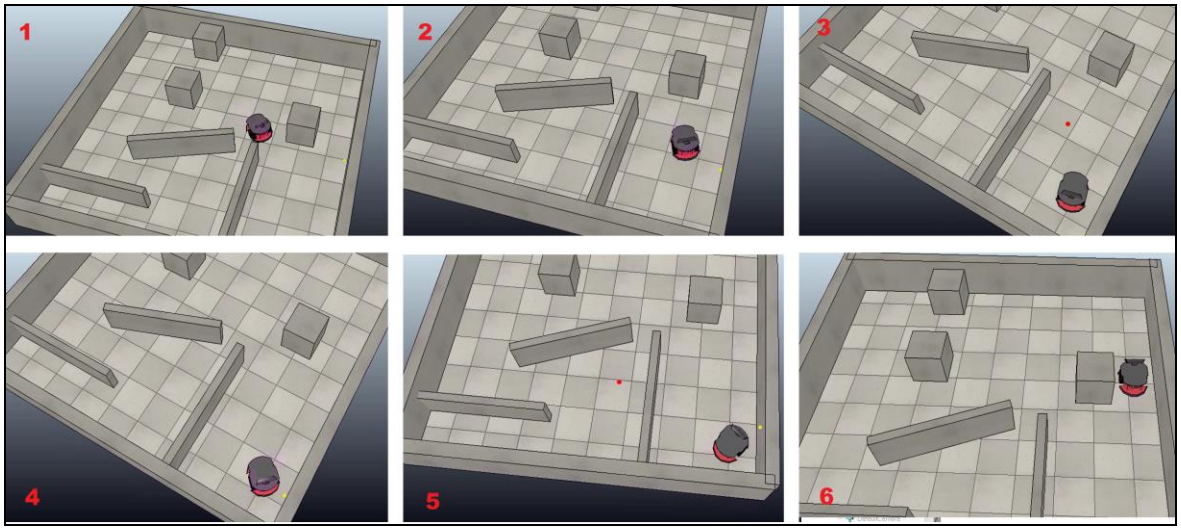



**Figura 39. Parte de la navegación evitando obstáculos realizada por el Pioneer P3-DX. Elaborado por el autor en V-REP.**

Otro punto importante de esta etapa es que la técnica que se utilizó para no quedar atrapado en las esquinas sirvió muy bien. Cuando se llegaba a estos puntos, se pudo observar en la simulación que el robot únicamente actuaba con velocidad angular y giraba sobre su propio eje, hasta que salía de esa situación. Por lo tanto, de esta etapa se puede decir que se obtuvieron los resultados esperados.

En la siguiente etapa, donde se define una trayectoria y se pasa de navegar sin rumbo a tener un objetivo claro, se pudo observar que la implementación de los campos potenciales funcionó perfectamente. Por un lado, las variables que se utilizaron en las

fórmulas permitieron definir rangos adecuados para la atracción y repulsión de las fuerzas imaginarias que actúan sobre el robot.

Por otro lado, su uso permitió obtener un valor proporcional a la velocidad a la que se debe navegar en función de las fuerzas que actúan sobre el vehículo, permitiendo así que se pudiera desarrollar una fórmula de velocidad más eficiente que en la etapa anterior. Esto no solo se debe al hecho de que se tomó en consideración el modelo cinemático del robot, sino también porque dicha fórmula se adaptó para las características de la aplicación y su uso con dicho valor proporcional. Al igual que en la sección anterior, con esta combinación de técnicas, se logró que el robot navegara con trayectorias suaves y que el efecto oscilatorio que pueden generar los campos potenciales se redujera al mínimo.

Con respecto a seguir una trayectoria definida por "*waypoints*", se puede decir que su uso no solo significó un cumplimiento de un requerimiento sino que también significó la reducción de la principal desventaja al utilizar campos potenciales, que son los mínimos locales. Al ubicar estos puntos, definiendo la trayectoria que se debe llevar, se logra girar mejor cuando la orientación de un pasillo cambia, y se logra pasar por puertas o espacios con más facilidad.

Esto se pudo determinar al realizar la simulación descrita en la sección 4.4. Para cada uno de los tramos, se pudo observar cómo el robot se desplazaba hacia el "*waypoint*" determinado, cumpliendo los requerimientos del proyecto. En los pasillos con obstáculos, se pudo observar cómo navegaba el robot hacia su destino, evitándolos y manteniéndose alejado de las paredes. Se pudo observar cómo modificaba su trayectoria y velocidad correctamente en función de las distancias a las que eran detectados los objetos y la distancia a la que se encontraba el robot del punto meta.

También, se pudo observar lo bien que responde la función de velocidad cuando el robot debe girar, ya sea porque el pasillo cambia de orientación o porque debe entrar por una puerta. De igual forma se pudo ver una buena respuesta en situaciones cuando existe una diferencia de casi 180° entre la orientación del robot y el ángulo resultante de la trayectoria.

Esto se debe a que la fórmula de velocidad implementada toma en consideración el ángulo que debe girar el robot, en función de la velocidad máxima que este tiene. También porque su magnitud es proporcional a las fuerzas que actúan sobre el robot. Una parte de la simulación de la navegación final realizada por el robot se puede observar en la siguiente imagen.

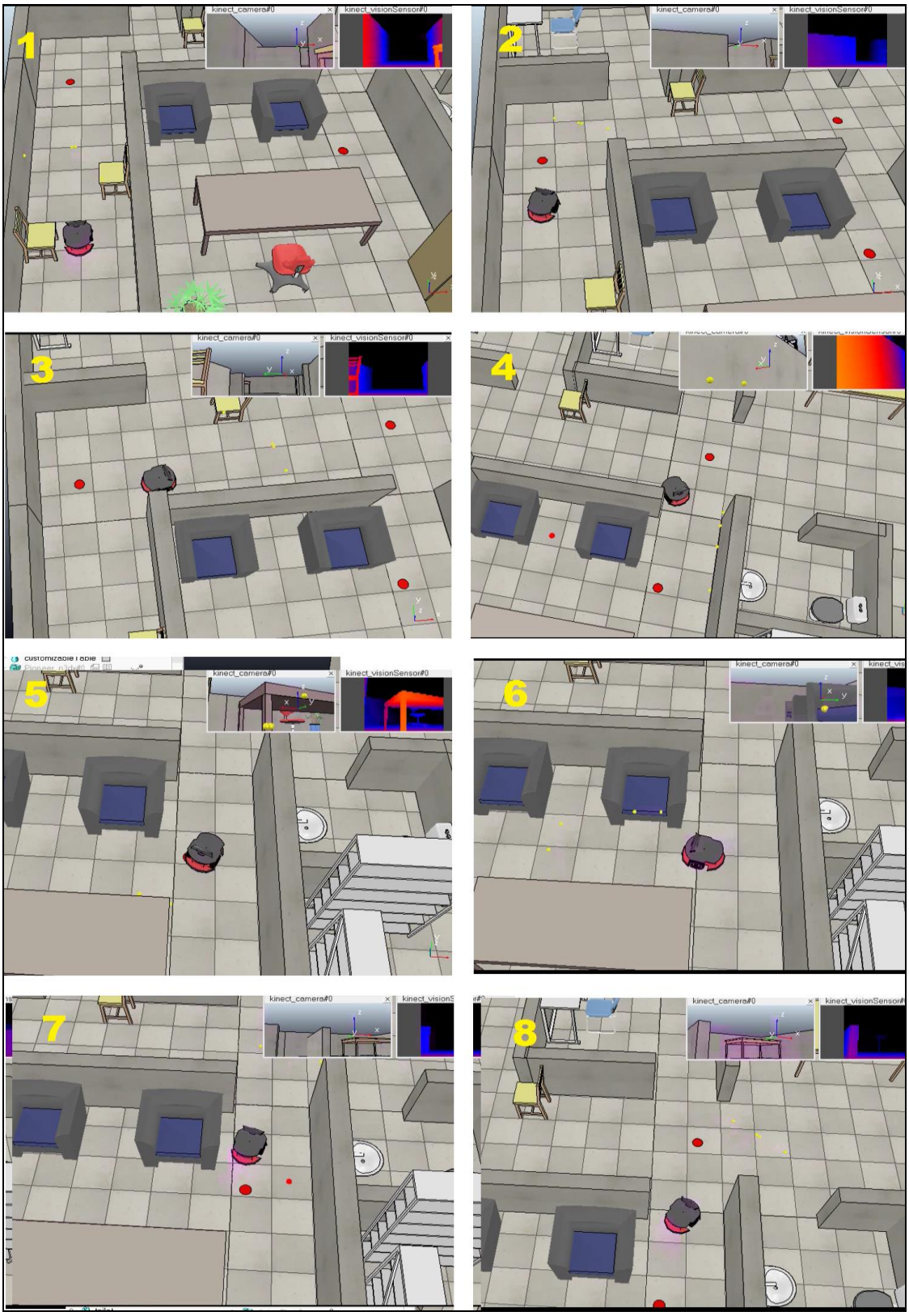



**Figura 40. Parte de la navegación final realizada por el Pioneer P3-DX. Elaborado por el autor en V-REP.**

En el Capítulo 5 ya se mostraron los resultados que se obtuvieron al trabajar con los sensores reales. Para recapitular, se demostró que el comportamiento del robot al trabajar con la información real de los sensores muestra un comportamiento coherente con los datos que se obtuvieron anteriormente en las simulaciones. Con esto, se verifica que los algoritmos desarrollados para procesar la información de ellos sirvieron correctamente, así como los algoritmos para controlar la trayectoria del vehículo.

Otro punto importante es que se pudo profundizar en el manejo del "*point cloud*", ya que a nivel de la simulación, este no presenta el ruido natural del ambiente y que está asociado al sensor. Se logró demostrar que con un pre-procesamiento adecuado de estos datos, se puede reducir bastante el ruido presente en ellos y así realizar un correcto análisis donde se detectan los puntos deseados.

Si bien es cierto los resultados obtenidos fueron los esperados, se pudieron evidenciar algunas limitaciones del proyecto. Uno de ellos fue que debido al ángulo de visión que posee el Kinect, en situaciones muy específicas, el robot puede pasar muy cerca de los objetos o paredes y en algunas ocasiones tocarlos.

Esto ocurre cuando el robot navega cerca de estos elementos y debe realizar un giro. Cuando los objetos eran detectados por el sensor, estos ejercían su influencia ya sea en la fórmula de velocidad utilizada en el primer algoritmo de evitar obstáculos o modificando el campo potencial resultante que actúa sobre el robot al incrementar la repulsión, para la sección de navegación autónoma. Sin embargo, una vez que se deja de detectarlos, ya no se tiene este efecto y no se cuenta con sensores a los lados del robot que permitan determinar la presencia de objetos cercanos fuera del ángulo de visión del Kinect.

Como se menciona, esta es una situación muy específica y no se presenta con frecuencia pero puede ocurrir. Una forma sencilla de eliminar esta limitación sería colocando sensores ultrasónicos en diferentes áreas del robot, donde el ángulo de visión del Kinect no alcanza.

Otra limitación que se tiene, es que se debe tener cierto conocimiento por parte del usuario para saber dónde colocar los "*waypoints*" por los que navegará el robot, y con ello evitar posibles mínimos locales. Dependiendo del tipo de aplicación, se podría utilizar un

manual de usuario donde se explique de forma sencilla cómo colocarlos o con imágenes mostrar cuáles situaciones se deben evitar.

La última limitación que posee el proyecto, es que el GPS funciona muy bien para ambientes exteriores, pero para interiores no trabaja correctamente. Como se había mencionado anteriormente, a nivel de programación, se trabajó con un GPS ya que es el único sensor por medio del cual se podía medir la posición de un objeto, por lo que se constituyó como una limitación a nivel del simulador.

Sin embargo, existen otras aplicaciones para interiores cuyo funcionamiento es muy similar al del GPS, por lo que su escalamiento o acondicionamiento para un ambiente interior no variaría mucho el trabajo realizado en este proyecto.

Con respecto al error de este sensor, como recomendación para mejorar su funcionamiento más que como limitación, se podría utilizar un filtro de Kalman. Si bien es cierto la técnica de promediar los últimos tres datos brindados por el sensor para disminuir el sesgo de las mediciones funcionó bien en la simulación, para trabajar en condiciones reales sería bueno implementar este tipo de filtro, independientemente del tipo de sistema de localización que se utilice.

Una vez que se presentó el análisis de resultados y las limitaciones del proyecto, se pueden obtener las conclusiones del mismo y las recomendaciones, en caso de realizarse trabajos futuros o llegar a implementar la aplicación para resolver algún problema específico. Esta información se presenta en el próximo capítulo.

# Capítulo 7: Conclusiones y recomendaciones

## 7.1. Conclusiones

- Se procesó adecuadamente la información ("*point cloud*") generada por el sensor 3D, que para el caso de este de proyecto fue un Kinect, tanto a nivel de simulación, como a nivel real con los datos del sensor en físico.

- Se diseñaron algoritmos de navegación basados en la información procesada que brinda el Kinect, con los cuales se logró que el robot navegará exitosamente evitando obstáculos y llegando a una coordenada específica.

- Se simuló correctamente la aplicación diseñada, en el programa V-REP, donde los algoritmos desarrollados se probaron en distintos ambientes modelados, demostrando así la validez y efectividad de los mismos.

- Se diseñó una aplicación a nivel de software por medio de la cual se puede leer y procesar adecuadamente la información de los sensores en físico.

- Se logró implementar el sistema, por medio de la aplicación diseñada y los sensores en físico y con ello corroborar las señales de actuación sobre el robot.

## 7.2. Recomendaciones

- En caso de implementar la aplicación para un vehículo que trabajará en ambientes internos, buscar una alternativa al GPS que funcione en este tipo de ambientes.

- En caso de implementar la aplicación para tareas donde se requiere una mayor precisión con las coordenadas a las que se desea llegar, agregar al sistema un filtro de Kalman para mitigar el error asociado a la señal de ubicación.

- Analizar bien el espacio donde se utilizará la aplicación, para colocar correctamente los "*waypoints*" y disminuir el efecto de los mínimos locales en caso de que existan.

- Considerar el uso de sensores ultrasónicos o el que se desee, en las áreas del robot donde no alcanza el ángulo de visón del Kinect, para situaciones en donde se debe evitar a toda costa que el robot pueda tocar paredes u objetos.

# Referencias


*28IM*. (2015, septiembre). Retrieved noviembre 14, 2015, from http://www.28im.com/android/a719726.html

Adept MobileRobots. (2015). *Adept MobileRobots.* Retrieved Octubre 06, 2015, from Pioneer P3-DX: http://www.mobilerobots.com/ResearchRobots/PioneerP3DX.aspx

Aimagin. (2011, febrero 24). *About GPS and FV-M8 GPS Module*. Retrieved noviembre 26, 2015, from http://www.aimagin.com/learn/index.php?title=About_GPS_and_FV-M8_GPS_Module

Baddeley, G. (2001, julio 20). *GPS-NMEA sentence information*. Retrieved noviembre 26, 2015, from http://aprs.gids.nl/nmea/

Bañó, A. (2003, junio). *Análisis y diseño del control de posición de un robot móvil con tracción diferencial.* Retrieved noviembre 16, 2015, from http://deeea.urv.cat/public/PROPOSTES/pub/pdf/333pub.pdf

Bouchier, P. (2014, octubre 23). *GitHub*. Retrieved noviembre 26, 2015, from Building an AHRS using the SparkFun "9DOF Razor IMU" or "9DOF Sensor Stick": https://github.com/ptrbrtz/razor-9dof-ahrs/wiki/Tutorial

Braga, M. (2014, octubre 21). *Tested*. Retrieved septiembre 28, 2015, from Research Robots Versus the Volcano: http://www.tested.com/science/466283-research-robots-versus-volcano/

Braitenberg, V. (1986). *Vehicles: Experiments in Synthetic Psychology.* Cambridge: MIT Press.

Carnegie Mellon University. (2015, Septiembre). *The Robotics Institute*. Retrieved Septiembre 20, 2015, from https://www.ri.cmu.edu/research_project_detail.html?project_id=163&menu_id=261

Carvalho, M. (2004, Enero 01). *MundoGeo*. Retrieved noviembre 20, 2015, from GPS 21: http://mundogeo.com/blog/2004/01/01/gps-21-11/

Coppelia Robotics. (2015, Septiembre 26). *v-rep*. Retrieved Septiembre 26, 2015, from http://www.coppeliarobotics.com/

Cyberbotics. (2015, Septiembre 25). *Webots*. Retrieved Septiembre 26, 2015, from https://www.cyberbotics.com/webots/

DARPA. (2014, Febrero). *DARPA Urban Challenge*. Retrieved Septiembre 21, 2015, from http://archive.darpa.mil/grandchallenge04/

DARPA. (2014, Febrero). *Grand DARPA Challenge*. Retrieved Septiembre 21, 2015, from http://archive.darpa.mil/grandchallenge04/

DePriest, D. (2015, octubre 7). *NMEA data*. Retrieved noviembre 26, 2015, from http://www.gpsinformation.org/dale/nmea.htm

Duarte, G. (2013, Octubre 16). *BetaPlay*. Retrieved Noviembre 20, 2015, from http://www.betaplay.com.br/microsoft-vai-aumentar-poder-do-xbox-one-com-a-nuvem/

Dudek, G., & Michael, J. (2010). *Computational Principles of Mobile Robotics.* New York, Estados Unidos: Cambridge University Press.

Gasperi, M. (2015, junio 28). *http://www.extremenxt.com*. Retrieved septiembre 28, 2015, from Machina Speculatrix: http://www.extremenxt.com/walter.htm

Gnecco, P. (2012, marzo 4). *Youtube*. Retrieved septiembre 28, 2015, from first point cloud.kinect: https://www.youtube.com/watch?v=s3wl2y9ANDs

Goodrich, M. (2008, mayo 12). *Potential Fields Tutorial.* Retrieved from http://phoenix.goucher.edu/~jillz/cs325_robotics/goodrich_potential_fields.pdf

Google. (2015, Julio). *Google Self;Driving Car Project*. Retrieved Septiembre 21, 2015, from http://www.google.com/selfdrivingcar/how/

Goss, R. (2015, Mayo 1). *arxterra*. Retrieved septiembre 28, 2015, from Selection of MPU 6050: http://www.arxterra.com/selection-of-mp6050/

Grupo de Investigación en Robótica Autónoma. (2015, julio 15). *GIRA*. Retrieved octubre 06, 2015, from Conociendo coordenadas realies con Kinect: http://tecnodacta.com.ar/gira/2015/07/conociendo-coordenadas-reales-con-kinect/

Higgins, S. (2015, junio). *SPAR Point Group*. Retrieved septiembre 28, 2015, from Velodyne Announces $7,999 "Puck" LiDAR Sensor: http://www.sparpointgroup.com/news/vol12no37-velodyne-announces-puck-lidar-sensor

Horaud, R. (2014, Abril). *Three Dimensional Sensors.* Retrieved Septiembre 23, 2015, from http://perception.inrialpes.fr/~Horaud/Courses/3DS_2013.html

Inman, M. (2015, Septiembre). *The Oatmel*. Retrieved septiembre 28, 2015, from 6 things I learned from riding in a Google Self-driving car: http://theoatmeal.com/blog/google_self_driving_car

iRobot. (2015). *iRobot.* Retrieved Septiembre 20, 2015, from http://www.irobot.com/images/consumer/cs/Roomba-owners-manual.pdf

Laboratório de Robótica Móvel. (2015, Agosto 24). *Laboratório de Robótica Móvel*. Retrieved Septiembre 18, 2015, from http://www.lrm.icmc.usp.br/web/index.php?n=Port.Home

Laboratório de Robótica Móvel. (2015, Julio 18). *Laboratório de Robótica Móvel*. Retrieved 21 Septiembre, 2015, from Projeto CaRINA 2: http://www.lrm.icmc.usp.br/web/index.php?n=Port.ProjCarina2Info

Laboratório de Robótica Móvel. (2015, Octubre 06). *Laboratório de Robótica Móvel*. Retrieved Octubre 06, 2015, from Caminhão Autônomo: http://www.lrm.icmc.usp.br/web/index.php?n=Port.ProjSTruck

Latombe, J.-C. (1991). *Robot Motion Planning.* New York: Springer .

LiDAR UK. (2015). *LiDAR UK*. Retrieved Septiembre 23, 2015, from What is LiDAR?: http://www.lidar-uk.com/what-is-lidar/

Mihai, D. (2015, Julio 26). *smashingrobotics.* Retrieved septiembre 28, 2015, from Telepresence Robots Reviewed – Part 2: http://www.smashingrobotics.com/telepresence-robots-reviewed-part-2/

Montserrat, M. (2015, Enero 28). *Kinect for developers*. Retrieved noviembre 20, 2015, from http://www.kinectfordevelopers.com/es/2014/01/28/caracteristicas-kinect-2/

NASA. (2015, Septiembre). *NASA.* Retrieved Septiembre 20, 2015, from Programs & Missions: http://www.tested.com/science/466283-research-robots-versus-volcano/)

National Oceanic and Atmospheric Administration. (2015, Mayo 29). *National Ocean Service*. Retrieved Septiembre 23, 2015, from What is LIDAR?: http://oceanservice.noaa.gov/facts/lidar.html

Nehmzow, U. (2003). *Mobile Robotics: A Practical Introduction.* Londres: Springer.

Ollero, A. (2001). *Robótica: manipuladores y robots móviles.* Barcelona: Marcombo S.A.

Onwubolu, G. (2005). *Mechatronics: Principles and Applications.* Oxford: Butterworth-Heinemann.

Patranabis. (2003). *Sensors and Transducers.* PHI Learning.

Player. (2014, Febrero 2014). *The Player Project*. Retrieved Septiembre 26, 2015, from http://playerstage.sourceforge.net/

PointCloud.org. (2015, Septiembre 14). *pcl.* Retrieved Septiembre 23, 2015, from What is PCL?: http://pointclouds.org/about/

Reis, O. (1996). *Waypoints e Navegação de Grande Círculo*. Retrieved Septiembre 25, 2015, from http://www.tecepe.com.br/nav/nav_c3.htm

ROS. (2015, Septiembre 20). *ROS*. Retrieved Septiembre 25, 2015, from About ROS: http://www.ros.org/about-ros/

Salvatore, J., Osio, J., & Morales, M. (2014, julio 22). *Latin American and Caribbean Consortium of Engineering Institution.* Retrieved Septiembre 23, 2015, from http://www.laccei.org/LACCEI2014-Guayaquil/RefereedPapers/RP178.pdf

Siegwart, R., & Nourbakhsh, I. (2004). *Intrduction to Autonomous Mobile Robots.* Cambridge: MIT Press.

Sociedade Brasilera de Computação. (2015). *Anais da 34a Jornada de Actualização em Informática JAI 2015.* Recife: Sociedade Brasilera de Computação.

Sparkfun. (2015, Noviembre 13). *Sparkfun*. Retrieved Noviembre 20, 2015, from 9 Degrees of Freedom - Razor IMU: https://www.sparkfun.com/products/10736

Sunsky. (2015, Noviembre 13). *USB GPS Receiver XC-GD75, Working on Laptop, Desktop Computer(Dark Blue)*. Retrieved Noviembre 20, 2015, from http://www.sunsky-online.com/view/19265.htm

Universidade de São Paulo. (2015). *Universidade de São Paulo*. Retrieved Septiembre 18, 2015, from http://www5.usp.br/institucional/a-usp/historia/

University of Maryland. (2004, noviembre 2004). *Space System Labs*. Retrieved Septiembre 23, 2015, from Inertial Measurement Unit (IMU): http://www.ssl.umd.edu/projects/RangerNBV/thesis/2-4-1.htm

Weber, H. (2014, Junio 13). *Venture Beat*. Retrieved Septiembre 22, 2015, from Find out if Google's driverless car is legal in your state: http://venturebeat.com/2014/06/13/find-out-if-googles-driverless-car-is-legal-in-your-state/

# Apéndices

## Apéndice A.1. Glosario

Para facilitar la comprensión del texto, se presenta la siguiente sección con un listado de los principales conceptos que se utilizaron.

- **Campos Potenciales**: teoría utilizada en navegación autónoma, donde se estipula que el robot se puede modelar como una partícula que se ve influenciada por fuerzas de atracción generadas por el punto de destino y fuerzas de repulsión generadas por los obstáculos, esto con el fin de determinar la trayectoria a seguir.

- **IMU**: unidad de medición inercial que integra tres sensores con los cuales puede medir el ángulo de giro, la aceleración y las fuerzas que actúan en un eje determinado.

- **GPS**: sistema de posicionamiento global, que por medio de un receptor puede leer la información de la posición, velocidad y tiempo de una coordenada en el planeta. Dicha información se determina por medio de la triangulación de tres satélites de un sistema de 24 que giran alrededor del planeta.

- **Kinect**: sensor desarrollado por la empresa Microsoft, el cual posee una cámara RGB y una cámara infrarroja por medio de las cuales puede determinar la profundidad de distintos puntos del ambiente en el que se utilice y así generar un "*point cloud*".

- **Pioneer P3-DX**: robot móvil de tracción diferencial desarrollado por la empresa Adept. Es uno de los robots más utilizados en el área de investigación debido a su versatilidad y prestaciones. Es el robot con el que se trabaja en la parte de simulación de este proyecto.

- ***Point cloud***: estructura de datos que permite representar colecciones de puntos multidimensionales, los cuales se usan generalmente para representar información en tres dimensiones.

- **Robótica móvil**: área de la robótica que se encarga del estudio y desarrollo de robots que tienen la libertad de moverse en el campo de trabajo. En otras palabras, se encarga de todos aquellos robots que no poseen una base fija en un punto.

- **Script**: es un pequeño programa que posee código desarrollado específicamente para un elemento o función.

- **V-REP**: ambiente de simulación especial para aplicaciones en robótica, donde cada objeto o modelo puede ser controlado individualmente por medio de un *script* propio. Es el simulador utilizado en este proyecto.

- **Vehículo autónomo**: vehículo que tiene la capacidad de navegar de forma independiente en un ambiente determinado.

- **Vehículo de Braitenberg**: concepto desarrollado por Valentino Braitenberg en 1984, para describir un conjunto de vehículos con comportamientos totalmente reactivos, donde su actuación era completamente proporcional a la señal de estímulo medida.

- ***Waypoints***: método de navegación muy popular que utiliza una serie de puntos o coordenadas que definen una trayectoria de navegación.

- ***Yaw***: es un tipo de ángulo de Euler usado en navegación. Permite describir la orientación de un objeto alrededor del eje Z.

## Apéndice A.2. Otras aplicaciones con campos potenciales

En el Capítulo 4 se presentó una aplicación con campos potenciales, en la que el usuario define una ruta por medio de "*waypoints*" para que posteriormente sea ejecutada por el robot móvil. En esta sección, se presenta otra aplicación que se puede implementar utilizando los mismos algoritmos descritos en el Capítulo 4.

En este caso, la aplicación que se simuló fue la de seguir una persona. Para llevar esto a cabo, lo que se debe realizar es colocar un GPS en el usuario para que determine su ubicación. A nivel práctico, esto ni siquiera se debe realizar porque actualmente la gran mayoría de teléfonos móviles cuentan con un localizador que usa esta tecnología.

La diferencia con la simulación anterior es que ya no se requieren guardar las coordenadas de los puntos por los que se desea que pase el robot, sino que se realizan los cálculos de todos los algoritmos con la posición meta definida por las coordenadas de la persona, las cuales se estarán actualizando constantemente.

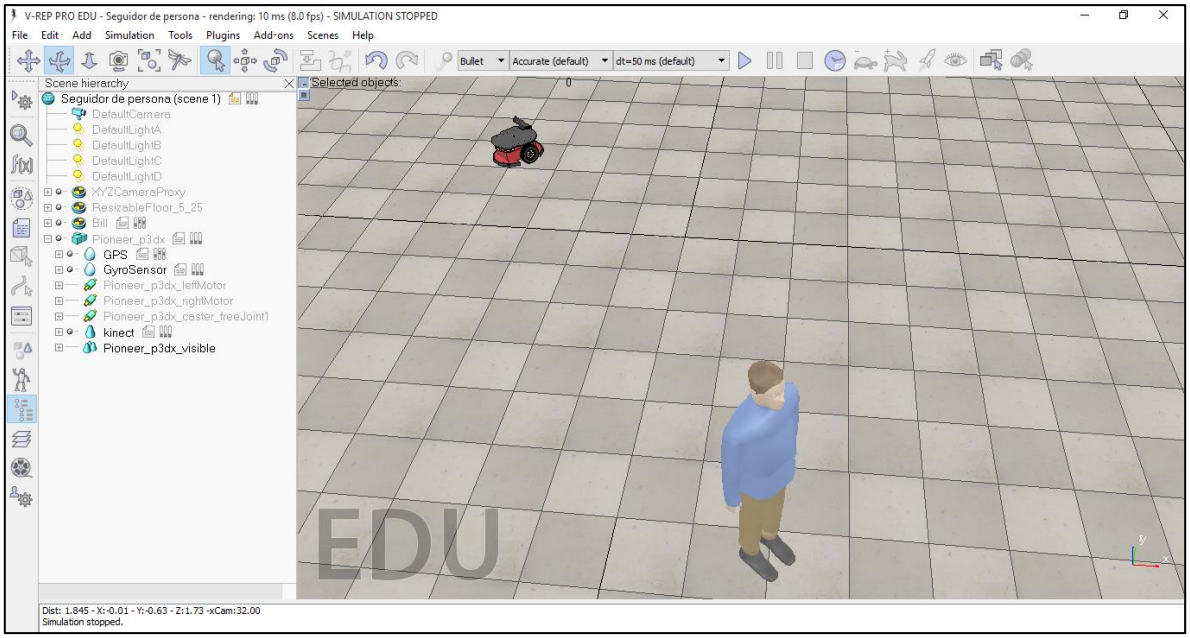



**Figura A.2.1. Ambiente de prueba para la aplicación de seguidor de persona. Elaborado por el autor en V-REP.**

En caso de que la velocidad de la persona sea mucho mayor que la velocidad máxima del robot, se puede recurrir al algoritmo elaborado para la aplicación anterior donde se usan "*waypoints*". Esto se debe a que, si el usuario se desplaza muy rápido, no se tendrá una noción muy clara del camino que siguió y esto podría provocar que el robot quede atrapado en un mínimo local.

El método para utilizar "*waypoints*" funcionaría de la siguiente forma. El vector donde se guardan las coordenadas de la ubicación de la persona funcionaría como una cola. A esta cola entrarían las coordenadas actualizadas de la ubicación cada cierto intervalo transcurrido, el cual estaría definido en función de la velocidad del usuario. Al realizar esto, se define una mejor y más suave trayectoria, haciendo que para el robot sea más fácil seguir a la persona.

A nivel de simulación, únicamente se trabajó en un ambiente muy sencillo, donde la velocidad de la persona no era muy elevada. Por ello, únicamente se probó la primera situación posible. Los resultados que se obtuvieron fueron sumamente convincentes, ya que el robot seguía a la persona con mucha facilidad y se orientaba hacia ella rápidamente. Conforme se iba acercando a ella, se veía cómo su velocidad se reducía. Lo cual demuestra de nuevo el correcto funcionamiento de los campos potenciales.

Con este ejemplo, se quiere mostrar la versatilidad que tiene el programa desarrollado para el proyecto, así como las distintas aplicaciones que se le pueden dar. También muestra la posibilidad de seguir trabajando en uno u otro enfoque para trabajos futuros.

# Anexos

## Anexo B.1. Hoja de datos del Pioneer P3-DX

adept mobilerobots

# Pioneer 3-DX

Pioneer 3-DX is a small lightweight two-wheel two-motor differential drive robot ideal for indoor laboratory or classroom use. The robot comes complete with front SONAR, one battery, wheel encoders, a microcontroller with ARCOS firmware, and the Pioneer SDK advanced mobile robotics software development package.

Pioneer research robots are the world's most popular intelligent mobile robots for education and research. Their versatility, reliability and durability have made them the preferred platform for advanced intelligent robotics. Pioneers are pre-assembled, customizable, upgradeable, and rugged enough to last through years of laboratory and classroom use.

### Product Features and Benefits

- **Easy to Use** - Comes assembled and integrated with its accessory packages.
- **Reliable** - Construction is durable and rugged. Easily handles the small gaps, minor bumping, jarring, or other obstacles that hinder other robotic platforms. Some Pioneer robots have been in service for over 15 years.
- **Pioneer Software Development Kit** - All Adept MobileRobots platforms include Pioneer SDK, a complete set of robotics applications and libraries that accelerate the development of robotics projects. Pioneer SDK is backed by our product support team.
- **Customizable** - Easily accessorize by choosing from dozens of supported and tested accessories that integrate with the robotic platform. Additional help is available for future upgrades or added accessories.
- **Reference Platform** - Pioneer robots are a standard in intelligent mobile platforms. Search your preferred robotics journal or conference listings to find many examples of Pioneer platforms in research applications.
- **Technical Support** - Pioneer software and hardware comes fully documented with additional help available through our product support team.

### Specifications

**Construction**
- Body: 1.6 mm aluminum (powder-coated)
- Tires: Foam-filled rubber

**Operation**
- Robot Weight: 9 kg
- Operating Payload: 17 kg

**Differential Drive Movement**
- Turn Radius: 0 cm
- Swing Radius: 26.7 cm
- Max. Forward/Backward Speed: 1.2 m/s
- Rotation Speed: 300°/s
- Max. Traversable Step: 2.5 cm
- Max. Traversable Gap: 5 cm
- Max. Traversable Grade: 25%
- Traversable Terrain: Indoor, wheelchair accessible

**Power**
- Run Time: 8-10 hours w/3 batteries (with no accessories)
- Charge Time: 12 hours (standard) or 2.4 hrs (optional high-capacity charger)
- Available Power Supplies:
  - 5 V @ 1.5 A switched
  - 12 V @ 2.5 A switched

**Batteries**
- Supports up to 3 at a time
- Voltage: 12 V
- Capacity: 7.2 Ah (each)
- Chemistry: lead acid
- Hot-swappable Batteries: Yes

**Available Recharge Options:**
- Direct plug-in
- Docking station
- Powercube (3-battery charging bay)

* Batteries are accessible through hinged latched access panel for hot-swapping (continuous operation)

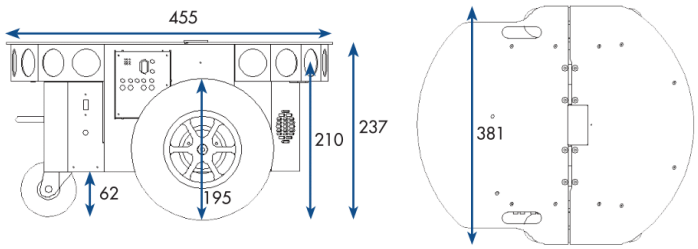
**Microcontroller I/O**
- System Serial
- 32 digital inputs
- 8 digital outputs
- 7 analog inputs
- 3 serial expansion ports

*Some ports may not be available if certain accessories are included with the robot

**User Control Panel**
- MIDI programmable piezo buzzer
- Main power indicator
- Battery charge indicator
- 2 AUX power switches
- System reset
- Motor enable pushbutton

# Pioneer 3-DX

### Dimensions (mm)




### Core Software - included with all research platforms

**ARIA** provides a framework for controlling and receiving data from all MobileRobots platforms, as well as most accessories. Includes open source infrastructures and utilities useful for writing robot control software, support for network sockets, and an extensible framework for client-server network programming.

**MobileSim** open-source simulator which includes all MobileRobots platforms and many accessories.

**MobileEyes** graphical user interface client for remote operation and monitoring of the robot.

**Mapper 3-Basic** tool for creating and editing map files for use with ARIA, MobileSim, and navigation software.

**SONARNL** provides sonar-based approximate localization and navigation.

### Accessory Support Software - bundled with purchase of robotic accessory

**ARNL** enables robust, laser-based autonomous localization and navigation.

**Robotic Arm Support** Pioneer arms are packaged with integrated software support.

**Speech Recognition and Synthesis Library:** Easy-to-use C++ development library for speech recognition based on the open source Sphinx2 system. Speech synthesis (text-to-speech) based on Cepstral synthesizer.

**ACTS Color Tracking System:** Software application which reads images from a camera and tracks the positions and sizes of multiple color regions. Information can be incorporated into your own software via ARIA.

### Optional Industrial Grade Internally Mounted Computers

| Mamba EBX-37 (Dual Core 2.26 GHz - 2-8 GB RAM) |
|---|
| 6 X USB2.0 Ports |
| 2 X PC/104+ Slots |
| 4 X RS-232 Serial Ports |
| 2 X 10/100/1000 Ethernet Ports |
| Onboard Audio & Video |
| Solid State Drive |
| Optional Wireless Ethernet |

### Optional Accessories:

- Laser-range finders
- Mono- and stereo-vision cameras
- Rear SONAR
- Wireless serial to Ethernet for remote operation
- Robotic arms and grippers
- Gyroscope
- Segmented bumper arrays
- Speakers and microphones
- Joystick
- Many more...

**Include our integrated & supported accessories with your Pioneer 3-DX.**

Here are some popular configurations to choose from:

Mapping & Vision

Gripping & Manipulation

Audio & Speech

### More Information:

See our website www.mobilerobots.com for a full range of supported accessories or contact our sales department to discuss your application.

Adept Technology, Inc. 10 Columbia Drive, Amherst, NH 03031
Tel: 603-881-7960 Email: sales@mobilerobots.com
www.mobilerobots.com

Specifications subject to change without notice.



09366-P3DX Rev. A

(Adept MobileRobots, 2015)

## Anexo B.2. Conexión del 9DOF Razor IMU con el "FTDI Basic Breakout"

La conexión entre estos dos componentes es muy sencilla realmente. El primer paso que se debe realizar es soldar adaptadores para cables en los conectores del "FTDI" de la placa del Razor.

Una vez que se suelda esa pieza, se conectan los pines de las dos placas de la forma en que se muestra en la siguiente figura. (Bouchier, 2014)

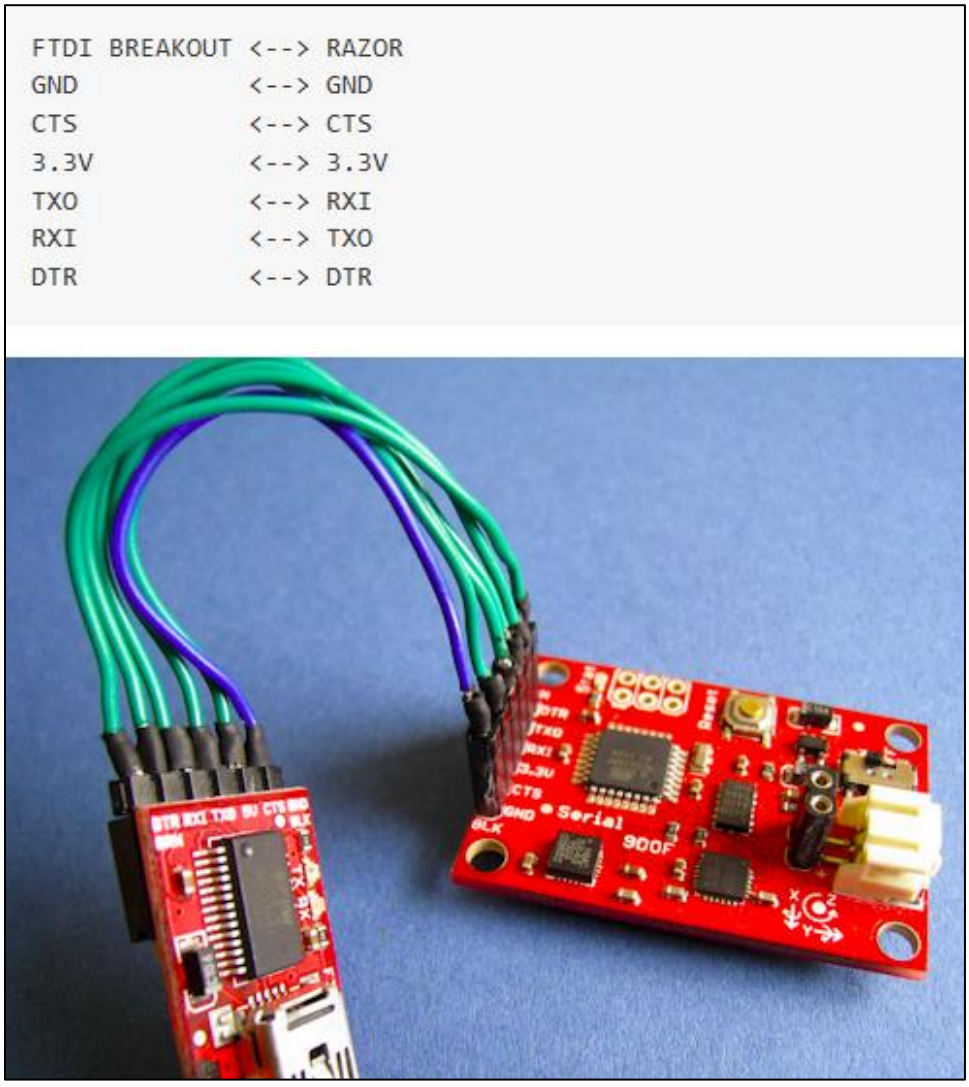



**Figura B.2.1. Conexión del 9DOF Razor IMU con el "FTDI Basic Breakout". Recuperado de: (Bouchier, 2014)**

## Anexo B.3. Padrón NMEA

En el Capítulo 5 se explicó que el GPS trabaja con un padrón universal para enviar su información. Este padrón fue desarrollado para definir la interfaz entre varias piezas de equipamiento electrónico marino. Este estándar permite a esos dispositivos electrónicos enviar información a computadores y otros elementos.

La comunicación del receptor GPS está definida con esta especificación NMEA. La mayoría de los programas de computadora que proveen información de posición en tiempo real, entienden y esperan que los datos se encuentren en este formato. Estos datos incluyen la solución procesada completa PVT (posición, velocidad, tiempo) para el receptor GPS. La idea de este padrón es enviar una línea de datos, llamada sentencia, la cual es independiente de otras. Hay sentencias estándar para cada categoría de dispositivo.

Todas las sentencias estándar poseen un prefijo de dos letras que define el tipo de dispositivo que las está empleando (para GPS el prefijo es GP). Esto viene seguido de una secuencia de tres letras que define el contenido de la misma. Este formato también brinda la posibilidad de definir sentencias propias de los fabricantes de hardware. Estas sentencias propias se inician con el prefijo P y son seguidos de tres letras que identifican su contenido.

Cada sentencia comienza con el símbolo $ y no puede ser mayor a 80 caracteres de texto visible (más los finales de línea). La información se almacena en una única línea donde los diferentes elementos de información están separados por una coma. Los datos son simplemente texto ASCII y se puede extender por múltiples sentencias en ciertas instancias especializadas, pero normalmente se contiene en una única sentencia de longitud variable. (DePriest, 2015)

A continuación se muestra una figura con los diferentes tipos de sufijos y su significado, para los dispositivos de prefijo GP, es decir GPS.

```
$GPBOD - Bearing, origin to destination
$GPBWC - Bearing and distance to waypoint, great circle
$GPGGA - Global Positioning System Fix Data
$GPGLL - Geographic position, latitude / longitude
$GPGSA - GPS DOP and active satellites
$GPGSV - GPS Satellites in view
$GPHDT - Heading, True
$GPR00 - List of waypoints in currently active route
$GPRMA - Recommended minimum specific Loran-C data
$GPRMB - Recommended minimum navigation info
$GPRMC - Recommended minimum specific GPS/Transit data
$GPRTE - Routes
$GPTRF - Transit Fix Data
$GPSTN - Multiple Data ID
$GPVBW - Dual Ground / Water Speed
$GPVTG - Track made good and ground speed
$GPWPL - Waypoint location
$GPXTE - Cross-track error, Measured
$GPZDA - Date & Time
```

**Figura B.3.1. Diferentes formatos NMEA para GPS y su significado. Recuperado de: (Baddeley, 2001)**

También es importante entender el formato GGA, que fue con el que se trabajó para efectos de la implementación con los sensores en físico. La descripción de cada uno de los componentes para este tipo de sentencia se presenta en la siguiente figura.

**$GPGGA,gga1,gga2,gga3,gga4,gga5,gga6,gga7,gga8,gga9,gga10,gga11,gga12,gga13,gga14*hh<CR><LF>**

| Parameters | Descriptions | Notes |
|---|---|---|
| gga1 | UTC time as position is fixed | hhmmss.sss: hh – hour; mm – minute; ss.sss – second |
| gga2 | Latitude | ddmm.mmmmmm: dd – degree; mm.mmmmmm – minute (0° ~ 90°) |
| gga3 | Latitude sector | N – North; S - South |
| gga4 | Longitude | dddmm.mmmmmm: dd – degree; mm.mmmmmm – minute (0° ~ 180°) |
| gga5 | Longitude sector | E – East; W - West |
| gga6 | GPS quality indicator | 0 – No fixed or invalid position<br>1 – SPS Position available<br>2 – Differential GPS (SPS) |
| gga7 | Number of SVs used in position estimation | xx: 00 ~ 12 |
| gga8 | HDOP | xx.xx: 00.00 ~ 99.99 |
| gga9 | Altitude above mean sea level (geoid) | xx.xxx: 00.000 ~ 99.999 |
| gga10 | Unit for Altitude | M: meter |
| gga11 | Geoidal separation | |
| gga12 | Unit for geoidal separation | M: meter |
| gga13 | Age of differential corrections | unit : second; null when DGPS is not used |
| gga14 | Reference station ID (DGPS) | xxxx: 0000 ~ 1023 |
| hh | Checksum | hex number (2 – character) |
| <CR><LF> | End of message | |

**Figura B.3.2. Componentes del formato GGA y su significado. Recuperado de: (Aimagin, 2011)**