INSTITUTO POLITÉCNICO NACIONAL

UNIDAD PROFESIONAL INTERDISCIPLINARIA EN
INGENIERÍA Y TECNOLOGÍAS AVANZADAS

Trabajo Terminal II

"Coordinación de dos robots móviles terrestres bajo un esquema líder seguidor"

Que para obtener el título de

"Ingeniero en Mecatrónica"

Presentan:

Mauricio Sánchez Ortega

David Alejandro Toro Sandoval

Luis Fernando Zarazua Aguilar

Asesores:

Dr. Juan Luis Mata Machuca

M. en C. Mauricio Méndez Martínez

Mayo 2019

INSTITUTO POLITÉCNICO NACIONAL

UNIDAD PROFESIONAL INTERDISCIPLINARIA EN INGENIERÍA Y TECNOLOGÍAS AVANZADAS

Trabajo Terminal II

"Coordinación de dos robots móviles terrestres bajo un esquema líder seguidor"

Que para obtener el título de

"Ingeniero en Mecatrónica"

Presentan:

Mauricio Sánchez Ortega

David Alejandro Toro Sandoval

Luis Fernando Zarazua Aguilar

Asesores:

Dr. Juan Luis Mata Machuca

M. en C. Mauricio Méndez Martínez

Presidente del Jurado

Dra. Blanca Rosa Briseño Tepepa

Profesor titular

M. en C. Griselda Sánchez Otero

DEDICATORIA

"La única diferencia que existe entre existo y el fracaso está en los ojos con los que te miras".

Dedicamos este trabajo a cada uno de nuestros profesores de la UPIITA ya que somos la suma de todos y cada uno de ellos. A nuestros padres que nuestros éxitos también son sus éxitos. Y a cada persona que esté interesada en este tipo de trabajos, esperando le sea de gran utilidad y ayuda.

AGRADECIMIENTOS

Agradezco a Dios por haberme permitido terminar una de las etapas más importantes de mi vida, esperando que sus planes y mis planes sigan coincidiendo.

A mi familia que siempre ha estado ahí, apoyándome en cada paso que doy y que nunca han dejado de creer en mi. En especial a mi mamá que sufrió conmigo cada incertidumbre y desvelo y a mi papá que siempre ha estado ahí apoyándome.

Al Dr. Juan Luis Mata Machuca por habernos dado el beneficio de la duda y habernos permitido llevar acabo este proyecto que, sin su buena voluntad, este proyecto jamás se hubiera realizado, además de que siempre estuvo en la mejor disposición de ayudarnos en cualquier cosa; resolviéndonos cualquier duda, aconsejándonos.

A la profesora Griselda Sánchez Otero que nos guío durante todo un año y que, sin su pericia, este trabajo no se hubiera culminado de forma exitosa.

A mis compañeros de equipo Zarazua y Toro por haberme permitido pasar todo un año de arduo trabajo a su lado y que sin su empeño esto nunca hubiera salido.

Mauricio Sánchez Ortega

A nuestros padres y profesores por mostrarnos el camino de la superación. A nuestros amigos por permitirnos aprender de la vida a su lado.

David Alejandro Toro Sandoval

Agradezco a Dios, a mis padres, a mis profesores, a mis compañeros por todo apoyo, enseñanzas, presupuesto y tiempo de calidad brindados, ya que sin estos el proyecto nunca hubiera sido terminado.

Luis Fernando Zarazua Aguilar

Contenido

Nomenclatura xix

Glosario xxi

Resumen/Abstract xxv

Objetivos xxvii

Introducción xxix

Antecedentes xxx

Nacionales xxx

Internacionales xxxii

Planteamiento del problema xxxiii

Organización del reporte xxxiv

1. Marco de referencia 1

1.1. Marco teórico 1

1.1.1. Robots móviles terrestres 1

1.1.2. Cinemática de robots móviles 2

1.1.3. AMCL 5
1.1.4. VFH (Vector Field Histogram) 6
1.1.4.1. Primer nivel: Mapa 2D 6
1.1.4.2. Segundo nivel: Histograma polar 7
1.1.4.3. Tercer nivel: Comando de dirección 7
1.1.5. Esquemas líder - seguidor 8
1.2. Marco Procedimental 9
1.2.1. Sistema mecatrónico 10
1.2.2. Diseño mecatrónico 11
1.2.3. Proceso de análisis jerárquico (AHP) 11

2. Diseño del Sistema **13**
2.1. Diseño Conceptual 13
2.1.1. Necesidades y requerimientos 13
2.1.2. Descomposición por áreas funcionales 16
2.1.2.1. Área Funcional 1: Estructura 17
2.1.2.2. Área Funcional 2: Procesamiento 17
2.1.2.3. Área Funcional 3: Percepción 18
2.1.2.4. Área Funcional 4: Alimentación 18
2.1.2.5. Área Funcional 5: Movimiento 19
2.1.3. Características y/o necesidades principales de cada área funcional 19
2.1.4. Características principales de los robots considerados. ROSbot, Turtlebot, TX1 “Jet” Robot Kit 21
2.1.4.1. Rosbot 21
2.1.4.2. Turtlebot3 Burger 22
2.1.4.3. TX1 “Jet” Robot Kit 23
2.1.5. Tablas de selección de los robots móviles 23
2.1.6. Criterios para el diseño y selecciones de interfaces gráficas . . 26

2.1.6.1. Criterios a evaluar 26
2.1.6.2. 1. Amabilidad 26
2.1.6.3. 2. Interactividad 26
2.1.6.4. 3. Redundancia óptima en el proceso de comunicación 26
2.1.6.5. 4. Eficiencia y Utilidad 26
2.1.6.6. Tabla de Criterios 27
2.1.6.7. Lista de opciones de interfaz 28
2.1.6.8. Diseño de arquitectura general del software 28
2.1.6.9. Concepto Final 31
2.2. Diseño Detallado 33
2.2.1. Área Funcional 1: Estructura (Soporte de componentes) . . . 34
2.2.1.1. Elementos primordiales de la estructura 34
2.2.1.2. Elementos utilizados en cada piso 36
2.2.1.3. Dimensiones 38
2.2.2. Área Funcional 2: Procesamiento (Comunicación de componentes) 39
2.2.2.1. Primera parte: Dispositivos destinados para el procesamiento 40
2.2.2.2. Segunda parte: Funciones a desarrollar en cada tarjeta 45
2.2.2.3. Tercera parte: Diseño del software 46
2.2.3. Área Funcional 3: Percepción (Comunicación entorno - robot) 59
2.2.3.1. Datos propioceptivos 60
2.2.3.2. Sensores integrados 61
2.2.3.3. Encoders rotatorios 61
2.2.3.4. Encoder óptico rotatorio, de tipo relativo (o incremental) 61
2.2.3.5. Encoder óptico absoluto 62
2.2.3.6. Acelerómetros 62
2.2.3.7. Datos exteroceptivos 63

2.2.3.8. Sensores implementados 63
2.2.3.9. Escáner láser de medición de distancias 63
2.2.3.10. Sensor LIDAR LDS-01 64
2.2.3.11. Dimensiones 65
2.2.3.12. Montaje del sensor 66
2.2.3.13. Acondicionamiento 67
2.2.4. Área Funcional 4: Alimentación (Suministro de energía) . . . 68
2.2.4.1. Características Técnicas de la batería LiPo. 71
2.2.4.2. Ensamble de la batería LiPo con la estructura . . . 71
2.2.5. Área Funcional 5: Movimiento (Desplazamiento del robot) . . 72
2.2.5.1. Actuadores Dynamixel 73
2.2.5.2. Análisis interno del motor 74
2.2.5.3. Características del motor 76
2.2.6. Integración del Sistema 83
2.2.6.1. Elementos de Software 85
2.2.7. HMI 87
2.2.7.1. Comunicación peer to peer 88
2.2.8. Validación y análisis de resultados 89
2.2.8.1. Procesamiento 90
2.2.8.2. Seguimiento de trayectoria 90
2.2.8.3. Líder Seguidor 93
2.2.8.4. Evasión de obstáculos 94

3. Integración del Sistema 97

3.1. Implementación del Área Funcional Estructura 97
3.1.1. Armado de los robots móviles 98
3.1.2. Instrucciones de Armado 103
3.1.3. Resultados del proceso de armado 103
3.1.4. Verificación de ensamble del sistema 106

3.2. Implementación del Área Funcional Movimiento 106
3.3. Implementación del Área Funcional Percepción 107
3.3.1. Sensor LIDAR 107
3.3.2. Sensores del motor DYNAMIXEL 111
3.4. Implementación del Área Funcional Alimentación 113
3.5. Implementación del Área Funcional Procesamiento 114
3.5.1. Pasos previos 114
3.5.2. Comunicación 116
3.5.3. Prueba A: Verificación de la matriz de fuerza 120
3.5.3.1. Análisis de resultados 122
3.5.4. Prueba B: Validar el seguimiento de trayectoria 122
3.5.4.1. Análisis de resultados 124
3.5.5. Prueba C: Validar el algoritmo Líder-Seguidor 124
3.5.5.1. Análisis de resultados 125
3.5.6. Prueba D:Validar la evasión de obstáculos 126
3.5.7. Validación de seguimiento y evasión 130
3.5.7.1. Análisis de resultados 131
3.5.8. Prueba E: Validar la integración de evasión y seguimiento . . 131
3.5.8.1. Análisis de resultados 132
3.5.9. Realización de la HMI 132
3.5.9.1. Programa Principal 137
3.5.9.2. Ventana Gráfica 138
3.5.9.3. Programa de ROS 138
3.5.9.4. Framework de Trabajo 139
3.5.9.5. Puntos importantes a considerar 140
3.5.9.6. Modificación del Programa 140
3.5.9.7. Resultados de la ejecución 141
3.6. Especificaciones del Sistema 145
3.6.0.1. Especificaciones del Área de Trabajo 145

3.6.0.2. Especificaciones de la Alimentanción 145
3.6.0.3. Especificaciones del Procesamiento 145

4. Análisis de Costo **147**
4.1. Costos variables 147
4.2. Costos fijos 148
4.3. Costo total 150

5. Conclusiones **151**

Apéndices **161**

Apéndice 1 **163**

Anexos **169**

Anexo 1. Criterios de selección en Áreas funcionales **171**

Índice de figuras

1. (a) Robot Repartidor Autónomo. (b) Generación de trayectoria [8]. . xxxi
2. Ejecución del Sistema de Navegación Evasor de Obstáculos implementando un robot Turtlebot2 [17]. xxxi
3. Control de la formación en un plano. Trabajo de investigación de la Universidad de California [40]. xxxiii

1.1. *Ciclo: Ve, piensa, actúa.* Componentes que conforman un robot móvil autónomo. 2
1.2. Marco de referencia global y el marco de referencia local del robot. . 3
1.3. Parámetros de una configuración diferencial. 4
1.4. Algoritmo AMCL. 5
1.5. Ejemplo de histograma polar. 7

2.1. Áreas Funcionales propuestas. 16
2.2. Robot móvil Rosbot [37] 21
2.3. Robot móvil Turtlebot3 Burger [21] 22
2.4. Robot móvil TX1 “Jet” [13] 23

2.5. Opciones de interfaces gráficas 28
2.6. Tipos de Arquitecturas 29
2.7. Arquitectura de software propuesta para el proyecto 30
2.8. Concepto final 31
2.9. Integración de todas las Áreas Funcionales. 33
2.10. Ensamble de 2 pisos o chapas en forma de celda (base estructural). . 35
2.11. Estructura tipo multiplataforma. 35
2.12. Dimensiones del Turtlebot3 [2]. 39
2.13. Tarjeta ARM Cortex M7 [32]. 44
2.14. Tarjeta Raspberry PI 3, Modelo B [30]. 45
2.15. Entrada y salida del programa que genera la matriz de fuerza. 47
2.16. Algoritmo Generador de la Matriz de Fuerza. 47
2.17. Diagrama de Flujo para la rutina “Obtener al vecino más cercano”. . 48
2.18. Diagrama UML del objeto ForceController a implementar en nodo Líder. 49
2.19. Diagrama de flujo del nodo Líder. 49
2.20. Diagrama de flujo de los métodos del objeto ForceController. 50
2.21. Representación de las distancias entre el líder y el seguidor. 51
2.22. Diagrama UML del objeto FollowerRobot a implementar en nodo seguidor. 52
2.23. Diagrama de flujo del nodo seguidor. 52
2.24. Diagrama de flujo de los métodos del objeto FollowerRobot parte I. . 53
2.25. Diagrama de flujo de los métodos del objeto FollowerRobot parte II. 53
2.26. Diagrama de flujo de los métodos del objeto FollowerRobot parte III. 54
2.27. UML para nodo detector. 55
2.28. Diagrama de flujo nodo detector. 55
2.29. Diagrama de flujo método nodo detector. 56
2.30. Diagrama de flujo para dirección. 56
2.31. Configuración del IP’s para la comunicación vía WiFi.[21] 58

2.32. Ejemplo de configuración WIFI. [29] 58
2.33. Funcionamiento del encoder absoluto. 62
2.34. Funcionamiento del sensor LIDAR. 64
2.35. Dimensiones del sensor LIDAR. Vista superior y lateral. 66
2.36. Vista Frontal del sensor LIDAR. 66
2.37. Pines del sensor LIDAR. 67
2.38. Ubicación de la batería de alimentación. Primera capa del Turtlebot3. 69
2.39. Batería LiPo, con su entrada y salida [31]. 70
2.40. Ensamble de la batería LiPo en la primera capa del Turtlebot3. . . . 72
2.41. Ensamble del rin y la llanta de los robots. 72
2.42. Estructura de la locomoción de los robots. 73
2.43. Ensamble de los actuadores con la estructura. 73
2.44. Explosionado del motor del Dynamixel. 74
2.45. Características del actuador. 75
2.46. Sistema de locomoción en conjunto con los actuadores Dynamixel XL 430 - W250. 76
2.47. Sistema de locomoción en conjunto con los actuadores Dynamixel XL 430 - W250. Vista frontal e inferior. 76
2.48. Nomenclatura de los actuadores Dynamixel [26]. 77
2.49. Gráfica de funcionamiento del motor DYNAMYXEL [26]. 81
2.50. Interfaz de programación de los actuadores Dynamixel. 82
2.51. Capa 1. Implementación de la capa con los actuadores y la batería LiPo. 83
2.52. Capa 2. Implementación de la capa 1, capa 2 y tarjeta Open CR. . . 83
2.53. Capa 3. Implementación de la capa 1, capa 2, capa 3 y tarjeta Raspberry Pi 3. 84
2.54. Vista isométrica del robot con todas las capas ensambladas. 84
2.55. Explosionado y distribución de los componentes que conforman al robot. 85
2.56. Esquema general de funcionamiento del sistema de control para el seguimiento de trayectorias y evasión de obstáculos. 86

2.57. Esquema general de funcionamiento del sistema de control para el seguidor. 86
2.58. Comunicación entre tópicos de los distintos nodos. 87
2.59. Conexión típica de ROS. 88
2.60. Ejemplo de comunicación de ROS con agentes externos. 89
2.61. Trayectoria propuesta 1 (unidades en metros). 91
2.62. Trayectoria propuesta 2 (unidades en metros). 91
2.63. Matriz de fuerza para trayectoria propuesta 1 (unidades en metros). 92
2.64. Matriz de fuerza para trayectoria propuesta 2 (unidades en metros). 92
2.65. Resultados extraídos de la simulación de seguimiento de trayectoria (unidades en metros). 93
2.66. Resultados extraídos de la simulación del líder (unidades en metros). 94
2.67. Resultados extraídos de la simulación del seguidor (unidades en metros). 94
2.68. Obstáculos para la validación del algoritmo propuesto. 95
2.69. Puntos de ruta alcanzados por el líder durante la evasión de obstáculos (unidades en metros). 95

3.1. Caja con los diferentes componentes del robot. 98
3.2. Caja de componentes 1. *MicroSD, Raspberry Pi 3, motores Dynamixel y Tarjera OpenCR Cortex M7* 99
3.3. Caja de componentes 2. *Bateria LiPo, cargador, desarmador, sensor LIDAR, conector USB2LDS y soportes para pcb's* 99
3.4. Caja de componentes 3. *"Waffle - Plate"pisos de los robots* 100
3.5. Caja de componentes 4. *Llantas, ruedas, tornillos, piezas de ensamble, cables y una rueda loca* 100
3.6. Fuente de voltaje para bateria LiPo. 101
3.7. Unión de los pisos del robot. 101
3.8. Piezas Ensambladas. 102
3.9. Ensamble del robot. 102

3.10. Ensamble del primer piso. 104
3.11. Ensamble del segundo piso. 104
3.12. Ensamble del tercer piso. 105
3.13. Ensamble final del robot. 105
3.14. Robot líder y robot seguidor seguidor. 106
3.15. Area funcional movimiento, que cuenta con los dos motores Dynamixel y la OpenCR. 107
3.16. Montaje del sensor LIDAR en la estructura. 108
3.17. Sensor LIDAR del Turtlebot3 burger. 109
3.18. Distancias del LIDAR en RVIZ 110
3.19. Visualización en rqt de los valores del sensor LIDAR. 111
3.20. Motores DYNAMIXEL XL430. 111
3.21. Ensamble de los motores, OpenCR y Batería LiPo. 112
3.22. Localización de los botones de prueba en la tarjeta OpenCR. 112
3.23. Batería LiPo LB-012. 113
3.24. Creación de una nueva red en Ubuntu. 117
3.25. Elegir tipo de conexión. 117
3.26. Configuración del Modo y Nombre. 118
3.27. Configuración de la seguridad. 118
3.28. Muestra del resultado correcto de la ejecución del comando ping. . . 119
3.29. RQt mostrando los datos del tópico battery. 119
3.30. Reconstrucción de la trayectoria A. 120
3.31. Reconstrucción de la trayectoria B. 121
3.32. Reconstrucción de la trayectoria C. 121
3.33. Puntos de trayectoria A. 121
3.34. Puntos de trayectoria B. 122
3.35. Puntos de trayectoria C. 122
3.36. Error del líder en seguimiento de la trayectoria A. 123
3.37. Error del líder en seguimiento de la trayectoria B. 123

3.38. Gráfica de las posiciones alcanzadas y el error. 124
3.39. Error del seguidor en la trayectoria A. 125
3.40. Error del seguidor en la trayectoria B. 125
3.41. Nodo Laser_Distance. 126
3.42. Detección del obstáculo cercano colocado en el primer cuadrante. . . 127
3.43. Detección del obstáculo cercano colocado en el segundo cuadrante. . 127
3.44. Función follow para decidir que ejecutar. 128
3.45. Función de seguimiento de trayectoria. 129
3.46. Algoritmo para aplicar VFH. 129
3.47. Función para ir a un sector libre. 130
3.48. Error del Líder durante la evasión. 131
3.49. Error del Seguidor durante la evasión. 132
3.50. RQt para graficar distintos tópicos con respecto al tiempo. 133
3.51. RQt mostrando y filtrando los distintos tipos de mensajes. 133
3.52. Pantalla de bienvenida de QtCreator. 134
3.53. Ventana para compilar paquete en QtCreator. 135
3.54. Ingreso de la carpeta de compilación. 135
3.55. Ingreso y compilación con el comando CMake. 136
3.56. Ventana en Qt con el paquete qgui. 137
3.57. Framework de Qt para GUI. 139
3.58. Ventana incial de la interfaz. 142
3.59. Configuración de la interfaz con la trayectoria 1 seleccionada y apagada. 142
3.60. Configuración de la interfaz con la trayectoria 2 seleccionada y encendida. 143
3.61. Configuración de la interfaz con la trayectoria 3 seleccionada y encendida. 143
3.62. Interfaz con los mensajes de prueba enviados. 144
3.63. Interfaz con el historial de trayectorias seleccionadas. 144
3.64. Interfaz mostrando las opciones de graficación. 145

3.65. Nodos de Rosgraph. 146

1. Configuración de nodos en ROS parte A 163
2. Configuración de nodos en ROS parte B 164
3. Configuración de nodos en ROS parte C 165
4. Configuración de nodos en ROS parte D 166
5. Configuración de nodos en ROS parte E 167

Índice de Tablas

2.1. Tabla de Necesidades 14
2.2. Tabla de Selección: Procesamiento. 24
2.3. Tabla de Selección: Percepción. 24
2.4. Tabla de Selección: Estructura. 25
2.5. Tabla de Selección: Motores y Controladores. 25
2.6. Tabla de Selección de las diferentes opciones de interfaz. 27
2.7. Comparación entre Arquitecturas 29
2.8. Características de la Tarjeta ARM Cortex M7 [32] 41
2.9. Características de la Tarjeta ARM Cortex M7 [32]. 42
2.10. Características de la Tarjeta ARM Cortex M7 [32]. 43
2.11. Características de la tarjeta Raspberry PI 3, Modelo B [30]. 44
2.12. Características del sensor LIDAR. 65
2.13. Descripción de los pines del sensor LIDAR. 68
2.14. Descripción de los pines del motor del sensor LIDAR. 68
2.15. Comandos para operación del LIDAR. 68
2.16. Especificaciones técnicas de la batería. [11] 71

2.17. Características técnicas del actuador Dynamixel XL 430 W250 [25]. . 78
2.18. Características técnicas del actuador Dynamixel XL 430 W250 [25]. . 79
2.19. Características técnicas del actuador Dynamixel XL 430 W250 [25]. . 80

4.1. Tabla de costos variables. 148
4.2. Días trabajados TT1. 148
4.3. Días trabajados TT2. 149
4.4. Total de días y horas trabajadas. 149
4.5. Costo total de gastos fijos. 149

1. Tabla de selección para Percepción 171
2. Tabla de selección para Estructura 172
3. Tabla de selección para Procesamiento 173
4. Tabla de selección para Motores 174

Nomenclatura

Lista de nomenclaturas

2D - 2 dimensiones

3D - 3 dimensiones

V - Volts

A - Ampere

mA - MiliAmpere

A-h - Ampere-hora

mm - Milímetro

cm - Centímetro

m - Metros

g - Gramos

GB - Gigabyte

s - Segundo

m/s - Metros por segundo

RAM - Random Access Memory

SLAM - Simultaneous Localization And Mapping

USB - Universal Serial Bus

WPL - Lista de puntos de ruta

VFH - Vector Field Histogram

MCL - Monte Carlo Localization

AMCL - Adaptative Monte Carlo Localization

FLA - Follower Leader Aproach

MRS - Multi Robot Sistems

RBP - Raspberry Pi

APH - Analytic Hierarchy Proccess

ROS - Robotic Operating System

LFA - Lider Follower Algorithm

HMI - Human Machine Interface

PO - Programación orientada a objetos

IEEE - Institute of Electrical and Electronics Engineers

AI - Artificial Inteligence

ABD - Acrylonitrile Butadiene Styrene

UML - Unified Modeling Language

LASER - Light Amplification by Stimulated Emission of Radiation

LIDAR - Laser Imaging Detection and Ranging

LiPo - Polímero de Litio

PCM - Protection Circuit Module

Glosario

ROS

Por sus siglas en inglés *Robot Operating System* (Sistema Operativo Robótico). Es un sistema operativo para robots que posé herramientas, librerías y convenciones para ayudar a los desarrolladores de software a crear aplicaciones, con el objetivo de simplificar la tarea de crear complejos y robustos sistemas robóticos, proveyendo de varios ambientes de desarrollo especializado en programas y aplicaciones robóticas [42].

Nodo en ROS:

Un *nodo* es un proceso que ejecuta un cálculo, el cual se visualiza en un gráfico como un punto de intersección para mejor comprensión y se comunica con otros nodos mediante *tópicos* en ejecución. Un nodo esta diseñado para operar en una escala finita. Un ejemplo de como funcionan los nodos se podría ver en un sistema para controlar un robot; existirá un nodo para cada acción del robot, es decir, habrá un nodo que controle cada sensor que el robot tenga, habrá otro nodo que controle los motores del robot, la localización, la planificación de la trayectoria, hará un nodo

más que proporcione una vista gráfica del sistema y otro que manipule todos los datos obtenidos que se obtuvieron sensando y así siguiendo la misma lógica para cada acción o tarea que tenga que ejecutar el robot.

Tópico:

Un *tópico* es el nombre con el cual se identifica el contenido de un *mensaje*, es decir, literalmente un *tópico* es un tema de conversación [42]. Por lo tanto un nodo que está interesado en un determinado tipo de dato se *suscribe* al tópico correspondiente por ejemplo al estado de la batería de un determinado robot, en donde puede haber varios publicadores y suscriptores concurrentes en un mismo tópico. De igual manera un sólo nodo puede publicar y/o suscribirse a múltiples tópicos con el fin de extraer todos los datos que el nodo necesite procesar para dar una salida [1].

Mensaje en ROS:

Los *nodos* se comunican entre sí mediante la publicación de *mensajes* a los tópicos. Un *mensaje* es una estructura de datos simple, que comprende campos escritos [35]. En otras palabras, la forma en que un nodo envía o recibe información es a través de un *mensaje*. Los mensajes contienen variables como enteros, puntos flotantes y booleanos [42]. En analogía los mensajes son como las clases en PO y los tópicos como los objetos en PO, pues los mensajes indican que tipos de variables se tienen y los objetos guardan un valor correspondiente al tipo de dato en específico, usando el ejemplo de la batería el mensaje sería el campo del voltaje en la batería y el tópico el determinado valor de la batería 11.6v en promedio en este proyecto.

Callback:

Callback o "devolución de llamada", es una función en ROS que normalmente controla los mensajes, es decir, cada vez que llega un mensaje, ROS llama a su *message manager* (administrador de mensajes) y le pasa el nuevo mensaje, para que este comience a trabajar con base en el mensaje recibido, en resumen ejecuta la función al momento de recibir un mensaje actuando como una interrupción al sistema en PO.

Filtrado de partículas:

El *filtrado de partículas* consiste en estimar los estados internos en los sistemas dinámicos cuando se hacen observaciones parciales, y las perturbaciones aleatorias que están presentes en los sensores y en el sistema dinámico. El objetivo del filtrado de partículas es calcular las distribuciones posteriores de los estados de algún proceso de *Markov*, dadas algunas observaciones ruidosas y parciales.

Topología:

El *filtrado de partículas* consiste en estimar los estados internos en los sistemas dinámicos cuando se hacen observaciones parciales, y las perturbaciones aleatorias que están presentes en los sensores y en el sistema dinámico. El objetivo del filtrado de partículas es calcular las distribuciones posteriores de los estados de algún proceso de *Markov*, dadas algunas observaciones ruidosas y parciales.

Meta Sistema Operativo

Resumen/Abstract

"Coordinación de dos robots móviles terrestres bajo un esquema líder seguidor"

Palabras Clave: Líder, seguidor, autónomo, robots móviles, navegación, control, robótica cooperativa, seguimiento de trayectoria, turtlebot3, ROS, lidar, evasión de obstáculos, mapeo, slam, vector field histogram, upiita, ipn.

Abstract:

This work describes the proposal for the design, implementation and validation for the control of two land mobile robots under a *LFA*, where, the leading robot will have the task of following a predetermined trajectory for a user, with the autonomy to dodge fixed objects and share information of the waypoints reached, to the follower robot. The follower robot will have the ability to imitate the movements of the leading robot based on the information received and follow it. All this developed in controlled surfaces and environments.

Resumen:

Este trabajo describe la propuesta para el diseño, implementación y validación para el control de dos robots móviles terrestres bajo un esquema *"lider-seguidor"*,

donde, el robot líder tendrá la tarea de seguir una trayectoria predeterminada por un usuario, con la autonomía para esquivar objetos fijos y compartir información de la posición de dichos objetos con el robot seguidor. El robot seguidor tendrá la capacidad de imitar los movimientos del robot líder con base en la información proporcionada por parte del robot líder. Todo esto desarrollado en superficies y ambientes controlados.

Objetivos

Objetivo general

Coordinar dos robots móviles terrestres bajo un esquema *"líder-seguidor"* para el seguimiento de trayectorias predeterminadas en superficies y ambientes controlados.

Objetivos Particulares

1. Seleccionar y construir dos sistemas de locomoción para los móviles que les permita moverse en superficies y ambientes controlados.

2. Diseñar e implementar un control de seguimiento de trayectoria para el robot líder.

3. Diseñar e implementar un control de seguimiento al líder para el robot seguidor.

4. Establecer e implementar un método de comunicación apropiado para la configuración líder-seguidor.

5. Diseñar e implementar un algoritmo para la evasión de obstáculos fijos.

6. Seleccionar e implementar un sistema de sensado en el robot líder, para su propia localización y detección de obstáculos.

7. Seleccionar e implementar un sistema de sensado en el robot seguidor, para la identificación del líder.

8. Diseñar e implementar una GUI para el monitoreo del sistema, la selección de una trayectoria a seguir y delimitar el área de trabajo.

Introducción

Los seres humanos siempre hemos buscado la manera de simplificar la ejecución de tareas, por lo que hemos encontrado diferentes soluciones, unas mejores que otras, pero en donde sin duda se destaca es en la robótica, la cual, ofrece una solución para los problemas que el ser humano desea resolver en diversos escenarios de su cotidianidad usando en conjunto la mecánica y electrónica. Desafortunadamente no todas las tareas pueden ser realizadas por un solo robot y es ahí, dónde surge la robótica cooperativa, proporcionando métodos, teoremas e investigaciones para su desarrollo [24].

La coordinación de robots, en especial el control aplicado a la coordinación, empezó a ser estudiado y aplicado por la comunidad científica, quienes notaron significantes ventajas al implementar un grupo de robots para la realización de tareas [6], como: la eficiencia a la hora de realizar una tarea (dando como resultado la reducción de costos materiales, económicos, energéticos, etc.), el mejoramiento del desempeño y la robustez; permitiéndo realizar tareas como exploración, mantenimiento, inspección, vigilancia, búsqueda de objetos, operaciones de seguridad, construcción de mapas, reconocimiento de terrenos desconocidos o peligrosos, transportación, búsqueda y rescate, entre otras, abriendo paso a un amplio número de aplicaciones desarrolladas en diversos ambientes con la ayuda de diferentes tipos de robots (terrestres, aéreos, espaciales, marinos), según sea el caso. Quedando como única limitante la

imaginación humana [19].

Un aspecto que es de vital importancia en la robótica cooperativa es lograr controlar grupos de robots de manera simultánea, ya que estos, deben de trabajar de forma sincronizada. Para lograrlo y que los robots se muevan uniformemente manteniendo una formación, es necesario abordar el problema a través de técnicas de control cooperativo, entre las más conocidas encontramos: las *estructuras virtuales*, las configuraciones *líder-seguidor* y las *aproximaciones basadas en comportamientos* [24]. Cada una de estas técnicas han venido siendo estudiadas, y aplicadas por diferentes sectores de la sociedad, en donde sin duda destaca el sector militar y espacial.

Antecedentes

Nacionales

A nivel institucional, en 2017 se llevó a cabo en la UPIITA el Trabajo Terminal de Mecatrónica titulado "Robot Repartidor Autónomo". Dicho trabajo se desarrolló en el sistema operativo robótico ROS (Robotic Operating System) y se ejecutó a través de una tarjeta Nvidia Jetson y un sensor LIDAR para la recabación de datos. Como su nombre lo dice, el trabajo consistió en un robot capaz de cargar hasta 15 kg y el cual dentro de un sistema de navegación (área de trabajo) generaba una trayectoria de manera autónoma con la finalidad de llegar a trayectorias "metaz entregar cualquier producto que no sobrepasará la carga máxima que era capaz de soportar el robot.

A pesar de que este trabajo no comparte ninguna similitud con la robótica cooperativa, es un trabajo de gran ayuda para el desarrollo del proyecto, ya que se implementaron herramientas como odometría, o el algoritmo de Localización Monte Carlo (MCL por sus siglas en ingles Monte Carlo Localization) para la ubicación del robot dentro del medio de trabajo, al igual que la generación de campos estáticos, para su navegación autónoma.

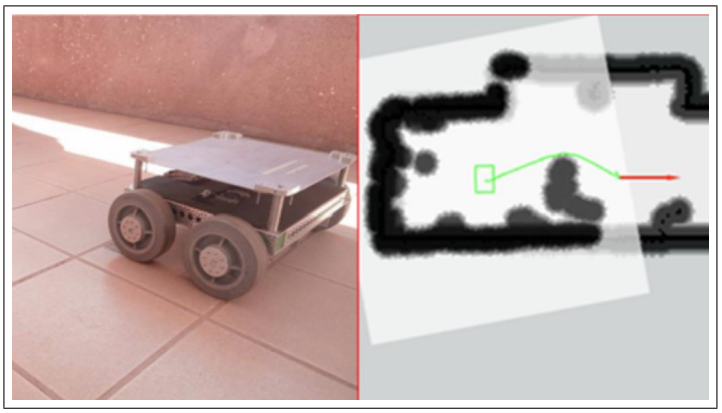

Figura 1: (a) Robot Repartidor Autónomo. (b) Generación de trayectoria [8].

De igual manera, dentro de la UPIITA en 2014, se llevó a cabo el Trabajo Terminal de Mecatrónica, titulado "Sistema de Navegación Evasor de Obstáculos en un Robot Móvil", el cual implementaba un robot Turtlebot2 y tenía como objetivo llegar de un punto inicial a un punto final, evadiendo obstáculos estáticos en el camino. En este trabajo se implementaron distintos algoritmos de control, como: algoritmo generador de celdas de ocupación probabilística en un mapa 2D, algoritmo de planeamiento global utilizando información 2D del mapa del área de trabajo, junto con un algoritmo de planeamiento local reactivo.

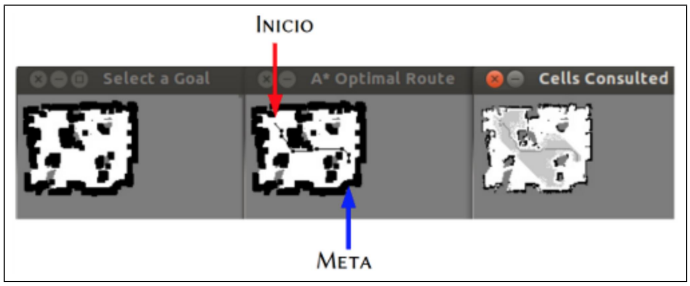

Figura 2: Ejecución del Sistema de Navegación Evasor de Obstáculos implementando un robot Turtlebot2 [17].

Por otra parte, la UNAM en conjunto con el Instituto Tecnológico de Ensenada, Baja California desarrollaron en 2013 una investigación que se tituló control visual

para la formación de robots móviles tipo uniciclo bajo el esquema líder-seguidor el cual habla sobre una propuesta de control visual para la formación de robot móviles, en donde considera una sola cámara fija observando el espacio de trabajo de los robots, y daba información sobre los movimientos y las posiciones de los robots, y esta información es compartida tanto por el líder tanto como por el seguidor [6].

Internacionales

En la Universidad Militar Nueva Granada (Bogotá, Colombia) en el 2014, llevó acabó una investigación que se tituló " Flotilla de robots para trabajos en robótica cooperativa.[el] cual habla de un diseño de un sistema de control descentralizado, el cual permite orientar un grupo de robots mientras se desplazan manteniendo una formación predeterminada. Basándose en el control líder-seguidor[19].

Por parte del Instituto de Ingeniería Eléctrica y Electrónica (IEEE por sus siglas en inglés) se llevó a cabo un trabajo titulado "Backstepping based multiple mobile robots formation control"(formación de múltiples robots móviles basado en el control Backstepping) que es un trabajo de investigación basado en el control de formación de múltiples robots móviles, liderados por un robot que se denomina "líder", en donde se presenta un modelo cinemático para el sistema "líder-seguidor.[el] cual utiliza coordenadas cartesianas en lugar de coordenadas polares, tomando la idea del integrador backstepping, en donde se deriva un control global estable para todo el sistema[40].

La Universidad Politécnica Superior de Elche desarrolló un trabajo de investigación llamado "Seguimiento de robots mediante visión artificial. Aplicación al mantenimiento de formaciones"donde se describe el desarrollo e implementación de algoritmos de navegación que permite a un robot móvil seguir la ruta realizada por otro robot (líder) de forma que un conjunto de robots siga la determinada trayectoria, manteniendo una formación entre ellos. El trabajo implementa diferentes algoritmos de control de movimientos, de manera que el robot que sigue a el líder se mantiene a

una distancia (posición y orientación) determinada con respecto a el robot líder[15].

El Departamento de Ingeniería Electrónica y Ciencia de la Comunicación de La Universidad de California desarrolló un trabajo de investigación llamado "Formation Control of Nonholomic Mobile Robots with Omnidirectional Visual Servoing and Motion Segmentaion"(Control para la formación de robots móviles no- holonómicos con servo visión omnidireccional y movimiento segmentado) el cual habla acerca de un conjunto de robots móviles omnidireccionales basado en el diseño líder-seguidor usando visión omnidireccional, donde se especifica la formación deseada en un plano y esta es traducida en un control servo visual para cada seguidor. En el trabajo también muestran que la dinámica de la linealización de retroalimentación del líder-seguidor sufre en consecuencia de las restricciones no holonómicas de los robots y la no linealidad del modelo [40].

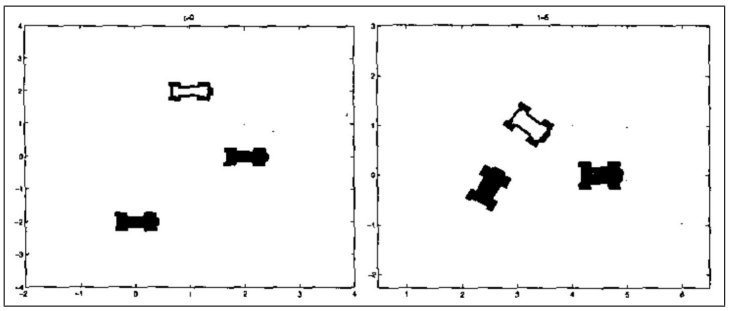

Figura 3: Control de la formación en un plano. Trabajo de investigación de la Universidad de California [40].

Planteamiento del problema

En el presente trabajo se aborda el desarrollo de un esquema de control *líder-seguidor*, implementado a dos robots móviles terrestres, los cuales tendrán el objetivo de llegar a diferentes puntos o trayectorias dentro de un área de trabajo, evadiendo obstáculos que pudieran haber dentro del escenario de pruebas o en la trayectoria.

La tarea a cumplir por parte del *robot líder* es la de seguir alguna trayectoria proporcionada por el usuario (por medio de una interfaz gráfica) y evadir los obstáculos fijos dentro de un área de trabajo controlada, mientras que la tarea a cumplir por parte del *robot seguidor*, es imitar el comportamiento del líder de manera autónoma.

Todo esto con el objetivo de aportar conocimiento en el rubro de la robótica cooperativa, en especial en la robótica aplicada al control líder-seguidor, con fundamentos en el desarrollo del proyecto SIP 20181591.

Organización del reporte

El presente trabajo se divide en 6 capítulos principales , que son: *Marco de Referencia, Diseño Conceptual, Diseño Detallado, Integración del Sistema, Análisis de Costos* y *Conclusiones*.

En el primer capítulo llamado *Marco de Referecia* se fundamenta toda la metodología del diseño del control de los robots móviles y del control líder-seguidor. Se describen las características fundamentales del sistema operativo ROS *(Robotic Operating System)*, así como sus librerías AMCL *(Adaptive Monte Carlo Localization)* y VFH *(Vector Field Histogram)*, en general se aporta todos los conceptos y teoría previa para realizar el proyecto.

En el segundo capítulo llamado *Diseño Conceptual* se analizan las principales Áreas Funcionales que conforman a cada robot, como los son *Estructura, Procesamiento, Alimentación, Movimiento* y *Percepción*, y de las cuales se habla de cada una de ellas, explicando su interpretación, necesidad, y requerimiento, al igual que se describe la razón por la cual se escogieron dichas áreas funcionales dentro de los robots móviles y todo lo que conllevan, aplicándoles *tablas de selección* con la finalidad de seleccionar los mejores componentes que conforman cada área.

En el tercer capítulo llamado *Diseño Detallado* se aborda más a fondo cada una de las áreas funcionales junto con cada uno de sus componentes y se dan sus las

especificaciones técnicas haciendo la integración de cada área funcional que conforma el total del sistema. De igual manera, se muestran diagramas de flujo de los distintos algoritmos de control.

En el cuarto capítulo llamado *Integración del Sistema* se habla acerca de la implementación, físicamente todo lo que se ha venido trabajando en los capítulos anteriores y de la misma manera se aborda la integración del sistema por cada área funcional, complementándolo con fotos, gráficas, e imágenes reales del proyecto.

En el capítulo *Análisis de Costos* se lleva acabo un análisis del costo total de proyecto, partiendo de sus costos fijos y costos variables.

En el sexto y último capítulo se *Conclusiones* se detalla los pormenores del proyecto y se explican todos los objetivos que se cumplieron así como de las posibles mejoras y trabajos a futuro.

Marco de referencia

1.1. Marco teórico

1.1.1. Robots móviles terrestres

Una manera de entender este tipo de robots es por medio de las distintas áreas de conocimiento que se integran para su desarrollo. Lo primero en tomar en cuenta es ¿cómo van a moverse? y ¿qué mecanismo de movimiento usarán? La cinemática, la dinámica y la teoría del control de los mecanismos utilizados, son necesarios para el desplazamiento correcto de los robots. El diseño e implementación de los sistemas perceptuales robustos, la visión computacional y los sistemas sensoriales, son necesarios para la obtención de la información del entorno y con base en esta información la toma de decisiones.

De igual manera, es necesario tener conocimientos acerca de localización y navegación que implican algoritmos informáticos, de inteligencia artificial y teoría probabilística, para el cumplimiernto de tareas de manera autónoma.

Todos estos elementos forman el llamado ciclo *"Ve, piensa, actúa"*, tal como se muestra en la Figura 1.1 [36].

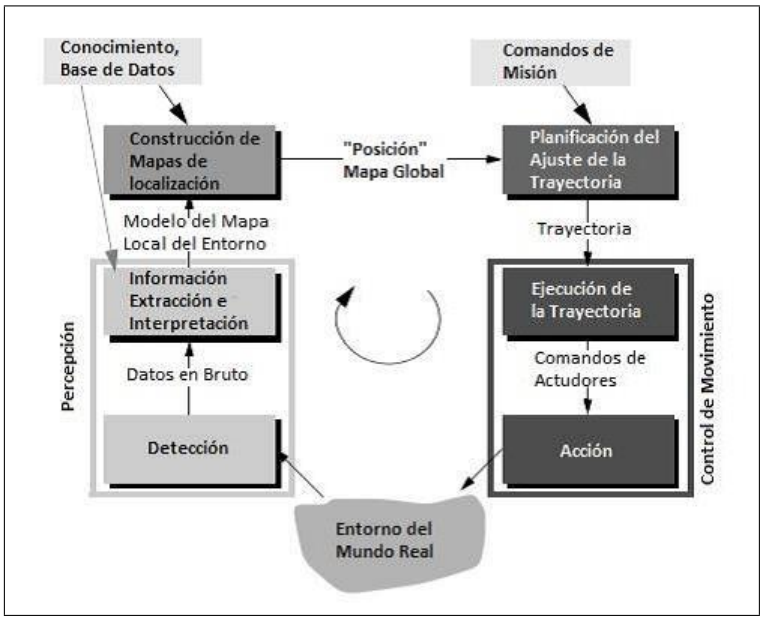

Figura 1.1: *Ciclo: Ve, piensa, actúa.* Componentes que conforman un robot móvil autónomo.

1.1.2. Cinemática de robots móviles

El análisis cinemático es esencial para el buen diseño de los robots móviles y para el diseño de controladores, dada cierta configuración de un robot móvil. Para comenzar a entender el movimiento de los robots móviles primero es necesario representar la posición de un robot en un marco de referencia [36].

En la Figura 1.2 la posición de "P" en el marco de referencia global se especifica mediante las coordenadas "x" e "y", y la diferencia angular entre los marcos de referencia global y local viene dada por "θ". Se puede describir la posición del robot (ξ_l) como un vector conformado por estos tres elementos, teniendo en cuenta el uso del subíndice "I" el cual nos indica el marco de referencia global.

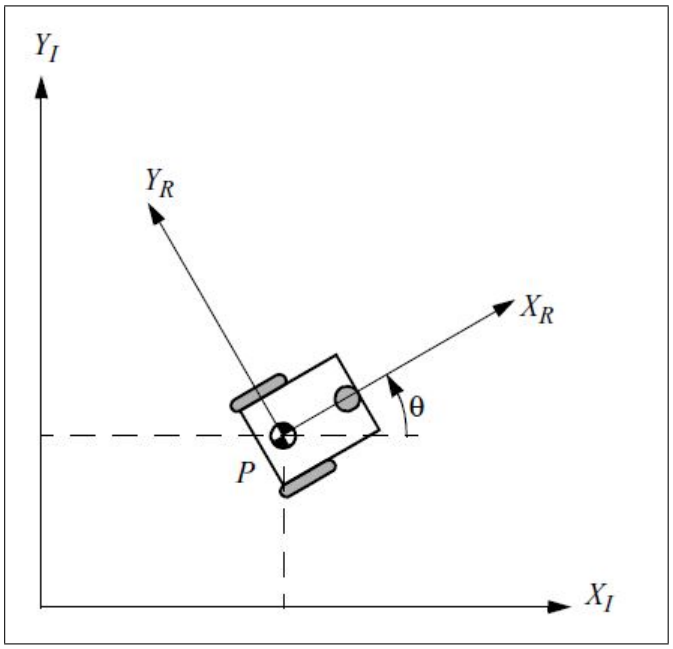

Figura 1.2: Marco de referencia global y el marco de referencia local del robot.

$$\xi_l = \begin{bmatrix} x \\ y \\ \theta \end{bmatrix}$$

ξ_l Posición conforme a un marco de referencia l.

Para describir la cinemática del robot se calcula el movimiento del motor en términos de las componentes del movimiento, recurriendo así a la matriz de rotación ortogonal:

$$R(\theta) = \begin{bmatrix} \cos(\theta) & \sin(\theta) & 0 \\ -\sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix}$$

Dicha matriz puede ser usada para inspeccionar el movimiento del robot en ambos marcos de referencia, es decir el uso de esta matriz permite cambiar del marco de referencia local de robot al global y viceversa. Las expresiones para obtener el movimiento conforme a cada marco de referencia son:

$$\dot{\xi_R} = R(\theta)\,\dot{\xi_I}$$
$$\dot{\xi_I} = R(\theta)^{-1}\,\dot{\xi_R}$$

ξ_R: Velocidad conforme al marco de referencia global.

ξ_I: Velocidad conforme al marco de referencia del robot.

$R(\theta)^{-1}$: Inversa de la matriz de rotación.

A su vez $\dot{\xi_I}$ se compone de las siguientes variables:

$$\dot{\xi_I} = \begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix}$$

$\dot{x}$: Velocidad en "x" conforme al marco de referencia propuesto.

$\dot{y}$: Velocidad en "y" conforme al marco de referencia propuesto.

$\dot{\theta}$: Velocidad angular conforme al marco de referencia propuesto.

Estas velocidades se representan en el robot como se indica en la Figura 1.3.

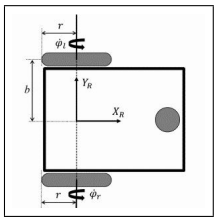

Figura 1.3: Parámetros de una configuración diferencial.

Parte de la cinemática de los robots móviles también es determinar la maniobrabilidad del robot δ_M, la cual se compone de la suma del grado de conducción δ_s y el grado de movilidad δ_m. Para el caso de una configuración diferencial los valores son:

$$\delta_m = 2$$
$$\delta_s = 0$$
$$\delta_M = \delta_m + \delta_s = 2$$

1.1.3. AMCL

AMCL es una librería de ROS *(ver definicón de ROS en la sección Glosario)*, la cual hace uso de un sistema de localización probabilístico para un robot, en un entorno de dos dimensiones (2D). Esta librería implementa el enfoque de localización adaptativo de Monte Carlo [38], junto con los siguientes algoritmos.

Algorithm Augmented_MCL($\mathcal{X}_{t-1}, u_t, z_t, m$):
- static $w_{\text{slow}},\ w_{\text{fast}}$
- $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$
- for $m = 1$ to M do
  - $x_t^{[m]} =$ **sample_motion_model**$(u_t, x_{t-1}^{[m]})$
  - $w_t^{[m]} =$ **measurement_model**$(z_t, x_t^{[m]}, m)$
  - $\bar{\mathcal{X}}_t = \bar{\mathcal{X}}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$
  - $w_{\text{avg}} = w_{\text{avg}} + \frac{1}{M}\, w_t^{[m]}$
- endfor
- $w_{\text{slow}} = w_{\text{slow}} + \alpha_{\text{slow}}(w_{\text{avg}} - w_{\text{slow}})$
- $w_{\text{fast}} = w_{\text{fast}} + \alpha_{\text{fast}}(w_{\text{avg}} - w_{\text{fast}})$
- for $m = 1$ to M do
  - with probability $\max(0.0,\ 1.0 - w_{\text{fast}}/w_{\text{slow}})$ do
    - add random pose to $\mathcal{X}_t$
  - else
    - draw $i \in \{1, \ldots, N\}$ with probability $\propto w_t^{[i]}$
    - add $x_t^{[i]}$ to $\mathcal{X}_t$
  - endwith
- endfor
- return $\mathcal{X}_t$

Figura 1.4: Algoritmo AMCL.

De manera general, para un mapa ya establecido este algoritmo estima la *posición* y *orientación* de un robot a medida que este empieza a moverse, usando un *filtrado de partículas* para representar todas las posibles posiciones, en donde si los datos son confiables y se relacionan con el estado predicho el *filtrado de partículas* tiende a converger en una posición en particular.

1.1.4. VFH (Vector Field Histogram)

Es un método para la *detección* y *evasión de obstáculos*, el cual al ser aplicado permite continuar con su objetivo y así poder seguir una trayectoria. El método toma como entrada los datos capturados por sensores de tipo láser o ultrasónicos y con base en ello forma tres niveles de representación de datos. El primero es un mapa de dos dimensiones divido por celdas donde se guardarán las distancias. El segundo es un histograma polar, construido alrededor de la posición momentánea del robot y el tercero es la dirección que debe seguir el robot para llegar a su meta y a su vez evadir un obstáculo si está cercano.

1.1.4.1. Primer nivel: Mapa 2D

De los datos obtenidos por los sensores los cuales son distancia para cada ángulo, se va formando un mapa de celdas o matriz que representa el espacio alrededor del robot, en donde el el valor de cada celda es dado por la ecuación 1.1 .

$$m_{(i,j)} = (c^*_{(i,j)})^2(a - bd^2_{(i,j)}) \tag{1.1}$$

$c^*_{(i,j)}$: Valor de certeza en dicha celda

$d_{i,j}$: Distancia entre la celda i,j y el centro del vehículo

a, b: Constantes positivas que cumplan $a - bd_{max} = 0$

Para el cálculo del valor de certeza se deben obtener 2 mediciones muy próximas y con ello corroborar si hay un obstáculo en la celda, dándole así un mayor valor a la celda, posteriormente las constantes se eligen considerando $''d_{max}$" como distancia mayor a la celda más lejana al robot y con base a esta distancia se calculan las constantes $''a$" y $''b$" de tal manera que la más lejana devuelva un valor de 0 y la más cercana el máximo valor posible a su vez manteniendo una relación lineal. Con estas condiciones se tendrá una magnitud para cada celda $m_{(i,j)}$ que nos indicará la presencia o no de un obstáculo.

1.1.4.2. Segundo nivel: Histograma polar

Una vez obtenido el mapa 2D se procede a simplificar estos datos para obtener un histograma polar como el mostrado en la Figura 1.5, primero se define una resolución arbitraria α de modo que $n = \frac{360}{\alpha}$.Posteriormente se definen k sectores que corresponden a un ángulo discreto ρ tal que $\rho = k\alpha$, dividiendo así en un cúmulo de datos (histograma polar) que nos indica para cada determinados grados por lo general 5, cual es la distancia al objetivo más cercano. Para cada sector k, el histograma polar h_k es calculado siguiendo la ecuación 1.2.

$$h_k = \sum_{i,j} m_{i,j} \tag{1.2}$$

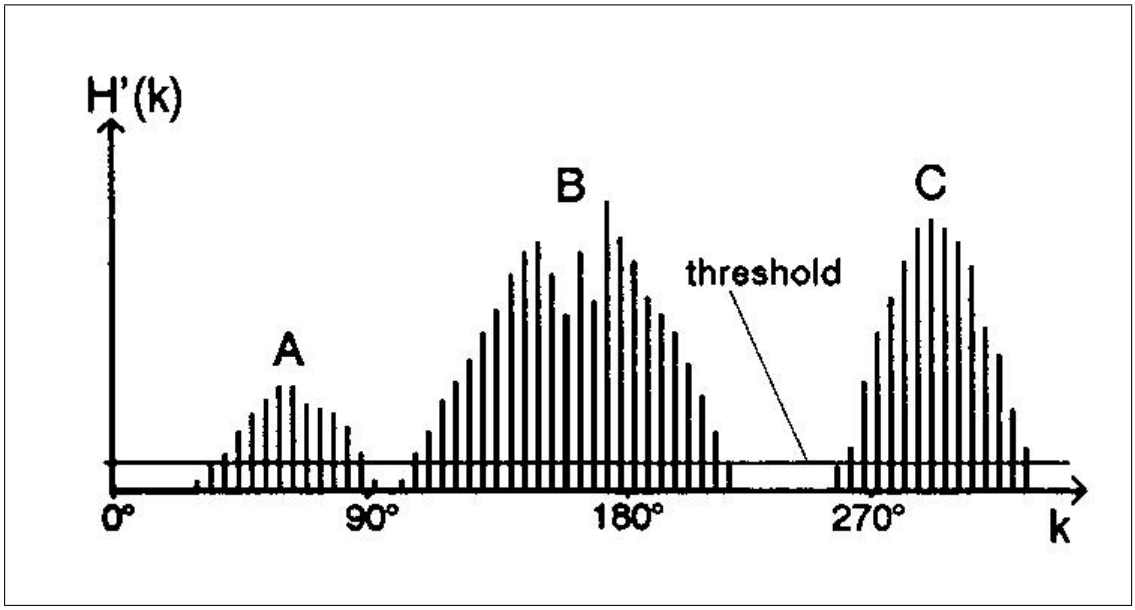

Figura 1.5: Ejemplo de histograma polar.

1.1.4.3. Tercer nivel: Comando de dirección

Para el cálculo del comando de dirección se tienen distintas opciones. La primera es, una vez encontradas las aperturas, las cuales se definen por los sectores límite k_r

k_l y según un umbral s_{max} (definido por el usuario considerando la distancia mínima a la que se considera como un obstáculo o como un sector libre) se pueden elegir distintas direcciones por las cuales pasar ya que no hay obstáculos cercanos.

$$c_d = \frac{k_r + k_l}{2} \tag{1.3}$$

$$c_r = k_r + \frac{s_{max}}{2} \tag{1.4}$$

$$c_l = k_l - \frac{s_{max}}{2} \tag{1.5}$$

$$c_t = k_t \, si \, k_t \epsilon [c_r, c_l] \tag{1.6}$$

Para la ecuación 1.3 se elige la dirección centrada, para la ecuación 1.4 se elige la dirección del lado derecho, para la ecuación 1.5 se elige la dirección del lado izquierdo y para la ecuación 1.6 con dirección al objetivo.

La otra opción es definir una función de costo de la siguiente forma y elegir el candidato con menor costo.

$$g(c) = \mu_1 \Delta(c, k_t) + \mu_2 \Delta(c, k_R) + \mu_3 \Delta(c, k_{R-1}) \tag{1.7}$$

Donde μ_k son valores positivos y el operador Δ representa la diferencia angular entre ambos valores, c es la dirección candidato, k_t es la dirección del objetivo y k_R son las direcciones del robot.

1.1.5. Esquemas líder - seguidor

Dentro de la literatura que trata el control bajo un esquema líder-seguidor, el lector se puede dar cuenta que la definición del esquema líder-seguidor no es homogénea y depende de cada autor, ya que son muchos los métodos presentados en textos científicos que incluyen las palabras, *"líder"*, y/o *"seguidor"*, en sus títulos o resúmenes, y cada uno de ellos adapta o define bajo sus propios términos y métodos la definición de este esquema. Para ejemplificar a continuación se presentan dos definiciones de diferentes autores con la intensión de hacer una comparación entre a lo

que se le puede denominar *esquema líder-seguidor*.

1. Uno de los robots es asignado como el *líder* y otro como *seguidor*. El líder sigue una trayectoria predefinida, mientras que los seguidores mantienen una posición y dirección con cierta distancia respecto al líder [20].

Dicha definición es muy parecida a lo que también es conocido como estructuras virtuales [19] dentro de los MRS (por sus siglas en inglés Multi Robot Sistems).

Una definición más general y con la posibilidad de complementarla con nuevas ideas es la siguiente:

2. Usando el enfoque *líder - seguidor (LFA)*, un robot actúa como el líder cuyo movimiento define el trayecto para el grupo de seguidores. Todos los seguidores usaran el trayecto definido para alcanzar una meta o lograr una tarea definida [18].

Existen más definiciones con ideas muy similares a las dos anteriores definiciones y que dependiendo del contexto es como implementan su propia definición del esquema líder-seguidor, pero siempre partiendo de la misma premisa de que un *"ente"*, debe dictar el comportamiento de *"otro u otros entes"*.

1.2. Marco Procedimental

El *Marco Procedimental* corresponde a los conceptos y técnicas implementadas a lo largo del desarrollo del proyecto, permitiendo definir e identificar los problemas, acotarlos y visualizarlos de manera sistemática con la finalidad de obtener así elementos fundamentales para la resolución de cada uno de los problemas identificados.

1.2.1. Sistema mecatrónico

El concepto de mecatrónica surge en Japón en los años setenta con la siguiente definición:

"La mecatrónica es la integración sinérgica de la ingeniería mecánica con la electrónica y el control informático inteligente en el diseño y la fabricación de productos y procesos industriales [12]".

Tal definición ha ido evolucionando y generando nuevos métodos de diseño, los cuales han permitido crear productos inteligentes. Actualmente es complicado dar una definición completa de lo que es la mecatrónica, sin embargo, se entiende que la mecatrónica abarca diversas disciplinas de la ingeniería y la cual permite caracterizar los elementos que la componen en: sistemas de modelos físicos, sensores y actuadores, señales, sistemas computacionales, software y adquisición de datos.

Con esto en mente un sistema mecatrónico para Bishop [4] se divide en:

- Máquinas mecatrónicas.
- Vehículos mecatrónicos.
- Mecatrónica de precisión.
- Micro mecatrónica.

Esto demuestra que la integración de un sistema con la electrónica comprende muchas clases de sistemas técnicos. En varios casos, la parte mecánica del proceso está acoplada con una parte de procesamiento eléctrico, térmico, termodinámico, químico o de información. Esto es especialmente cierto para los convertidores de energía como máquinas donde, además de la energía mecánica, aparecen otros tipos de energía. Por lo tanto, los sistemas mecatrónicos en un sentido más amplio comprenden procesos mecánicos y también no mecánicos. Sin embargo, la parte mecánica normalmente domina el sistema.

Esto es debido a que se requiere una energía auxiliar para cambiar las propiedades fijas de los sistemas mecánicos pasivos, mediante el avance o el control con retroalimentación [4].

1.2.2. Diseño mecatrónico

Parte de la identificación de un problema o necesidad es asignar *funciones* necesarias para resolver dichos problemas y asociarlas con las características iniciales con las que se cuentan en la etapa del *Diseño Conceptual*, en la que se analizan diferentes alternativas acerca de como satisfacer dichas funciones y en donde también, se evalúan para elegir la mejor alternativa. Posteriormente se formaliza ese concepto en la parte *Diseño Detallado* cuyo objetivo es tener una aproximación bastante cercana del producto final y cómo se debería construir. Siempre con las validaciones de todas las funciones propuestas.

El diseño mecatrónico no es lineal, sino concurrente y requiere de un desarrollo sistemático y el uso de herramientas modernas de diseño, como lo son según Bishop [4]:

- Herramientas para CAD/CAE.
- Procesos de modelado matemático.
- Simulaciones en tiempo real.
- Simulaciones de hardware en lazo cerrado.
- Prototipado de sistemas de control.

1.2.3. Proceso de análisis jerárquico (AHP)

El proceso de análisis jerárquico AHP *(Analytic Hierarchy Proccess)* es un proceso diseñado e implementado por Thomas L. Saaty [9]. Este proceso de análisis jerárquico

está diseñado con la finalidad de resolver problemas complejos con criterios múltiples, en este proceso el diseñador, subjetivamente proporciona evaluaciones respecto a la importancia entre cada criterio considerado y después se realiza una comparación directa entre las alternativas para obtener la mejor propuesta. El resultado de este procedimiento es una jerarquización que muestra los aspectos más relevantes de cada una de las alternativas, la cual además abre un campo de posibilidades de llevar a cabo el desarrollo e implementación.

Diseño del Sistema

2.1. Diseño Conceptual

En este capítulo se definen las necesidades y requerimientos del problema planteado, posteriormente el problema se divide en las principales áreas funcionales para resolver el problema y se plantean las diferentes formas de resolver cada una de las funciones donde estas son evaluadas y con base en esta evaluación un concepto final es obtenido.

2.1.1. Necesidades y requerimientos

Es necesario identificar el problema planteado a resolver y extraer las necesidades de la persona o personas que hayan planteado para dicho problema. Una vez identificadas dichas necesidades se prosigue a generar requerimientos los cuales surgen de traducir las necesidades en sentencias cuantificables, para que a partir de ellas se hagan propuestas de diseño las cuales posteriormente son evaluadas. Dado que el enfoque es para investigación, las necesidades no son tan estrictas y son pocas, sin embargo, al hacer una interpretación de estas, surgen muchos requerimientos, los cuales permite acotar el proyecto.

Necesidad	Interpretación
Tener dos robots móviles con ruedas para el desarrollo del proyecto SIP.	Selección de dos robots móviles terrestres para su compra o manufactura, con el objetivo de desarrollar un proyecto de investigación.
Los robots deberán ser usados para futuras aplicaciones y/o proyectos.	Robots con la documentación suficiente para poder ser replicados y reutilizados con una estructura modular, que permitan futuras adaptaciones .
Controlar dos robots móviles terrestres bajo un esquema líder−seguidor.	Coordinación de dos robots móviles bajo la implementación de un esquema de control líder−seguidor.
Los robots deberán evadir obstáculos de manera autónoma y mantener una comunicación continua entre ambos robots.	El robot líder deberá seguir una trayectoria seleccionada y evadir los obstáculos que puedan estar dentro de esta trayectoria, mientras que el robot seguidor deberá alcanzar los mismos puntos que el robot líder con base en la información proporcionada por este último.
Una interfaz humano máquina con la cual se puedan compartir datos del proceso.	Interfaz gráfica que le sea posible, iniciar y visualizar el progreso del proceso del control y seguimiento de trayectoria y la configuración líder−seguidor.

Cuadro 2.1: Tabla de Necesidades

Con la interpretación de la Tabla 2.1 se procede a plantear los requerimientos técnicos enlistados a continuacion:

- Control basado en el esquema líder-seguidor.
- Móvil terrestre, con ruedas.
- Batería con duración mínima de 10 minutos (uso continuo).
- Estructura estable para el soporte de circuitos y movimiento del robot móvil y para futuras aplicaciones.

- Material ligero.
- Ambiente controlado. (Obstáculos fijos y trayectorias predefinidas).
- Superficie lisa y plana.
- Lugar cerrado, con luz controlada.
- Tarjeta de procesamiento de información con arquitectura mínima de 32 bits, con lenguaje de programación de medio/alto nivel (Python, C++).
- Capacidad de soportar procesamiento en paralelo como lo es ROS.
- Capacidad para procesar información de distintos sensores y protocolos de comunicación, USB, WIFI, I^2C.
- Capacidad para adquirir datos confiables.
- Resolución menor a 5cm y 10° o su equivalente en coordenadas rectangulares.
- No susceptibles a ruido de agentes externos.
- Establecer un protocolo de comunicación inalámbrico con suficiente velocidad y alcance para asegurar una buena comunicación entre todos los agentes del sistema.
- Interfaz gráfica de rápido procesamiento capaz de establecer comunicación bidireccional con ambos robots móviles y desplegar los datos transmitidos por los robots.
- El movimiento deberá ser con pausas no mayores a 3s, es decir, la trayectoria deseada deberá fluir lo mejor posible.
- Respuesta rápida de los motores para establecer la velocidad deseada según la salida generada por el sistema de control.

- Establecimiento de algoritmos específicos para la maniobra de evasión de los obstáculos y el control de seguimiento de trayectoria basado el control líder-seguidor, tomando en cuenta las capacidades de las tarjetas de procesamiento.

Adicionalmente se tienen los siguientes requerimientos administrativos:

- Costo total menor a $50,000.00 MXN.
- Software de uso libre, para evitar el pago de licencias.
- Componentes fácilmente reemplazables.
- HMI en inglés y español.
- Duración máxima para finalizar el proyecto en 1 año.

2.1.2. Descomposición por áreas funcionales

Para tener una comprensión mayor del proyecto se procede a descomponerlo en áreas funcionales como se observa en la Figura 2.1, lo cual permite visualizar una tarea complicada en funciones más sencillas que contribuirán a resolver el problema principal.

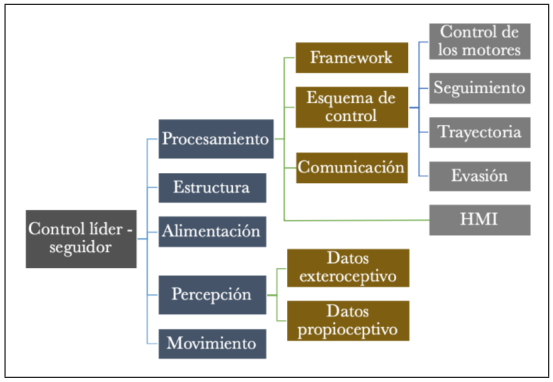

Figura 2.1: Áreas Funcionales propuestas.

2.1.2.1. Área Funcional 1: Estructura

La primera área de interés es una plataforma o estructura del móvil, donde alberga todas las herramientas que cumplirán con las funciones posteriores, esta estructura será la encargada de soportar todos los circuitos, sensores y procesadores capaces de satisfacer nuestras necesidades.

Es importante mencionar que no se realizará un diseño de ella, sino que se pretenden analizar las propuestas existentes comercialmente o con acceso a un manual para su construcción. Nuestra razón es que al estar en el mercado sabemos que soporta todos los elementos que conlleva un móvil, ya que se hace la selección orientada a robots móviles que cumplan los requisitos de la mejor manera, con un proceso de diseño concurrente, una línea de producción, un sistema de pruebas, documentación y al ser comerciales con costos atractivos. Para el desarrollo de este proyecto no es necesario algo realmente robusto hablando estructuralmente, pues el objetivo es la implementación del esquema líder-seguidor.

Dentro de los móviles que seleccionamos cada uno posee características específicas, aunque similares, es ahí donde entrará la metodología mecatrónica para seleccionar una opción adecuada que se encuentre disponible en el mercado o que tenga documentación para construirse, cubriendo los requerimientos antes mencionados, como lo es el límite de presupuesto.

2.1.2.2. Área Funcional 2: Procesamiento

La localización, detección y evasión de obstáculos, la comunicación entre robots e interfaz (HMI), el seguimiento de trayectorias, así como la sinergia de las demás áreas funcionales están englobadas en el *Área de Procesamiento* también llamada *"Pensamiento propio del robot"*. Está es el área encargada de lograr el objetivo principal de este trabajo con la ayuda de tarjetas de procesamiento. Como entrada se manejan los datos obtenidos por la función percepción. El control, en términos de un diagrama IDEF0 es la selección del algoritmo a usar para cumplir cada una de

estas funciones. Adicionalmente se necesitará establecer algún tipo de comunicación preferentemente inalámbrico entre ambos móviles y la computadora central, lo cual permitirá al robot seguidor conocer los puntos de interés por alcanzar, proporcionados por el líder. Así mismo, monitorear diversos parámetros durante la ejecución del esquema líder-seguidor.

2.1.2.3. Área Funcional 3: Percepción

La percepción es el área funcional encargada de la recabación de datos internos y externos de cada robot (datos propioceptivos y exteroceptivos) para posteriormente procesar dicha información y extraer los datos de interés, y así llevar acabo la localización de los móviles, la detección de los obstáculos, la detección del entorno y la evasión de los obstáculos. Para llevar a cabo la recabación de datos se implementó el uso de dispositivos electrónicos (sensores) capaces de variar sus propiedades internas con base en cambios de magnitudes físicas como distancias y ángulos de los objetos del entorno, aceleraciones y desplazamientos.

2.1.2.4. Área Funcional 4: Alimentación

La alimentación es el área funcional encargada de suministrar la energía a cada móvil por la tanto es necesario que la fuente de alimentación pueda ser portable y que a la vez pueda suministrar una potencia suficiente al móvil, principalmente por los motores que son los que mayor cantidad de energía consumen, ya que llevan una carga. Asimismo al ser portable debe de ser una batería con una capacidad de energía considerable para que los móviles puedan realizar el seguimiento de la trayectoria totalmente sin dejarla a medias o en algún momento interferir con el proceso por falta de energía, también cabe recalcar que el sistema deberá tener una alerta de batería baja.

2.1.2.5. Área Funcional 5: Movimiento

La parte del acto o movimiento, tiene como característica fundamental la elección del tipo de locomoción ya sea Ackerman, triciclo, diferencial, entre otras. Esta locomoción tendrá como requisito que sea comercial o capaz de construirse con un manual para de esta manera poder satisfacer los requerimientos del proyecto. Otra de las herramientas para lograr esta función son los motores y sus controladores, las cuales son una parte fundamental para lograr tener un funcionamiento adecuado, refiriéndonos a la capacidad de transformar los comandos generados por los esquemas de control en acciones que además deberán tener un control de velocidad y dirección lo suficientemente preciso para asegurar que los móviles sigan la trayectoria de una manera correcta y consistente sin tener problemas mecánicos en los motores como lo es que no soporten la potencia debido a una alta velocidad.

2.1.3. Características y/o necesidades principales de cada área funcional

Ya propuestas las áreas funcionales se procede a enlistar las características y criterios a considerar según las necesidades y requerimientos haciendo énfasis en los elementos de cada área funcional.

- Percepción del entorno
  - Alta resolución en mediciones.
  - Poco costo computacional.
  - Sensores con largo alcance en sus mediciones.
- Configuración (Configuración y motores)
  - Maniobrabilidad.
  - Facilidad de implementación y mantenimiento.
  - Necesidad de un controlador externo (etapa de potencia).

- Costo.
- Consumo energético.
- Tamaño.
- Bajo peso
- Compatibilidad con diversos tipos de control (Torque, Velocidad, Posición).
- Cantidad de motores.

- Procesamiento
  - Velocidad de procesamiento.
  - Arquitectura.
  - Costo.
  - Compatibilidad con framework.
- Software/Framework
  - Paralelismo inherente para tareas concurrentes.
  - Con herramientas de simulación, visualización de información y desarrollo de interfaces gráficas.
  - Facilidad de programación, física y digitalmente.
  - Compatible con las tarjetas de procesamiento.
  - Posibilidad de realizar diagnósticos sobre posibles fallas en el sistema.
- Comunicación
  - Bidireccional.
  - Alta velocidad de transmisión.
  - Costo.
  - Sin necesidad de componentes adicionales a las tarjetas de procesamiento.

- Estructura
  - Modular.
  - Adaptable a diversos sensores.
  - Compacta.
  - Ligera.

2.1.4. Características principales de los robots considerados. ROSbot, Turtlebot, TX1 "Jet" Robot Kit

Para la selección de los robots móviles se hizo una búsqueda contemplando los criterios antes mencionados, los más relevantes y dentro del presupuesto son las siguientes tres propuestas. Para cada propuesta se da una breve descripción.

2.1.4.1. Rosbot

Figura 2.2: Robot móvil Rosbot [37]

En la Figura 2.2 se muestra el Rosbot el cual es un robot autónomo basado en el sistema operativo ROS (Robotic Operating System) que cuenta con un sensor LIDAR 360º (RPLIDAR A2) y una cámara RGBD. Para el procesamiento y control del móvil se utilizan dos unidades centrales de control; la primera es un microcontrolador STM32F4 con instrucciones de un DSP (M4) el cual se encarga del procesamiento de la información proporcionada por los sensores, hacer la retroalimentación de los encoders, guardar en memoria las distancias de los sensores conectados a esta tarjeta y cualquier adquisición que ese necesite en tiempo real. El segundo controlador es

una tarjeta Core2 con Asus tinker Board la cual se encarga de ejecutar el sistema operativo ROS y así poder realizar el procesamiento de todos los algoritmos de alto nivel, asimismo está tarjeta le da la posibilidad de comunicarse inalámbricamente con otros dispositivos que ejecuten ROS.

Precio del robot: $1299.00 USD

2.1.4.2. Turtlebot3 Burger

Figura 2.3: Robot móvil Turtlebot3 Burger [21]

En la Figura 2.3 se muestra el Turtlebot3 Burger el cual es un robot modular de bajo costo y software abierto basado en el sistema operativo ROS. Este robot es el que garantiza mejor compatibilidad con el sistema operativo ROS, ya que no es necesario realizar muchos modelos de este pues ya se encuentran disponibles. Cuenta con dos tarjetas de procesamiento y una interfaz de simulación y monitoreo; la primera de ella es una tarjeta Raspberry Pi 3 modelo B y la segunda es una tarjeta Open CR ARM Cortex 32 bits. Pose un sensor LIDAR 360º (LDS-01), una batería de 1800 mAh, un giroscopio, con un acelerómetro y un magnetómetro de 3 ejes cada uno. Para el control de la posición y velocidad del robot tiene un par motores inteligentes Dynamixel XL430 y una estructura escalable y modular con la cual se pueden extender sus aplicaciones. Uno de los objetivos primordiales del Turtlebot3 Burger es reducir drásticamente el tamaño de la plataforma y bajar el precio sin tener que sacrificar la calidad o limitar sus funciones de respuesta ofertando al mismo tiempo capacidad de expansión (escalabilidad).

Precio del robot: $549.00 USD

2.1.4.3. TX1 "Jet" Robot Kit

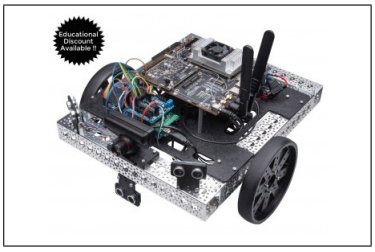

Figura 2.4: Robot móvil TX1 "Jet" [13]

En la Figura 2.4 se muestra el robot TX1 "Jet" el cual es un robot inteligente y autónomo que utiliza componentes Acotobotics y se basa en la potente plataforma de desarrollo integrada NVIDIA Jetson. El cerebro de Jet se basa en el SoC NVIDIA Tegra® y utiliza los mismos núcleos de computacionales NVIDIA diseñados para las supercomputadoras de todo el mundo. Esto proporciona a la computadora una visión computacional intensiva en cálculos, inteligencia artificial (AI) y capacidades de auto manejo en un paquete de bajo costo. Como "Jet" utiliza el sistema operativo de robot (ROS) para la abstracción, se puede escalar desde proyectos de desarrollo hasta aplicaciones industriales específicas en entornos complejos. El Jetson TX1 es un supercomputador que presenta la nueva arquitectura NVIDIA Maxwell, 256 núcleos NVIDIA CUDA®, CPU de 64 bits y una eficiencia energética competitiva. Sin embargo no posee tanta flexibilidad estructuralmente hablando para añadir nuevos elementos mecánicos.

Precio del robot: $1,165.99 USD

2.1.5. Tablas de selección de los robots móviles

Los robots móviles considerados fueron separados acorde a las áreas funcionales antes propuestas y los elementos que componen cada área funcional fueron sometidos

a una comparación directa con el fin de obtener el mejor candidato, dando como resultado las siguientes tablas de selección:

Procesamiento				
	Velocidad del CPU	Compatibilidad con sensores y librerías	Documentación	Total
Rosbot	0.1437	0.2500	0.4000	0.2503
Tb3	0.0747	0.5000	0.3500	0.3136
Jet	0.7816	0.2500	0.2500	0.4361
Ponderación	0.3500	0.4000	0.2500	

Cuadro 2.2: Tabla de Selección: Procesamiento.

Percepción				
	Resolución de sensores	Costo de procesamiento	Alcance	Total
Rosbot	0.2857	0.1320	0.5350	0.3672
Tb3	0.57142	0.3396	0.3439	0.4572
Jet	0.1428	0.5283	0.1210	0.1755
Ponderación	0.5	0.1071	0.3928	

Cuadro 2.3: Tabla de Selección: Percepción.

Estructura				
	Modular	Adaptabilidad a nuevos sensores	Peso	Total
Rosbot	0.1915	0.2857	0.2857	0.2677
Tb3	0.5957	0.5714	0.5714	0.5761
Jet	0.2128	0.1429	0.1429	0.1562
Ponderación	0.1915	0.2128	0.5957	

Cuadro 2.4: Tabla de Selección: Estructura.

Motores y Controladores				
	Corriente Máxima	Modos de operación en el controlador	Torque	total
Rosbot	0.1915	0.2500	0.5000	0.3050
Tb3	0.5957	0.5000	0.2500	0.4525
Jet	0.2128	0.2500	0.2500	0.2426
Ponderación	0.2000	0.5333	0.2667	

Cuadro 2.5: Tabla de Selección: Motores y Controladores.

De las tablas de selección se observa que la mejor opción para la mayoría de los casos resultó ser el *Turtlebot3 Burger*, con lo que se concluye con la parte de selección de los robots móviles. En algunos casos no fue necesaria la realización de tablas ya que poseían las mismas características, como : configuración, framework y comunicación. Para mayor detalle de como se obtuvieron los anteriores valores revisar en anexos.

2.1.6. Criterios para el diseño y selecciones de interfaces gráficas

2.1.6.1. Criterios a evaluar

Para llevar a cabo la implementación de la interfaz gráfica se decidió primero considerar sus requerimientos técnicos y necesidades, posteriormente para lograr una correcta realización de la interfaz se investigaron diferentes criterios para una interfaz gráfica dentro de los cuales seleccionamos los más importantes, estos criterios para el diseño y selección de interfaces gráficas son mencionados a continuación.

2.1.6.2. 1. Amabilidad

La amabilidad se refiere a la facilidad de uso, aunque, dicho parámetro sea relativo de acuerdo al tipo de usuario, se considera que una interfaz es amigable cuando le es más fácil de usar para un mayor número de una población objetivo.

2.1.6.3. 2. Interactividad

Una interfaz es interactiva cuando le proporciona al usuario un sentido de comunicación deseado, en otras palabras al usuario le será fácil plasmar su idea o propósito en la interfaz sin limitaciones.

2.1.6.4. 3. Redundancia óptima en el proceso de comunicación

Eficiencia a la hora de comunicarse entre distintos modos de comunicación como pueden ser visuales, verbales, auditivos, entre otros.

2.1.6.5. 4. Eficiencia y Utilidad

El objetivo principal de una interfaz gráfica es hacer que las ideas, los conocimientos y la información sean comprensibles y útiles.

2.1.6.6. Tabla de Criterios

Criterios	Peso	Parámetro	Opción 1			Opción 2			Opción 3			Opción 4		
			M	C	V	M	C	V	M	C	V	M	C	V
Amabilidad	0.3	Facilidad de uso		9	2.7		8	2.4		6	1.8		5	1.5
Interactividad	0.2	Sentido de comunicación deseado		9	1.8		8	1.6		4	0.8		5	1
Redundancia óptima en el proceso de comunicación	0.1	Integración de distintos modos de comunicación		8	0.8		7	0.7		6	0.6		7	0.7
Eficiencia y Utilidad	0.4	Información comprensible y útil		9	3.6		9	3.6		8	3.2		8	3.2
			8.9			8.3			6.4			6.4		

Cuadro 2.6: Tabla de Selección de las diferentes opciones de interfaz.

2.1.6.7. Lista de opciones de interfaz

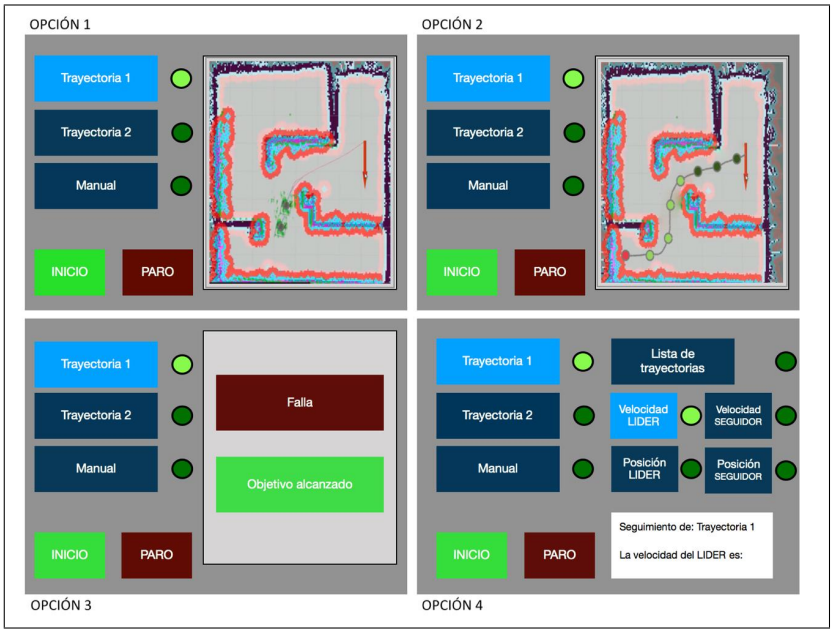

Figura 2.5: Opciones de interfaces gráficas

2.1.6.8. Diseño de arquitectura general del software

Como parte del diseño conceptual también se contemplan las distintas arquitecturas usadas en este tipo de proyectos, donde existen dos o más elementos comunicados y capaces de realizar procesos o tomar decisiones. Los tipos de arquitectura pueden ser:

- Centralizado.
- Distribuido.
- Descentralizado.
- Una combinación entre los anteriores.

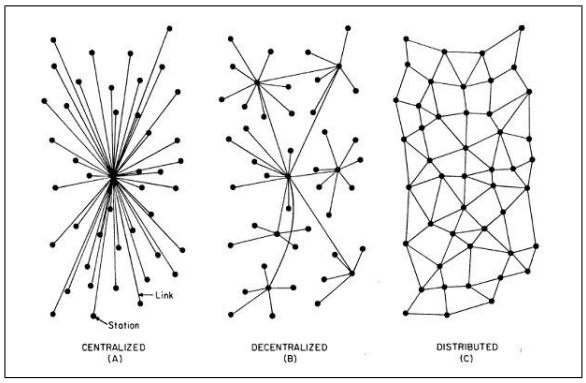

Figura 2.6: Tipos de Arquitecturas

A continuación, se incluye la Tabla 2.7 donde se mencionan las ventajas y desventajas de cada uno de estos paradigmas.

	Centralizado	Descentralizado	Distribuido
Características	▪ Fácil de implementar. ▪ Un solo servidor. ▪ Difícil escalar. ▪ Un solo punto de falla	▪ No hay una sola autoridad. ▪ Todos los nodos son iguales en la red en términos de autoridad. ▪ No importa si algunos nodos son suprimidos, el sistema sigue funcionando.	▪ Sistemas distribuidos, con poder de cómputo a través de los múltiples nodos, en lugar de 1.

Cuadro 2.7: Comparación entre Arquitecturas

Sin embargo, para nuestro caso donde solamente interactúan 2 robots y una computadora, únicamente para dar inicio a las trayectorias y modos de operación, las ventajas y desventajas que ofrecen estos paradigmas no se logran apreciar o reflejar completamente debido a la escala del proyecto y número de robots.

En nuestra propuesta de solución y adaptándose a los robots móviles seleccionados una opción adecuada es aplicar una arquitectura distribuida como se muestra en la Figura 2.7, ya que con este paradigma se realizará el procesamiento en cada uno de los robots móviles, así como en la computadora de monitoreo. Cada elemento dentro de nuestro sistema procesará partes de código específicas, relacionadas con sus acciones de manera autónoma sin necesidad de consultar todas sus acciones con un nodo central.

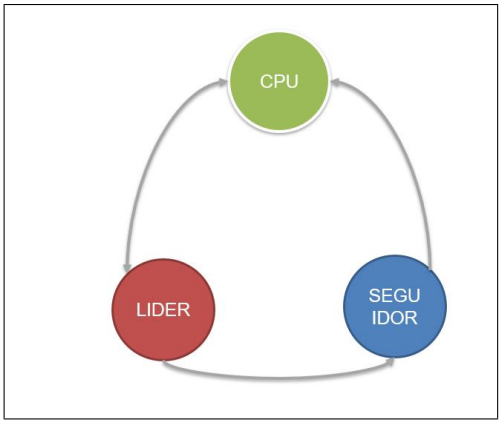

Figura 2.7: Arquitectura de software propuesta para el proyecto

Adicionalmente cada robot móvil tendrá a su vez distintos nodos ejecutándose para cumplir con sus funciones específicas las cuales se detallan en la *Sección 2.2 "Diseño Detallado"*.

2.1.6.9. Concepto Final

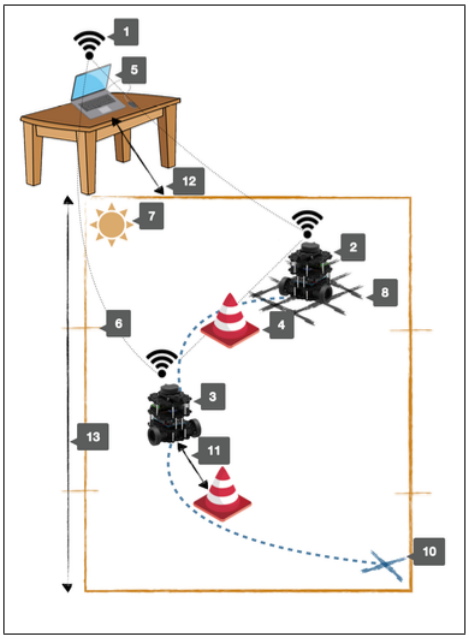

Figura 2.8: Concepto final

1. Comunicación WiFi entre la interfaz humano máquina y los robots líder y seguidor.
2. Robot Seguidor con sensores lidar y acelerómetro.
3. Robot Líder manipulado directamente desde la interfaz.
4. 2 obstáculos con forma regular de dimensiones cercanas a la de una caja de zapatos.
5. Computadora con la interfaz y sistema operativo ROS para comunicarse con los robots.
6. Marcas de distintas formas para una mejor identificación con el sensor lidar.
7. Ambiente de luz controlado favorable para la operación del sensor lidar.

8. Suelo plano sin ningún objeto móvil y con coeficiente de fricción mayor a 0.4.
9. Trayectoria fijada por medio de la interfaz la cual sigue tanto el líder como el seguidor.
10. Punto de objetivo propuesto por el usuario (la trayectoria llega a puntos meta).
11. Distancia mínima de separación entre los objetos o móviles de al menos 10 cm.
12. Cercanía entre la computadora y el área de prueba de no más de 2 metros.
13. Dimensiones del área de trabajo no mayores a 6 metros X 6 metros.

2.2. Diseño Detallado

En este capítulo se analiza con más profundidad cada una de las áreas funcionales previamente mencionadas junto con las características y datos técnicos de cada uno de los elementos que conforman las distintas áreas funcionales. De igual manera, se aborda la integración de cada una de las áreas funcionales con el sistema final que forman a los robots, tal y como se puede observar en la Figura 2.9.

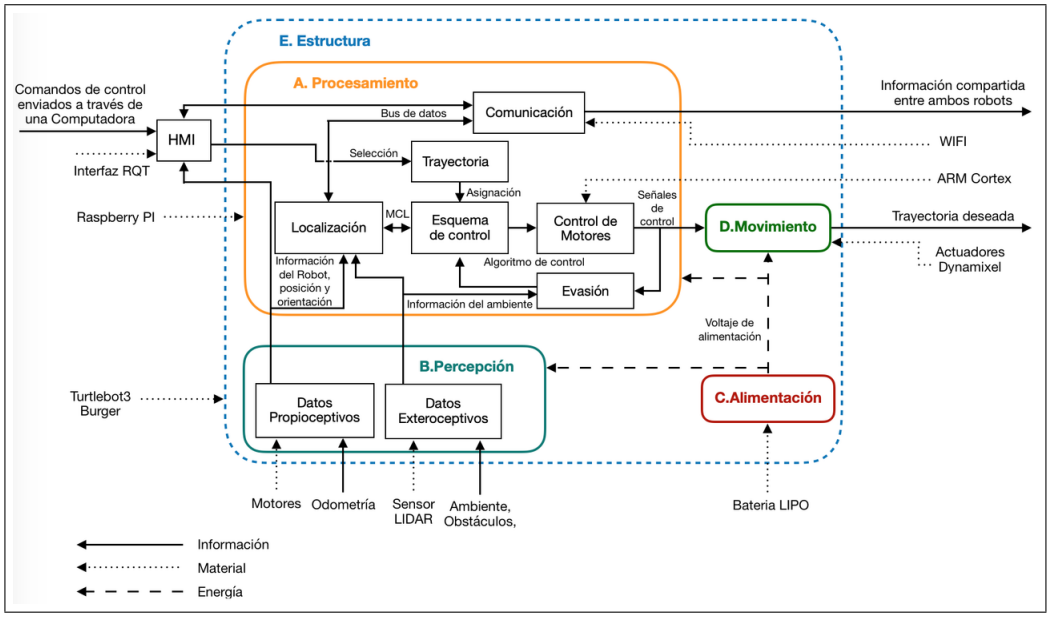

Figura 2.9: Integración de todas las Áreas Funcionales.

La Figura 2.9 muestra la interacción de las distintas áreas funcionales aplicadas a cada uno de los robots, así como su implementación con los distintos componentes los cuales en conjunto conforman al sistema robótico líder-seguidor que se va a utilizar para llevar acabo la coordinación, navegación y ejecución de tareas proporcionadas por un usuario.

Ajustándose a la Figura 2.9 la primera Área Funcional que hay que analizar es la de *Estructura* que tiene como principal objetivo albergar los distintos componentes que conforman a los robots, es decir, es la encargada de alojar todas las partes físicas

de las distintas áreas funcionales pertinentes a los robots, como por ejemplo, en el caso del Procesamiento, se necesita que dentro de la estructura exista un lugar donde establecer y resguardar las tarjetas de procesamiento *(Raspberry PI y ARM Cortex)* y que estás vayan a estar seguras para que no sufran ningún desperfecto que se pudiera traducir en algún error en los robots.

2.2.1. Área Funcional 1: Estructura (Soporte de componentes)

La Estructura está directamente relacionada con el tamaño y el peso de las distintas piezas que conforman los robots, así como el tipo de locomoción, en donde, los robots móviles Turtlebot3 Burger se acoplan perfectamente a estas necesidades y previos requisitos ya antes mencionados, gracias a que cuentan con una estructura modular y compacta que hace posible la distribución de los distintos componentes del robot a través de sus diferentes plataformas y cuya estructura es capaz de soportar hasta 15 kg de peso quedando libres 14kg para futuras implementaciones. Asimismo cabe mencionar que la estructura en conjunto con la batería LiPo y el sensor LIDAR es de apenas 1 kg [27] [2] [3].

Por otro lado, al tener una estructura modular dentro del robot, y que el Turtlebot3 Burger este clasificado como un robot móvil de Código Abierto, con flexibilidad en el diseño mecánico, se puede adaptar a cualquier tipo de locomoción, aunque el desarrollo del proyecto se llevará acabo con la configuración con la que ya viene integrada, que es de tipo diferencial sobre la cual en la sección de Movimiento se hablará a detalle.

2.2.1.1. Elementos primordiales de la estructura

Chapas

Los pisos o chapas que conforman los diferentes niveles del Turtlebot3 dentro de la estructura, son como se muestran en la Figura 2.10, los cuales están fabricados de plástico ABS y tienen distintos orificios que sirven para ahorrar material y con ello

peso, y que por otra parte, sirven para adaptar diferentes componentes, haciendo los pisos de la estructura universales para cualquier requerimiento, adaptación o aplicación.

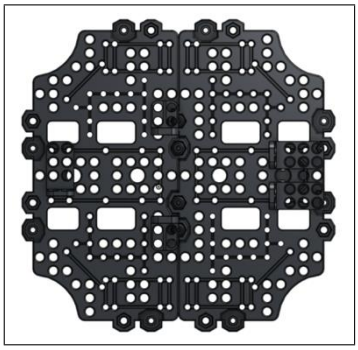

Figura 2.10: Ensamble de 2 pisos o chapas en forma de celda (base estructural).

Al ser un robot denominado multiplataforma o multicapa como se observa en la Figura 2.11 ofrece entre otras bondades un mejor uso del espacio y mejor maniobrabilidad a la hora de ensamblar los componentes que conforman al robot. Cuenta con 4 plataformas las cuales distribuyen los distintos componentes en cada una de estas y sus chapas o pisos en forma de celdas permiten la colocación y posterior implementación de diferentes componentes y funciones de acuerdo a cada tarea ejecutada por el robot que sea requerida.

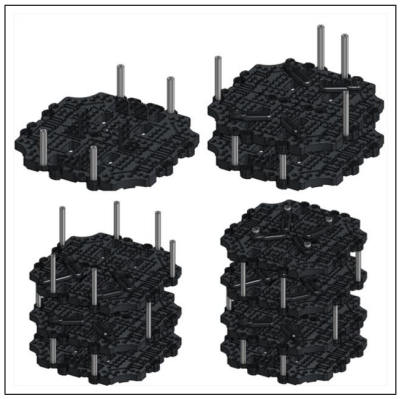

Figura 2.11: Estructura tipo multiplataforma.

2.2.1.2. Elementos utilizados en cada piso

Elementos generales

Para el ensamble de cada piso de la estructura, se utilizaron los siguientes elementos:

- Dos chapas tipo "waffle"(ver Figura 2.10).
- 16 tornillos M3 de 4mm de longitud.
- 16 tuercas M3 de 8mm de longitud.

Se colocaron 4 tornillos M3 de 8mm en la parte superior de la chapa, y por la parte inferior para ajustar se usaron 4 tuercas M3 como se muestra en la Figura 2.11.

Elementos específicos por cada piso

1. Piso 1
   a) 2 Chapas tipo "waffle".
   b) 8 Tornillos de 4 mm de longitud.
   c) 4 Tornillos de 6 mm de longitud.
   d) 8 Tornillos de 12 mm de longitud.
   e) 8 Tornillos de 8 mm de longitud.
   f) 4 Tuercas.
   g) 4 Soportes de 35 mm de longitud.
   h) 2 Motores DYNAMIXEL (XL430).
   i) 2 Cables DYNAMIXEL a OpenCR.
   j) 1 Rueda loca.
   k) 2 Llantas.
   l) 2 Bandas.

m) 1 bateria Li-Po.

n) 10 Remaches.

ñ) 5 Ménsulas.

2. Piso 2

a) 2 Chapas tipo “waffle”.

b) 4 Tornillos de 8 mm de longitud.

c) 8 Tornillos de 12 mm de longitud.

d) 12 Tornillos de 8 mm de longitud.

e) 4 Soportes de 45 mm de longitud.

f) 4 Remaches.

g) 5 Ménsulas.

h) 4 Soportes de PCB.

i) 4 Tuercas.

j) 1 Tarjeta OpenCR1.0.

k) 1 Cable de batería Li-Po.

3. Piso 3

a) 2 Chapas tipo “waffle”.

b) 8 Tornillos de 8 mm de longitud.

c) 12 Tornillos M3 de 8 mm de longitud.

d) 2 Remaches.

e) 8 Tuercas M2.5.

f) 4 Tuercas M3.

g) 6 Soportes M3 de 45 mm de longitud.

h) 1 Conector usb a lds.

i) 4 Soportes de PCB.

j) 1 Cable de alimentación Raspberry Pi 3.

k) 2 Cables usb a micro-usb.

l) 1 Chapa adaptadora.

m) 1 Tarjeta Raspberry Pi 3.

4. Piso 4

a) 2 Chapas tipo “waffle”.

b) 4 Tornillos M2.5 de 8 mm de longitud.

c) 4 Tornillos M2.5 de 16 mm de longitud.

d) 10 Tornillos M3 de 8 mm de longitud.

e) 4 Espaciadores.

f) 8 Tuercas M2.5.

g) 4 Tuercas M3.

h) 4 Soportes de PCB.

i) 1 Sensor Lidar.

2.2.1.3. Dimensiones

Como ya se ha mencionado el Turtlebot3, es un robot compacto con apenas una longitud (L) de 138 mm, una anchura (W) de 178 mm y una altura (H) de 192 mm tomando como referencia las llantas, tal como se muestra en la Figura 2.12.

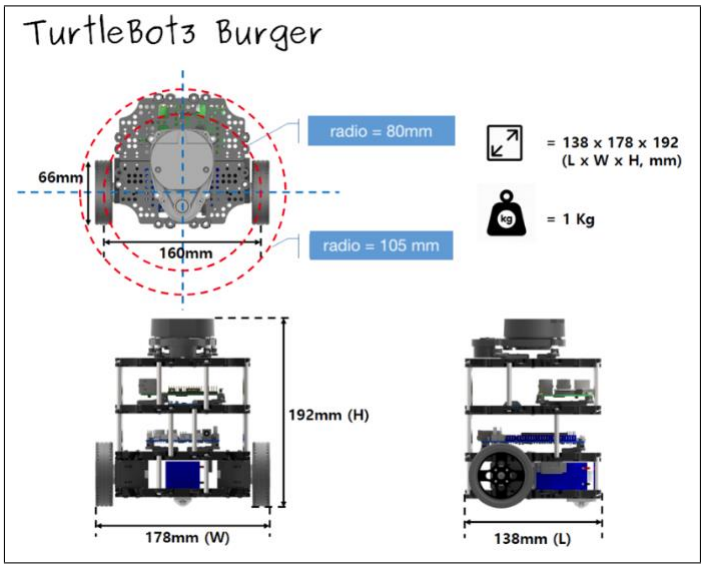

Figura 2.12: Dimensiones del Turtlebot3 [2].

2.2.2. Área Funcional 2: Procesamiento (Comunicación de componentes)

El Área Funcional de procesamiento es de suma importancia de acuerdo con el diagrama de la integración de las distintas áreas funcionales (ver la Figura 2.1), puesto que además de ser el área funcional con mayor número de subáreas, es una de las partes principales donde recae todo el desarrollo del esquema de control líder-seguidor. Para desarrollar esta área funcional se dividió en tres secciones con el objetivo de volver más transparente el proceso de diseño.

La primera parte consiste en la descripción de las tarjetas encargadas del procesamiento, razón por la cual se mencionan las distintas tarjetas y sus características más relevantes, es decir se define el recurso disponible para realizar el procesamiento.

En la segunda parte se definen las funciones que deberán cumplir cada una de las tarjetas, y finalmente en la tercera parte se describe como se están proponiendo cumplir dichas funciones.

2.2.2.1. Primera parte: Dispositivos destinados para el procesamiento

Recordando el ciclo "Ve, piensa, actúa", el procesamiento funciona como la parte del pensar, es a través de este pensar que los robots podrán ejecutar las acciones que el procesamiento solicite. Son cuatro los dispositivos donde existirá o se llevará a cabo el procesamiento.

1. Líder: Raspberry Pi 3 Model B .

2. Líder: OpenCR.

3. Seguidor: Raspberry Pi 3 Model B .

4. Seguidor: OpenCR.

5. Computadora de monitoreo.

Las características técnicas más relevantes de cada una de las tarjetas se muestran en las Tablas 2.8 y 2.11.

Elemento	Características
Microcontrolador	1. STM32F746ZGT6 / 32-bit ARM Cortex-M7 con FPU (216MHz, 462DMIPS).
Sensores	1. Giroscopio 3 ejes. 2. Acelerómetro 3 ejes. 3. Magnetómetro 3 ejes (MPU9250).
Programador	1. ARM Cortex 10 pines JTAG/SWD conector serial USB. 2. Dispositivo de Actualización de Firmware (Device Firmware Upgrade - DFU).
Digitales I/O	1. 32 pines (L 14, R 18) con conectividad a la interfaz Arduino. 2. Dispositivo de Actualización de Firmware (Device Firmware Upgrade - DFU). 3. 5 pines OLLO x 4. 4. GPIO x 18 pines. 5. PWM x 6. 6. I2C x 1. 7. SPI x 1.

Cuadro 2.8: Características de la Tarjeta ARM Cortex M7 [32]

Elemento	Características
Entradas analógicas	1. 6 canales de Convertidores Analógicos - Digitales de 12 bits de resolución.
Puertos de comunicación	1. USB x 1 (Micro-B USB connector/USB 2.0/Host/Peripheral/OTG). 2. TTL x 3 (B3B-EH-A / Dynamixel). 3. RS485 x 3 (B4B-EH-A / Dynamixel). 4. CAN x 1 (20010WS-04).
Botones y leds	1. LD2 (rojo/verder) : comunicación USB. 2. LED de usuario x 4 : LD3 (rojo), LD4 (verde), LD5 (azul). 3. Boton de usuario x 2.

Cuadro 2.9: Características de la Tarjeta ARM Cortex M7 [32].

Elemento	Características
Alimentación	1. Fuente de entrada externa. 2. 5 V (USB VBUS), 7-24 V (Bateria o SMPS). 3. Bateria por defecto : LI-PO 11.1V 1,800mAh 19.98Wh. 4. SMPS por defecto: 12V 5A. 5. Fuente de salida externa. 6. 12V max 5A(SMW250-02), 5V max 4A(5267-02A), 3.3V@800mA(20010WS-02). 7. Puerto de bateria externa para el reloj de tiempo real (RTC Real Time Clock) (Molex 53047-0210). 8. LED de encendido: LD1 (red, 3.3 V encendido). 9. Botón de reinicio x 1 (for power reset of board). 10. Botón de encendido o apagado x 1.
Dimensiones	1. 105(W) X 75(D) mm.
Masa	1. 60 gramos.

Cuadro 2.10: Características de la Tarjeta ARM Cortex M7 [32].

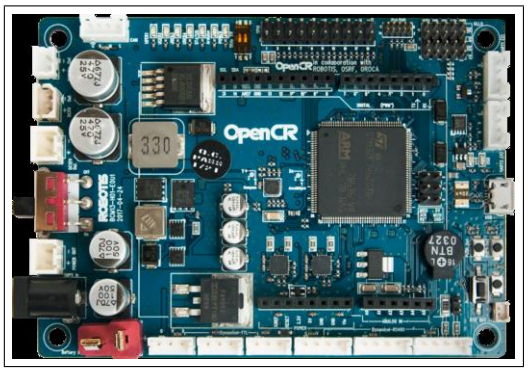

Figura 2.13: Tarjeta ARM Cortex M7 [32].

Características
Quad Core 1.2GHz Broadcom BCM2837 64bit CPU
1GB RAM
BCM43438 wireless LAN y Bluetooth Low Energy (BLE)
Puerto Ethernet
40-pin GPIO para expansión
4 puertos USB 2.0
Salida de audio de 4 polos y puerto de video compositivo
HDMI
Puerto para Raspberry Pi camera
DSI display para conectar una pantalla táctil Raspberry Pi
Puerto Micro SD para el sistema operativo
Fuente de poder Micro USB hasta 2.5 A

Cuadro 2.11: Características de la tarjeta Raspberry PI 3, Modelo B [30].

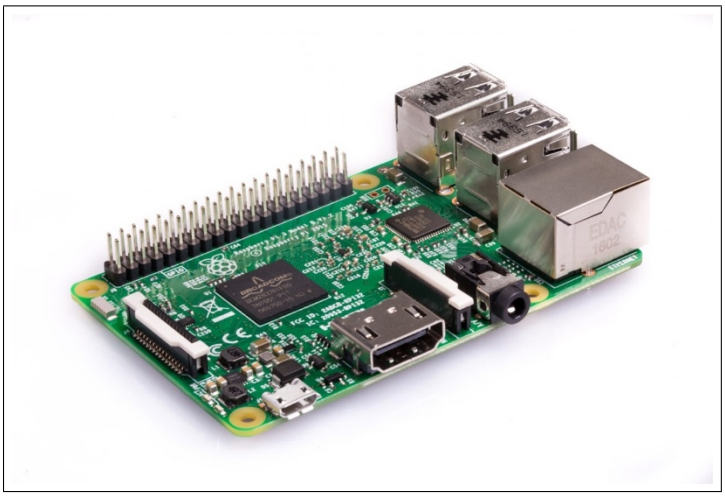

Figura 2.14: Tarjeta Raspberry PI 3, Modelo B [30].

2.2.2.2. Segunda parte: Funciones a desarrollar en cada tarjeta

Una vez definidos que elementos son necesarios, sigue hablar que realizarán, es decir que áreas funcionales están relacionadas con las tarjetas de procesamiento y de que se encargan dichas áreas funcionales. El procesamiento es la principal área funcional, pero de ella se derivan las siguientes subáreas, las cuales se recuperan del diseño conceptual:

Framework: Es una herramienta de software que se ejecuta en cada una de las unidades de procesamiento para facilitar ciertos aspectos como la manera en que se procesan los datos y el funcionamiento del sistema en general.

Esquemas de control: Son los encargados de definir la dirección de los turtlebot3 para mantener la trayectoria deseada, el esquema líder-seguidor y la evasión de obstáculos. Esta a su vez se divide en seguimiento de trayectoria, seguimiento al líder, evasión de obstáculos y control de motores.

Comunicación: Es la encargada de transmitir mensaje entre los distintivos agentes.

Estas áreas funcionales estarán distribuidas en cada una de las tarjetas siguiendo

la lista a continuación.

- RBP Líder.
  - Seguimiento de trayectoria.
  - Evasión de obstáculos.
- Open CR Líder.
  - Control de motores.
- RBP Seguidor.
  - Seguimiento al líder.
- Open CR Seguidor.
  - Control de motores.

Ya clarificadas las funciones específicas de cada una de las tarjetas, el siguiente paso es definir "como se cumplirán".

2.2.2.3. Tercera parte: Diseño del software

Seguimiento de trayectorias Para el seguimiento de trayectorias se optó por seleccionar la matriz de fuerzas [3] como método de seguimiento, es así como se le denomina matriz de fuerzas a un mapa de vectores que cubre toda el área del lugar donde el líder se desplazará, este algoritmo resulta muy adecuado debido a que se tiene la opción de usar una librería de ROS para localización del móvil en ambientes cerrados. Dicha librería hace uso de un algoritmo probabilístico llamado AMCL el cual se detalla en el marco teórico. Este algoritmo permite conocer la posición de los turtlebot3 con una incertidumbre de $2.5cm^2$. Este hecho facilita el uso de la matriz de fuerzas, ya que una vez determinadas las dimensiones del área de pruebas se procede a definir la trayectoria a seguir por el turtlebot3 líder, esto se hace discretizando

la trayectoria propuesta y listando dichos puntos en un archivo con extensión .txt siguiendo uno de los formatos creados para una de las competencias de DARPA [7]. Posteriormente esta serie de puntos son procesados por el algoritmo que genera la matriz de fuerzas como se muestra en la Figura 2.15 y es guardado en un archivo con extensión npy el cual puede ser fácilmente leído e interpretado por Python.

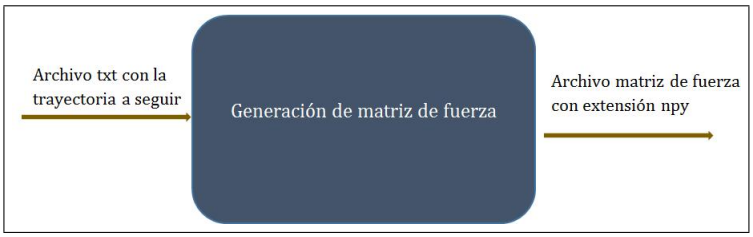

Figura 2.15: Entrada y salida del programa que genera la matriz de fuerza.

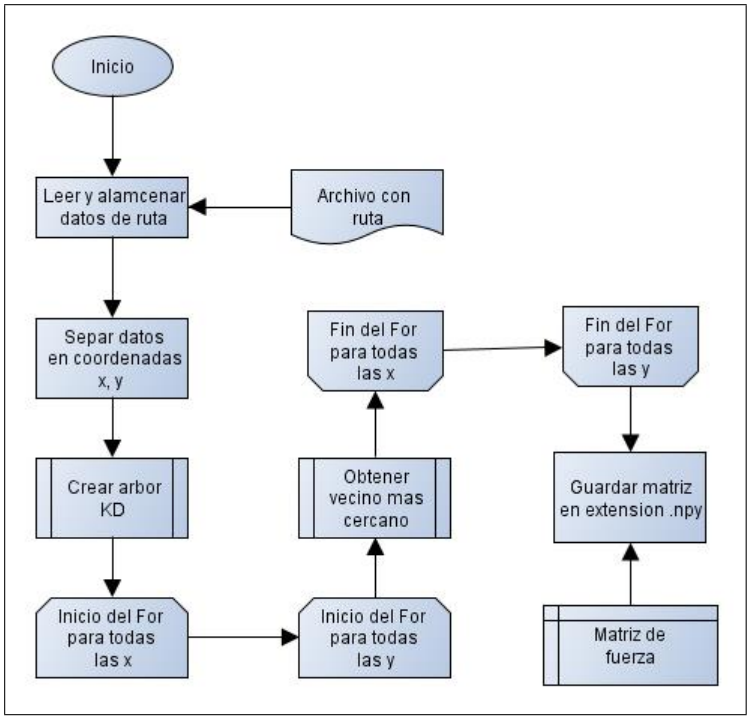

Figura 2.16: Algoritmo Generador de la Matriz de Fuerza.

Una vez obtenida la matriz de fuerza se debe implementar un código para que

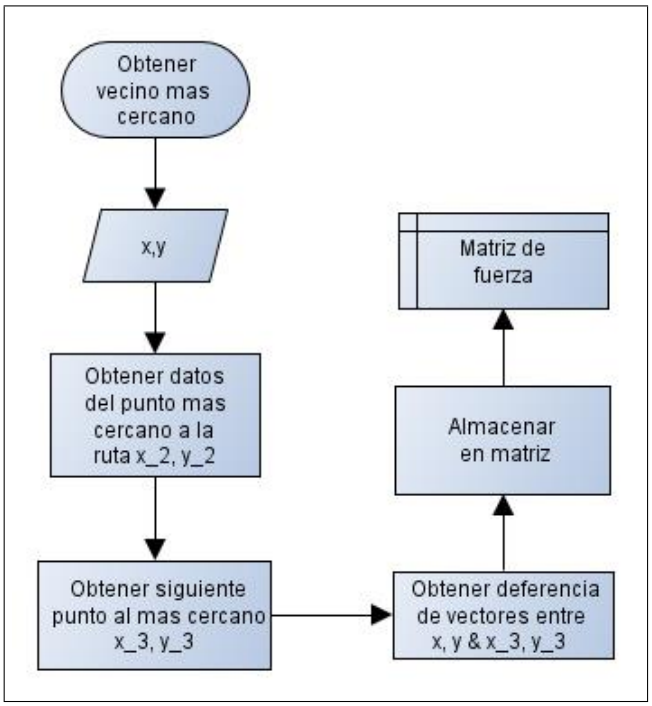

Figura 2.17: Diagrama de Flujo para la rutina “Obtener al vecino más cercano”.

el líder pueda seguir dicha trayectoria. La implementación de este seguimiento de trayectoria se realiza en el “Nodo MovimientoLíder”. En la Figura 2.18, se muestra el diagrama UML planteado para este nodo, asimismo en las Figuras 2.19 y 2.20 se muestran los diagramas de flujo que explican la funcionalidad de los métodos de la clase propuesta.

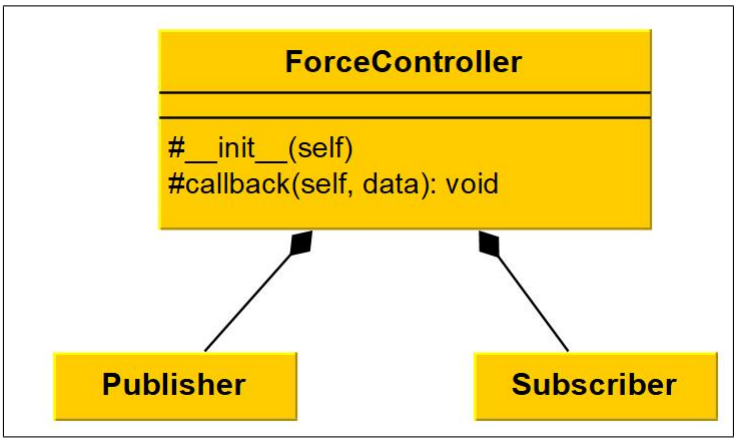

Figura 2.18: Diagrama UML del objeto ForceController a implementar en nodo Líder.

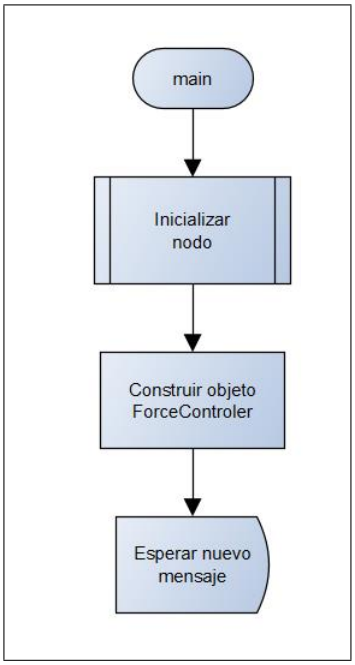

Figura 2.19: Diagrama de flujo del nodo Líder.

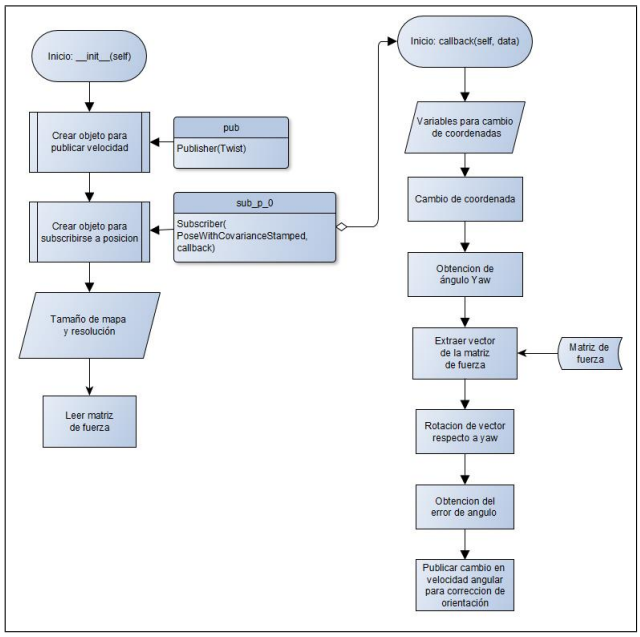

Figura 2.20: Diagrama de flujo de los métodos del objeto ForceController.

Este nodo es el encargado de suscribirse a su posición proveniente de la AMCL y con base a ella reorientarse para mantener la trayectoria junto con la ayuda de la matriz de fuerza.

Esquema Líder - Seguidor El implementar uno de los esquemas líder-seguidor propuestos en el marco de referencia implicaría mantener una posición relativa al líder d_1, bajo cualquiera de los métodos de control citados en el marco de referencia, lo cual podría causar colisiones con obstáculos. Para este caso en particular el seguidor más que mantener una posición relativa al líder, debe mantenerse sobre la trayectoria dictada por el líder.

Para evitar este problema y lograr que el seguidor se mantenga en la ruta, se pensó en mantener una velocidad lineal constante e igual para cada robot móvil, y enviar al seguidor los puntos de ruta por los que el líder pase, e irlos almacenando, sin ningún lazo cerrado encargado de mantener una posición relativa entre líder y seguidor. Con dichos puntos de ruta el seguidor, usando el mismo principio para

reorientarse que usa el líder, y de este modo alcanzará los puntos almacenados en el seguidor y por tener la misma velocidad se mantendrá una distancia constante sobre la trayectoria entre líder y seguidor, a la cual llamaremos d_2. La siguiente figura da un ejemplo más claro de la diferencia entre d_1 y d_2.

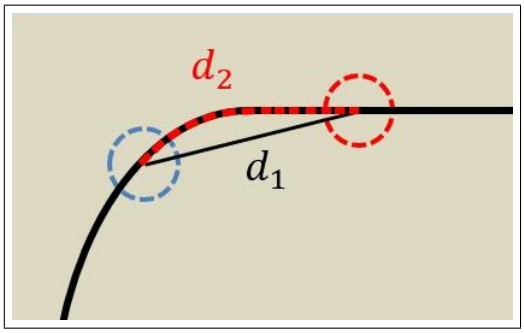

Figura 2.21: Representación de las distancias entre el líder y el seguidor.

En la Figura 2.21 se tiene al líder, de color rojo y al seguidor de color azul, la curva de color negro representa un tramo de la trayectoria a seguir, la cual está formada por $P_n(x_n, y_n)$ pares de coordenadas, siendo P_l la coordenada donde se localiza el líder y P_s donde se localiza el seguidor. Con esto definimos d_1 y d_2, de la siguiente manera:

$$d_2 = \sum_{P_n=x_n,y_n}^{P_l=x_{l-1},y_{l-1}} = \sqrt{(x_{n+1} - x_n)^2 + (y_{n+1} - y_n)^2} \tag{2.1}$$

$$d_1 = \sqrt{(x_l - x_n)^2 + (y_l - y_n)^2} \tag{2.2}$$

La distancia d_1 es únicamente la distancia entre líder y seguidor, mientras que d_2 es la distancia que separa al líder del seguidor, sobre la trayectoria.

Una vez definido el tipo de esquema líder-seguidor a usar sigue el diseño del algoritmo que cumpla lo propuesto, lo cual se llevará a cabo en el "Nodo seguidor". Hablando en términos más específicos de ROS dicho nodo se suscribe a las posiciones publicadas por el líder y las propias como se ve en la Figura 2.24 y ellas son almacenadas en una lista de python, que para este caso se llamará WPL (Figura 2.26),

después usando el mismo tipo de lazo de control usado para la orientación del líder, se reorienta el seguidor en función de su posición y la lista con los puntos de referencia almacenados (Figura 2.25). Los siguientes diagramas muestran el funcionamiento de dicho nodo.

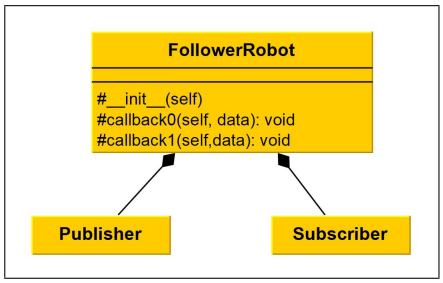

Figura 2.22: Diagrama UML del objeto FollowerRobot a implementar en nodo seguidor.

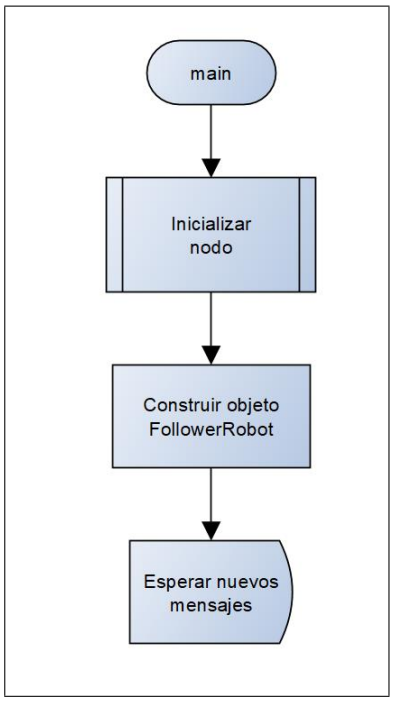

Figura 2.23: Diagrama de flujo del nodo seguidor.

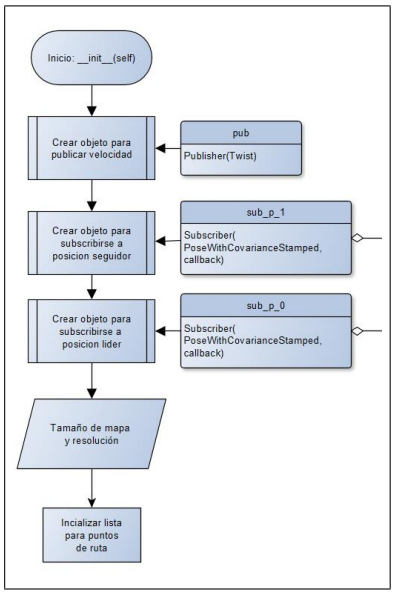

Figura 2.24: Diagrama de flujo de los métodos del objeto FollowerRobot parte I.

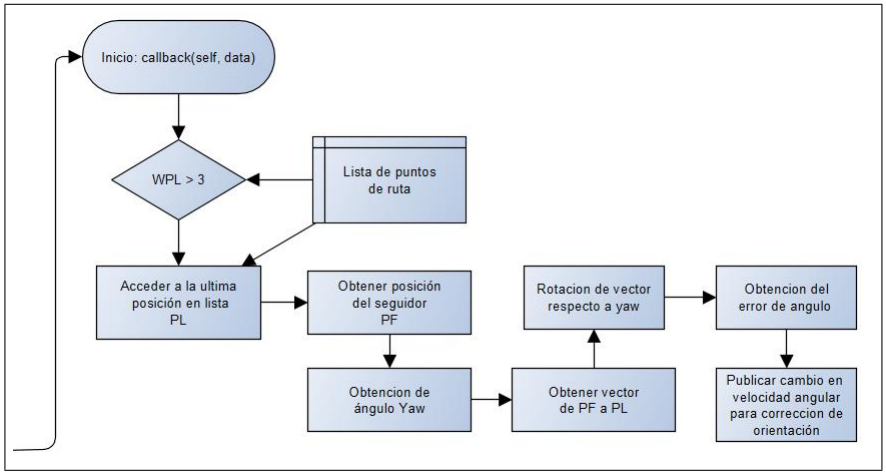

Figura 2.25: Diagrama de flujo de los métodos del objeto FollowerRobot parte II.

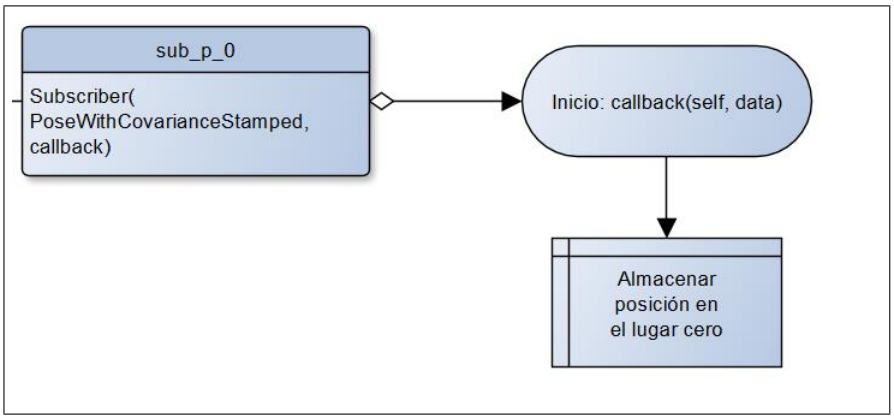

Figura 2.26: Diagrama de flujo de los métodos del objeto FollowerRobot parte III.

Evasión de obstáculos Para la evasión de obstáculos es necesario un método que se ajuste a la forma de operar de la matriz de fuerza por lo que como posibilidad se tiene el algoritmo por campos potenciales y el VFH. De [23] se tiene un análisis de los pros y contras de los distintos algoritmos para la evasión de obstáculos, basado en ese estudio se elige el VFH, ya era una forma que requería menor costo computacional y no afectaba de manera tan brusca el seguimiento de trayectoria y no se corre el riesgo de dejar al robot dentro de un mínimo local como sucede con los campos potenciales.

La manera de implementarlo fue la siguiente, este algoritmo se divide en dos nodos, el primero llamado NODO DETECTOR, genera el histograma (según los datos leídos por el sensor LIDAR) y se publican (Figura 35). El segundo se encarga de evaluar las posibles direcciones considerando también la dirección propia de la trayectoria obtenida de la matriz de fuerzas y en caso de no poder seguir por la dirección establecida por la matriz de fuerza se elige una cercana a ésta. Toda esta sección se une al NODO LÍDER (Figura 36).

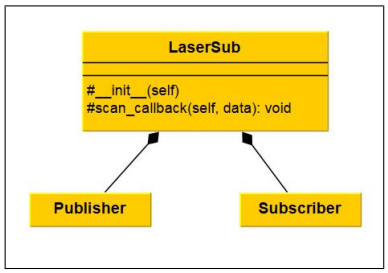

Figura 2.27: UML para nodo detector.

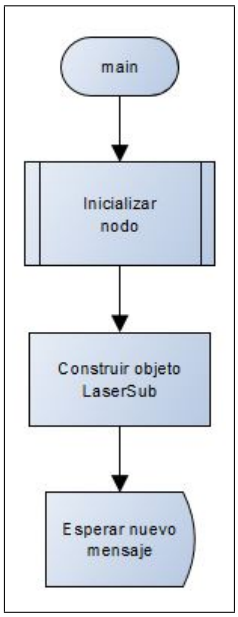

Figura 2.28: Diagrama de flujo nodo detector.

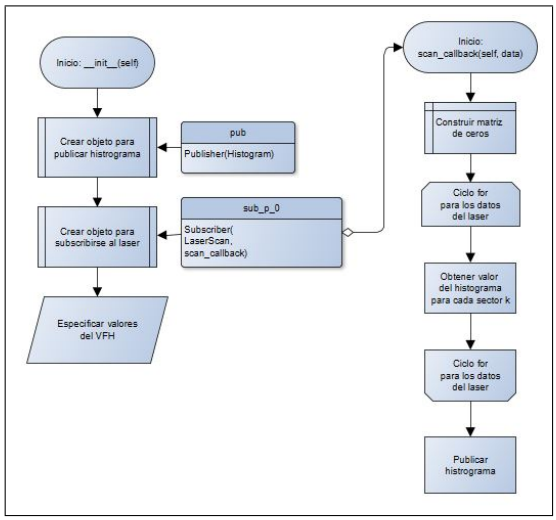

Figura 2.29: Diagrama de flujo método nodo detector.

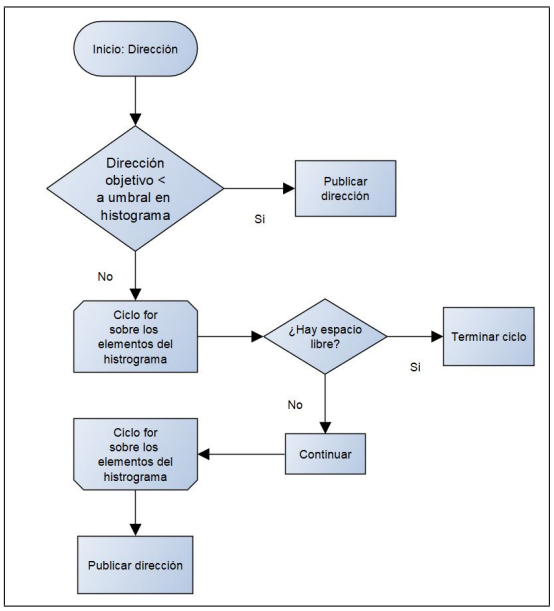

Figura 2.30: Diagrama de flujo para dirección.

Comunicación La comunicación se realizará por medio de WiFi entre los distintos dispositivos, es decir el turtlebot3 líder, el turtlebot3 seguidor y la computadora de monitoreo tendrán una comunicación de manera continua. Como se explicó en la

parte del diseño conceptual la arquitectura para el desarrollo de este proyecto será de tipo distribuida, ya que cada dispositivo antes mencionados, procesará distintos tipos de información y realizaran diversas tareas, que para un sistema centralizado teóricamente sería complicado y demandante computacionalmente mientras que para esta arquitectura el procesamiento de información se distribuye y reduce la demanda de poder computacional para un dispositivo en particular.

Los mensajes que serán transferidos vía WiFi son los tópicos publicados durante la ejecución del esquema líder seguidor. A continuación, se presenta una lista de ellos y la información que contienen.

- $/map$
- $/tb3_0/Histogram$
- $/tb3_0/amcl_pose$
- $/tb3_0/cmd_vel$
- $/tb3_0/odom$
- $/tb3_0/scan$
- $/tb3_1/amcl_pose$
- $/tb3_1/cmd_vel$
- $/tb3_1/odom$
- $/tb3_1/scan$

Para poder tener estos mensajes en comunicación con los distintos robots es necesario establecer una comunicación wifi entre los distintos robots y computadora de monitores. A continuación, se muestra como hacerlo.

Se obtiene la IP de la PC remota y se modifica en la computadora del Turtlebot3 como sigue:

- Ingresar el siguiente comando en cada Turtlebot3 y PC remota >> $\$nano\ /.bashrc$
- Modifique la dirección de localhost en ROS_MASTER_URI y $ROS_HOSTNAME$ con la dirección IP obtenida de la ventana de terminal anterior.

```
source /opt/ros/kinetic/setup.bash
source ~/catkin_ws/devel/setup.bash

export ROS_MASTER_URI=http://192.168.0.100:11311
export ROS_HOSTNAME=192.168.0.100
```

Figura 2.31: Configuración del IP's para la comunicación vía WiFi.[21]

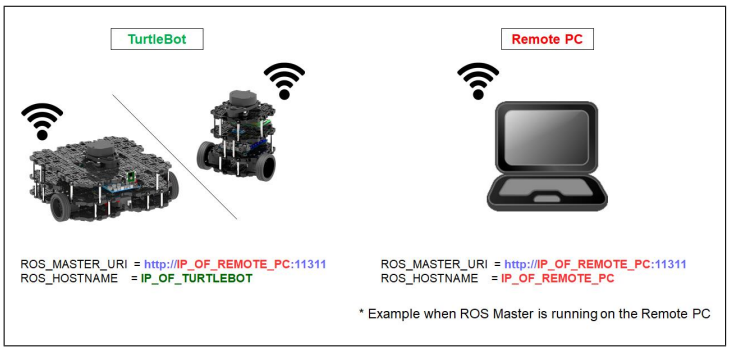

Figura 2.32: Ejemplo de configuración WIFI. [29]

Control de motores Para lograr que los robots móviles se comporten como se desea es necesario hacer uso de la cinemática inversa de su configuración. Estas ecuaciones fueron planteadas en el marco teórico. El enfoque es hacia el cálculo de la velocidad necesaria en cada una de las llantas para obtener así el cambio en la dirección y siempre mantener una velocidad lineal constante en todo momento. Esto se explica a continuación.

Lo primero es considerar la velocidad lineal a la que los turtlebot3 deseamos que vayan V_k y tomando como tope la velocidad máxima V_max, sabemos que estos van a $0.22\frac{m}{s}$, tomando la ecuación de la cinemática inversa tenemos que para la velocidad del motor 1 y 2 de nuestros turtlebot3 obtenemos 2.3 y 2.4 .

$$\phi_r^{'} = \frac{V_k + \theta^{'} b}{r} \tag{2.3}$$

$$\phi_l^{'} = \frac{V_k - \theta^{'} b}{r} \tag{2.4}$$

Debido a la restricción de velocidad máxima de los motores ($\phi_{max}^{'}$) el máximo de velocidad angular que podemos obtener es 2.5.

$$\theta^{'} = \frac{\phi_{max}^{'} r - V_k}{b} \tag{2.5}$$

Estas ecuaciones (2.3,2.4,2.5) son las implementadas por la librería que usa ROS para controlar el turtlebot3.

2.2.3. Área Funcional 3: Percepción (Comunicación entorno - robot)

Como se puede observar en la Figura 2.9 en la página 33, la tercer área funcional a analizar es la de Percepción, la cual tiene la función de adquirir los datos propioceptivos y extereoceptivos de cada uno de los robots, es decir, la recabación de datos como la distancia, velocidad y posición de los móviles para poderlos ubicar en el espacio ya sea por odometría (información propioceptiva), o por la recabación de la información del ambiente, para poder ubicar los obstáculos dentro del área de trabajo (información exteroceptiva), así mismo cabe indicar que estas dos formas trabajan en conjunto siendo una más efectiva para la detección del móvil en la zona de trabajo y la otra para dar ajustes más precisos como cual ha sido el desplazamiento total, de esta manera se evita que por el ruido del sensor lidar el móvil de repente aparezca en otro lugar muy lejano (para el programa) como suele pasar si no se usa odometría.

Para la recabación de la información propioceptiva se necesita un control preciso de los actuadores del móvil para lograr orientar a los móviles de manera correcta, y así, en conjunto con un buena lectura de los datos obtenidos de la rotación de

las ruedas se lleve a cabo una estimación más precisa de la posición de los móviles durante la navegación. Los datos recabados por los sensores posteriormente serán leídos por el hardware implementado en el área funcional de procesamiento, siendo así el hardware él que se encargará de su procesamiento para realizar una tarea en específico. Para poder llevar a cabo la solución de esta área funcional se plantea separar de manera general en las dos clasificaciones de información obtenida por los sensores y así desglosar cada uno de los elementos que conllevan.

2.2.3.1. Datos propioceptivos

Este tipo de datos son los obtenidos a partir de variables internas del móvil, por lo cual su medición es únicamente referente a la posición relativa del móvil y a las variables físicas donde este se encuentre, este concepto tiene su origen en la propiocepción, el cual es el sentido que informa al organismo de la posición de los músculos, en otras palabras, es la capacidad de sentir la posición relativa de partes corporales contiguas. Análogamente a este concepto en el estudio de la información de sensores se ocupa este término para diferenciar entre los sensores que dependen del entorno para poder realizar su medición de una manera relativa, y los que no.

De esta manera se tiene como principal ventaja que los sensores propioceptivos no dependen de estímulos externos, por su característica de medir únicamente los datos o variables referentes a los estados internos del móvil. Sin embargo, por esta misma característica es difícil obtener datos en concreto confiables, siendo por lo tanto una desventaja el nivel de error total almacenado o acumulado, el cual se puede estimar con un muestreo de los sensores, pero nunca se llegará a tener el valor verdadero sino solo con un margen de incertidumbre, así mismo este error podría ser reducido en gran manera si se captara la información externa como en el caso de los sensores exteroceptivos.

2.2.3.2. Sensores integrados

2.2.3.3. Encoders rotatorios

En robótica, los encoders se utilizan habitualmente para llevar a cabo tareas odométricas, las cuales permiten estimar la posición de un robot móvil con respecto a una posición inicial conocida, basándose en el número de revoluciones de cada rueda del robot, y en la dirección en que se mueven las ruedas. Los encoders más empleados son los rotatorios (o de eje), que son dispositivos electromecánicos que convierten la posición angular de un eje en un código digital. Los encoders pueden ser magnéticos, mecánicos y de otros tipos, si bien los más comunes son los ópticos.

2.2.3.4. Encoder óptico rotatorio, de tipo relativo (o incremental)

Este tipo de encoder está formado por un disco con agujeros (o ranuras) cerca de su borde y se coloca de modo que el eje del disco coincida con el eje del motor y el de la rueda (que normalmente son el mismo). El número de agujeros en el disco determina la precisión del encoder. A un lado del disco se coloca un led infrarrojo, mientras que, al otro lado, y enfrente del led infrarrojo, se coloca un fototransistor receptor de infrarrojos. Cuando el disco gira, los agujeros hacen que el fototransistor reciba luz de manera intermitente, generándose así una secuencia de pulsos. Se acostumbra a tener un par de leds que midan ranuras desfasadas 90 grados, cada una con una configuración como se describió anteriormente, esto con el fin de conocer el sentido de giro del motor y así poder calcular mediante un sistema relativo la posición del motor sin conocer la posición inicial. A pesar de que se puede programar mediante una técnica de interrupciones o leyendo constantemente la lectura del giro, una forma económica y fácil de obtener es mediante un flip-flop D con una señal conectada a la entradade reloj y el otro a la entrada de datos, al ser 2 señales cuadradas desfasadas 90 grados cuando detecta un cambio de estado en el pin de reloj, actualiza con el dato del segundo, el cual por su construcción coincide siempre con el giro (1-Sentido horario, 0-Sentido anti horario).

2.2.3.5. Encoder óptico absoluto

Son más apropiados para casos en los que se producen rotaciones más lentas, o poco frecuentes, como por ejemplo averiguar la rotación de un volante en un vehículo automatizado. El disco está formado por patrones codificados de zonas opacas y zonas transparentes. El funcionamiento es similar a los encoders relativos, pero en lugar de utilizarse un único led y un único foto receptor, se utilizan varios como se observa en la Figura 2.33. Por medio de estos sensores se tiene la posición absoluta sin necesidad de ningún cálculo como ocurre en el relativo, además de conocer siempre su posición inicial, pero su construcción es más compleja y al ser más señales su tiempo de respuesta no es tan alto como en el relativo.

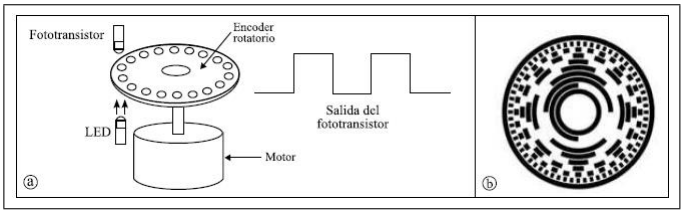

Figura 2.33: Funcionamiento del encoder absoluto.

2.2.3.6. Acelerómetros

Los acelerómetros son dispositivos que detectan cambios en la velocidad. La mayoría no están preparados para medir velocidad constante, sino que solamente miden aceleración o desaceleración. En un principio estaban restringidos al ámbito científico e industrial, debido a su alto coste. Sin embargo, gracias a su abaratamiento, cada vez se encuentran más presentes en aparatos de uso cotidiano, como ordenadores portátiles (que se suspenden en caso de caídas), mandos de consolas de videojuegos, o incluso teléfonos móviles.

Dentro de la robótica móvil, uno de los principales usos de un acelerómetro es la detección de movimiento. Los acelerómetros presentan la ventaja de que pueden detectar desplazamientos del robot incluso cuando las ruedas del robot están detenidas.

Otros posibles usos del acelerómetro son la detección de colisiones o la teleoperación robótica.

Algunas desventajas de los acelerómetros es que detectan con dificultad las aceleraciones de pequeña magnitud (como por ejemplo al realizar giros muy lentos) y que son muy sensibles a irregularidades en el suelo.

2.2.3.7. Datos exteroceptivos

Este tipo de datos son los que nos dan información del medio exterior, así como los propioceptivos tienen su origen en el cuerpo humano, también este concepto tiene su origen a partir del sistema exteroceptivo, el cual es el encargado de recibir estímulos externos al cuerpo como el frío, el calor, la presión, el dolor, etc., realizándolos por medio de los 5 sentidos entre otros. Las medidas de este tipo de sensores normalmente son interpretadas por el móvil para extraer características del entorno y a partir de estas elaborar un croquis del entorno, conocer los elementos que lo conforman, ver si es un entorno apropiado, etc.

2.2.3.8. Sensores implementados

2.2.3.9. Escáner láser de medición de distancias

La palabra LASER responde a las siglas Light Amplification by Stimulated Emission of Radiation. Entre el sonar (competencia del Lidar) y el LASER existe una diferencia fundamental: la velocidad de propagación. Para el sonido es 0.3 m/ms, mientras que para las señales electromagnéticas es 0.3 m/ns, es decir, 1 millón de veces más rápido. Por ejemplo, en una distancia de 3 m, un sonar tardaría 10 ms, mientras que un láser mediría la distancia en 10 ns. Es evidente que para medir el tiempo de vuelo de señales electromagnéticas se necesita una tecnología más avanzada que para medir el tiempo de vuelo de un sonar. Esto explica que el precio de un láser sea mucho más elevado que el de un sonar.

Como ya se mencionó el sensor LIDAR basa su funcionamiento en la medida del

tiempo de vuelo como se observa en la Figura 2.34. Para determinar la distancia a la que se encuentra un objeto, el sensor emite un pulso de luz infrarroja. Cuando el pulso incide sobre el objeto más cercano, regresa hacia el sensor y se determina el tiempo transcurrido. Conocido el tiempo de ida y vuelta del pulso (tiempo de vuelo), se calcula fácilmente la distancia al objeto detectado. De este modo se puede medir la distancia en una sola dirección, pero gracias a que dispone internamente de un espejo rotatorio, se logra un efecto de barrido en dos dimensiones. La amplitud del barrido en este modelo es siempre de 360°. También se puede observar que, si los pulsos emitidos no indicen sobre ningún objeto cercano, el láser devuelve la distancia máxima, dando lugar a un conjunto de medidas que forman un semicírculo. Al conjunto de medidas que obtiene el láser se le denomina barrido láser.

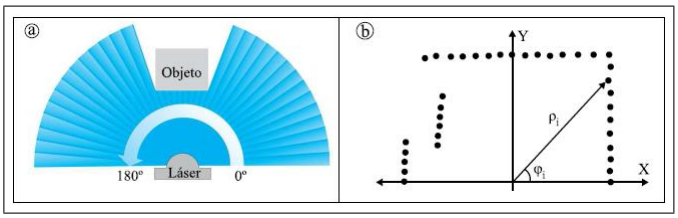

Figura 2.34: Funcionamiento del sensor LIDAR.

2.2.3.10. Sensor LIDAR LDS-01

Es un sensor láser 2D capaz de escanear en 360°, las diferentes distancias, las cuales son usadas en este proyecto para llevar a cabo el algoritmo SLAM para su localización y VFH para esquivar objetos.

Elemento	Características
Voltaje de Operación	5V ± 5 %
Fuente de Luz	Diodo semiconductor de láser (λ=785nm)
Seguridad del láser	IEC60825-1 Clase 1
Consumo de corriente	400 mA o menor (Corriente pico 1A)
Detección de distancia	120mm − 3,500mm
Interfaz	▪ 3.3V USART(230,400 bps) ▪ 42bytes por 6 grados, Opción full dúplex.
Resistencia a la luz	10,000 lux o menos
Frecuencia de Muestreo	1.8kHz
Dimensiones	69.5(Ancho) X 95.5(Largo) X 39.5(Alto)mm
Masa	Debajo de 125g

Cuadro 2.12: Características del sensor LIDAR.

2.2.3.11. Dimensiones

En la Figuras 2.35 y 2.36 se pueden observar las distintas dimensiones del sensor LIDAR, las cuales son importantes de considerar ya que dependiendo del tamaño del LIDAR y de otros elementos se puede llegar a la elección de la medida ideal para cada móvil.

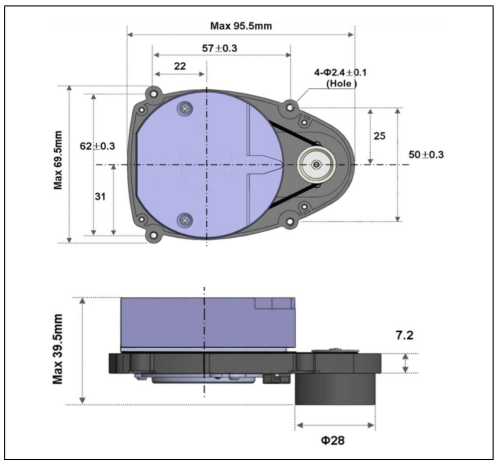

Figura 2.35: Dimensiones del sensor LIDAR. Vista superior y lateral.

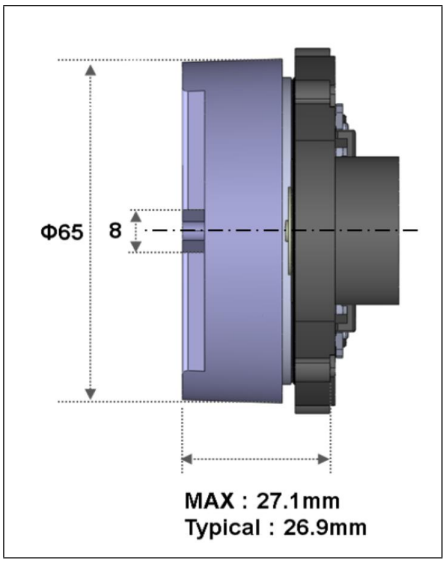

Figura 2.36: Vista Frontal del sensor LIDAR.

2.2.3.12. Montaje del sensor

El sensor LIDAR se coloca en la parte superior con 4 soportes removibles para facilitar su reemplazo en cualquier caso que se necesitará, además que favorece a no tener que retirar toda la capa. En cuanto a su función de obtener las distancias para cada grado de su alrededor es más factible colocarla en la posición superior ya que

así ningún soporte le impide realizar la medición correcta.

En su colocación los 4 soportes ofrecen un nivel de seguridad mayor, ya que para poder definir un plano en sus 3 dimensiones se necesitan 3 puntos que pasen por él, dando de esta forma un sistema completamente definido, sin embargo, al tener 4 puntos de sujeción se sobre define este haciendo forzosamente que el 4 punto solo se pueda colocar si pertenece al plano, en este caso al pertenecer si ocurriera que algún punto se quitará el sistema seguiría completamente definido y por lo tanto estable ante cualquier eventualidad referente a esto. En la sección de Implementación se habla más a detalle del proceso de montaje.

2.2.3.13. Acondicionamiento

Para conectar el sensor LIDAR a la tarjeta Raspberry Pi 3 se usa un convertidor de protocolo USB al protocolo UART como se observa en la Figura 2.37, esto debido a que el LIDAR recibe comandos de 40 bytes con los cuales decide cada cuantos grados va a realizar un escaneo, asimismo como la intensidad a la que se usa el láser, todo esto para obtener del LIDAR la distancia medida de la mejor manera posible. Al finalizar de recibir los 40 bytes se comprueba mediante una suma que los datos fueron correctos y se prosigue en caso contrario vuelven a enviar los datos.

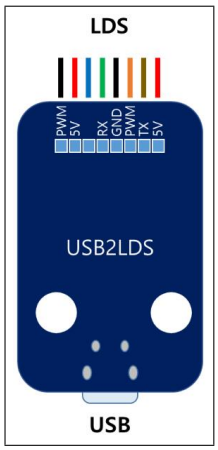

Figura 2.37: Pines del sensor LIDAR.

La descripción de los pines se muestra a continuación en la Tablas 2.13, 2.14 y 2.15 en las cuales se considera tanto el motor que se encarga del giro del LIDAR, así como el circuito de control del láser.

Número del Pin	Nombre	Descripción
6	Vcc (+5.0V)	Fuente de alimentación (positiva).
5	Tx	Salida serial del LIDAR.
4	PWM	Salida del LIDAR para controlar al motor.
3	GND	Tierra.
2	RX	Entrada serial del LIDAR
1	BOOT0	Pin para entrar en modo booteo.

Cuadro 2.13: Descripción de los pines del sensor LIDAR.

Número del Pin	Nombre	Descripción
2	Vcc (+5.0V)	Fuente de alimentación (positiva).
1	PWM	Entrada del motor para controlar posición.

Cuadro 2.14: Descripción de los pines del motor del sensor LIDAR.

Clave	Descripción
b	Comenzar la operación.
e	Pausar la operación.

Cuadro 2.15: Comandos para operación del LIDAR.

2.2.4. Área Funcional 4: Alimentación (Suministro de energía)

La cuarta área funcional es la de Alimentación, la cual es la encargada de abastecer de energía a tres áreas funcionales fundamentales para el móvil por medio de una batería como se muestra en la Figura 2.38. Cada una de estas áreas tiene sus propios elementos para poder llevar acabo sus tareas; en el caso de nuestra segunda área fun-

cional denominada “Procesamiento”, es necesario abastecer a dos tarjetas, las cuales son la Raspberry Pi 3 y la ARM Cortex M7, cada una con sus respectivas entradas y salidas que se abordan a detalle en la sección de Procesamiento. Hablando de la tercera área funcional denominada “Percepción”, es necesario alimentar de acuerdo a las especificaciones técnicas el sensor LIDAR, el cual nos dará toda la información del entorno y nos brindará los datos exteroceptivos del ambiente que posteriormente serán evaluados y procesados en la segunda área funcional. En cuanto al movimiento, la quinta y última área funcional, los encargados de llevar acabo está tarea son dos actuadores Dynamixel (uno en cada llanta) que son los responsables de la locomoción del móvil y de la ejecución de toda la información antes proporcionada por las distintas áreas funcionales. Teniendo en cuenta que estos actuadores (por el encoder interno) al igual que el sensor LIDAR, brindarán información acerca del movimiento (rotación) de las llantas y posteriormente esta información nos servirá para conocer la posición y orientación de los móviles (odometría). Teniendo así que todos estos elementos y aspectos mencionados tendrán que ser alimentados por una fuente.

La encargada de abastecer energía a todos los elementos que conforman cada área funcional es una batería LiPo (batería de Polímero de Litio) ubicada en la primera capa dentro del Turtlebot3 como se muestra en la Figura 2.38.

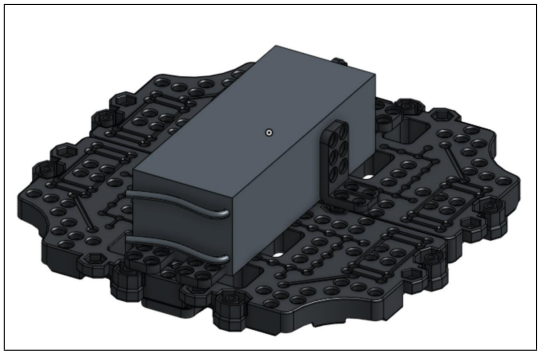

Figura 2.38: Ubicación de la batería de alimentación. Primera capa del Turtlebot3.

La batería tiene una capacidad nominal de 1800 mAh con una salida de 11.1V y cuenta con solo una salida, que es la que abastece a todo el robot y una entrada

como se muestra en la Figura 2.39, que es la que sirve para que se pueda cargar, adicionalmente para cargarla se usa un cargador de baterías LiPo en conjunto con una fuente de voltaje.

Figura 2.39: Batería LiPo, con su entrada y salida [31].

2.2.4.1. Características Técnicas de la batería LiPo.

Nombre	Característica
Nombre de la marca	GTK
Número de modelo	473474P
Capacidad nominal	1800 mAh
Capacidad	1500 mAh
Batería tipo	Polímero de Litio
Peso [g]	106
Tamaño [mm]	26 x 35 x 88
Componentes	3 celdas
Electricidad [Wh]	19.98
Voltaje [V]	11.1
Velocidad continua de descarga	10C
Seguridad	PCM (Por sus siglas en inglés Protection Circuit Module)

Cuadro 2.16: Especificaciones técnicas de la batería. [11]

2.2.4.2. Ensamble de la batería LiPo con la estructura

En la Figura 2.40 se muestra el proceso de ensamble de la batería en el Turtlebot3 burger, un proceso que se debe tener en mente, ya que el manual del Turtlebot3 burger aconseja retirar la batería de la estructura cada vez que se quiera recargar.

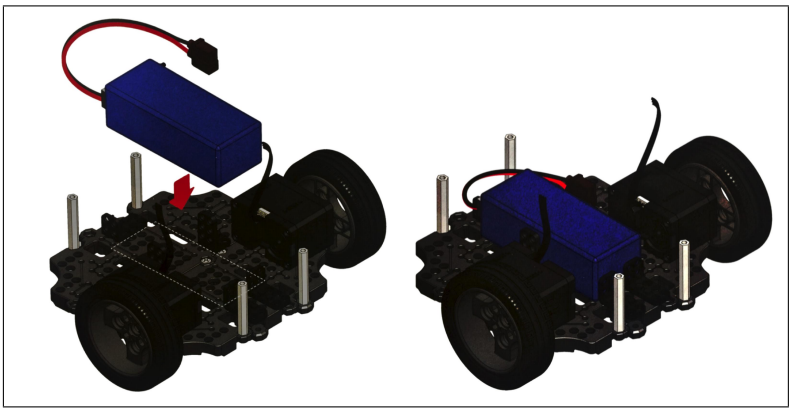

Figura 2.40: Ensamble de la batería LiPo en la primera capa del Turtlebot3.

2.2.5. Área Funcional 5: Movimiento (Desplazamiento del robot)

El movimiento o locomoción forma una parte fundamental del proyecto, ya que es la última función antes de que nuestro robot ejecute la información proporcionada, una vez esta haya pasado por las 4 funciones previas que engloban el proyecto. En la Figura 2.41 se muestra el proceso de ensamble para cada llanta.

Figura 2.41: Ensamble del rin y la llanta de los robots.

El sistema de locomoción está en la primera capa dentro de la estructura de nuestros robots, y tiene como eje fundamental dos actuadores Dynamixel XL 430 - W250 en cada llanta, lo que se traduce a una configuración cinemática de tracción diferencial como se muestra en las Figuras 2.42 y 2.43.

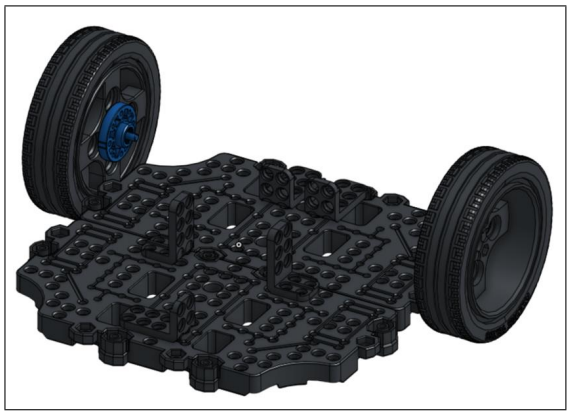

Figura 2.42: Estructura de la locomoción de los robots.

Figura 2.43: Ensamble de los actuadores con la estructura.

2.2.5.1. Actuadores Dynamixel

Para el control de la posición y velocidad del móvil se utilizan los motores Dynamixel XL430-W250-T los cuales cuentan con un controlador interno, el cual puede ajustarse en 6 distintos modos, para este caso principalmente se usará el control de velocidad, esto se logrará transmitiendo la información por medio del protocolo UART, por lo cual cada motor se conecta al Open CR que se encarga de mandarle la información de una manera que el motor pueda interpretar directamente [26].

2.2.5.2. Análisis interno del motor

El motor este ensamblado como se muestra en la Figura 2.44, de tal manera que eléctricamente lo compone un motor de CD, una tarjeta controladora y un encoder. La tarjeta controladora se encarga de recibir y ejecutar las instrucciones, así como de leer los valores del sensor de corriente y el encoder es el sensor para determinar su posición.

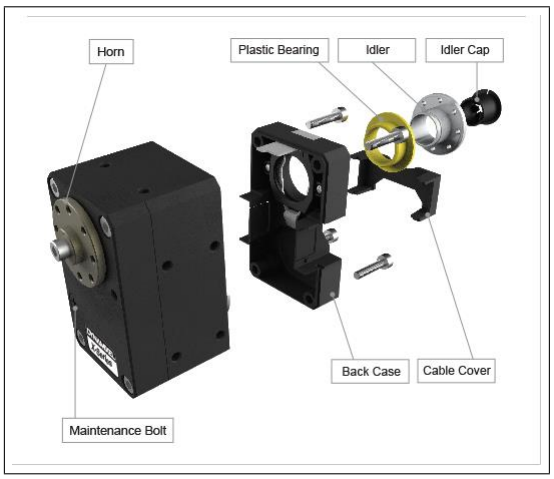

Figura 2.44: Explosionado del motor del Dynamixel.

Los tipos de control que puede realizar son los siguientes:

- Control de par.
- Control de velocidad.
- Control de posición.
- Control extendido de posición.
- Control de corriente basado en el control de posición.
- Control PWM.

Figura 2.45: Características del actuador.

Los actuadores Dynamixel son de los actuadores más avanzados para robótica económica de alto rendimiento, también llamados actuadores inteligentes, ya que ofrecen la posibilidad de programar entre 50 y 57 comandos permitiendo definir el comportamiento del actuador de una mejor manera otorgándoles funciones especiales como las que se muestran en la Figura 2.45, en comparación con el servomotor, más comúnmente usado, que solo interpreta la orden de "ángulo objetivo"proporcionado por una señal PWM.

En las Figuras 2.46 y 2.47, se muestran los motores ya ensamblados para formar la configuración diferencial.

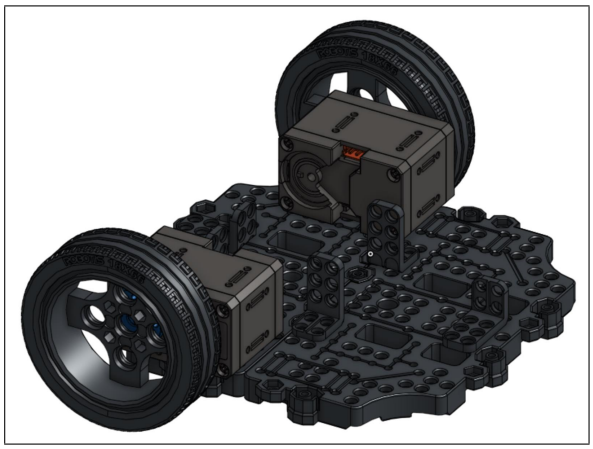

Figura 2.46: Sistema de locomoción en conjunto con los actuadores Dynamixel XL 430 - W250.

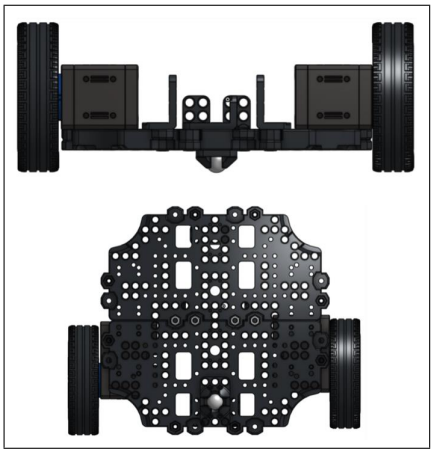

Figura 2.47: Sistema de locomoción en conjunto con los actuadores Dynamixel XL 430 - W250. Vista frontal e inferior.

2.2.5.3. Características del motor

Los actuadores Dynamixel son capaces de proporcionar valiosa información de retroalimentación permitiendo leer y procesar información en tiempo real captada por sus sensores embebidos, los cuales son sumamente útiles para hacer odometría con los móviles. Entre la información que pueden proporcionar los actuadores Dy-

namixel se encuentra leer la posición actual del motor, la velocidad, la temperatura interna, el par y la tensión de alimentación.

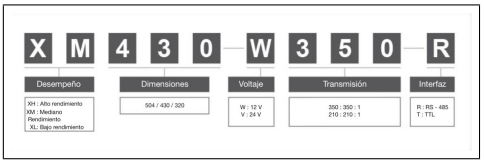

Figura 2.48: Nomenclatura de los actuadores Dynamixel [26].

En nuestro caso trabajaremos con los actuadores Dynamixel XL 430 - W250 que de acuerdo a la Figura 2.48, son los actuadores del más bajo desempeño dentro de las tres posibles clasificaciones de los actuadores, y cuyas especificaciones y características técnicas se mencionan a continuación en la Tabla 2.17 en conjunto con sus curvas características que se muestran en la Figura 2.49.

Elemento	Características
Nombre del modelo	XL 430 W250
Peso [g]	57.2
Dimensiones [mm]	28.3 x 46.5 x 34
Transmisión	258.5: 1
Voltaje de operación [V]	▪ 9.0 ▪ 11.1 ▪ 12.0
Torque	▪ 1.0 [N.m] (a 9.0 [V], 1.0 [A]) ▪ 1.4 [Nm] (a 11.1 [V], 1.3 [A]) ▪ 1.5 [Nm] (a 12.0 [V], 1.4 [A])
Velocidad de paso (sin carga)	▪ 47 [rev/min] (a 9.0 [V]) ▪ 57 [rev/min] (a 11.1 [V]) ▪ 61 [rev/min] (a 12.0 [V])

Cuadro 2.17: Características técnicas del actuador Dynamixel XL 430 W250 [25].

Elemento	Características
Algoritmo de control	▪ PID
Grados de precisión	▪ 0.088º
MCU	▪ ST CORTEX M3 32 Bits
Sensor de posición	▪ Enconder sin contacto (12Bit, 360) por AMS
Resolución	▪ 0.088 x 4.096 pasos
Rango de operación	▪ Modo control de velocidad: Encendido sin fin. ▪ Modo control de posición: 0º a 360º. ▪ Modo control extendido: 256 revoluciones. ▪ Modo control PWM: Encendido sin fin.
Voltaje de salida [V]	▪ 6.5 a 12.0V (Voltaje recomendado: 11.1 V)

Cuadro 2.18: Características técnicas del actuador Dynamixel XL 430 W250 [25].

Elemento	Características
Temperatura de operación	5ºC a 72ºC
Señales de control	▪ Paquete digital
Tipo de protocolo	▪ Comunicación serie asíncrona semidúplex (8 bits, sin paridad)
Transmisión de datos	▪ 9600 bps a 4.5 Mbps
Retroalimentación	▪ Posición ▪ Velocidad ▪ Carga ▪ Trayectoria ▪ Temperatura ▪ Voltaje de entrada
Material	▪ Carcasa: Plástico. ▪ Engranes: Plástico.

Cuadro 2.19: Características técnicas del actuador Dynamixel XL 430 W250 [25].

Data

	Unit	data
Dimension	mm (in)	28.5 x 46.5 x 34 (1.12 x 1.83 x 1.34)
Weight	g (oz)	57.2(2.02)
Operating Voltage	V	11.1
Gear Ratio	-	258.5 : 1
Stall Torque	N.m (oz.in)	1.4(198)
Stall Current	A	1.3
No Load Speed	RPM	57
Position Sensor	-	Contactless Absolute Encoder (AMS)
Motor	-	Cored Motor
Operating Range	°	360
Resolution	steps/turn	4096
Network Interface	-	TTL
Material	Casing	Enpla
	Gear	Enpla

Performance Graph

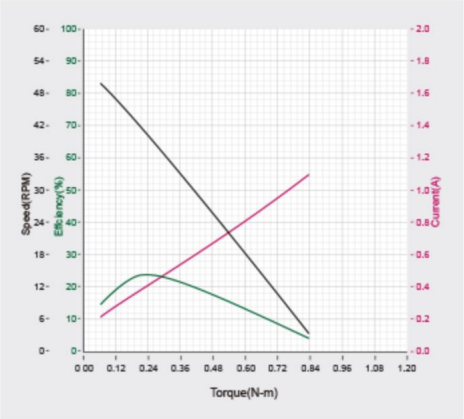

Figura 2.49: Gráfica de funcionamiento del motor DYNAMYXEL [26].

Entre cualidades del motor DYNAMYXEL XL 430 W250 actuador tenemos:

1. Torque mejorado y diseño compacto.
2. Durabilidad mejorada y capacidad de expansión.
3. Al tener la caja trasera hueca minimiza la tensión del cable.
4. Marcos directamente enroscados en la caja.
5. 6 modos de funcionamiento.
6. Control perfil para la planificación de movimientos.
7. Eficiencia energética con tiempo mejorado de operación.

Y en cuanto al ambiente de programación, los actuadores DYNAMYXEL pueden ser programados mediante:

- OpenCM ID.
- C/C++, Labview, Matlab, Visual Basic.

- Software exclusivo [Dynamixel Workbench].

Basándonos en este último, el Dynamixel Workbench es un metapaquete que contiene cuatro paquetes fundamentales, los cuales son; Administrador único (single manager), Controlador, Operador y Caja de herramientas (Toolbox). El paquete de Administrador único (single manager) provee de paquetes que pueden programar todas las series del Dynamixel, incluidas la serie X, y la PRO utilizando la biblioteca del Toolbox desarrollada en la base de Dynamixel SDK. Estos paquetes no sólo muestran el estado del Dynamixel, sino que también, permiten cambiar los valores de las direcciones de la tabla de control por comandos de línea o mediante la interfaz gráfica. El paquete de controladores nos presenta como utilizar los actuadores Dynamixel en diferentes modos de operación con la librería de Dynamixel Workbench Toolbox. La interfaz gráfica de Dynamixel Workbench se muestra en la Figura 2.50.

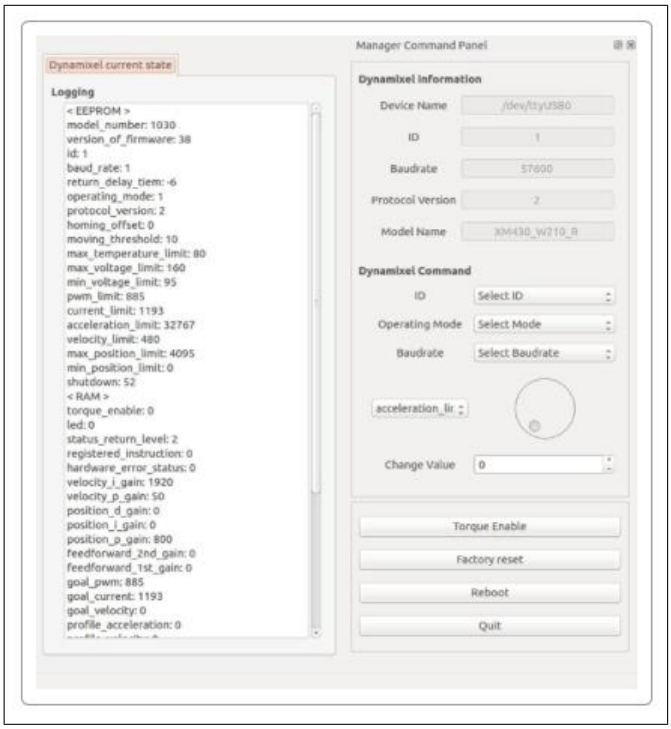

Figura 2.50: Interfaz de programación de los actuadores Dynamixel.

2.2.6. Integración del Sistema

Una vez conociendo todos los elementos que conforman a los robots móviles, se dará paso a la integración de todos los componentes dentro de la primera área funcional que es la de Estructura. La primera plataforma o capa es destinada al montaje de los dos actuadores Dynamixel, en conjunto con la batería Li-Po que suministra energía a todo el robot como se muestra en la Figura 2.51.

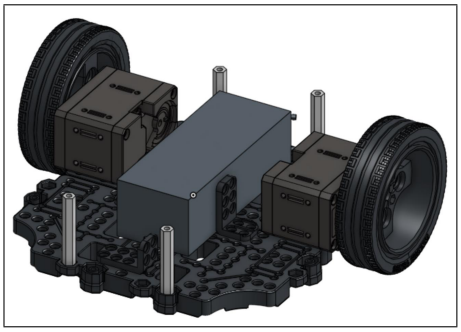

Figura 2.51: Capa 1. Implementación de la capa con los actuadores y la batería LiPo.

La segunda capa como se muestra en la Figura 2.52 es la destinada para el montaje la tarjeta OpenCR que es la encargada de mandar las señales de control a los actuadores y leer señales externas que ayuden al funcionamiento del robot como pueden ser sensores o interfaces por medio de botones mecánicos.

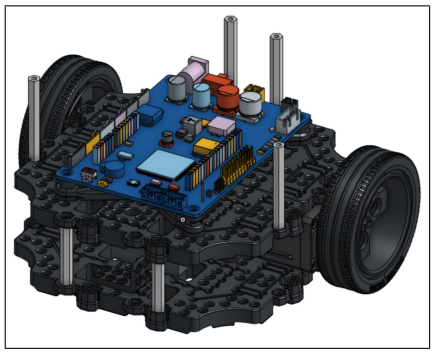

Figura 2.52: Capa 2. Implementación de la capa 1, capa 2 y tarjeta Open CR.

En la tercera capa como se muestra en la Figura 2.53 se coloca la Raspberry

Pi 3 encargada de ejecutar el sistema operativo ROS y el conector usb para leer el sensor lidar. Y por último en la última capa se coloca el sensor lidar sin ningún elemento extra para que este pueda escanear su alrededor sin ninguna interferencia otorgándole de esta manera un correcto funcionamiento al sensor.

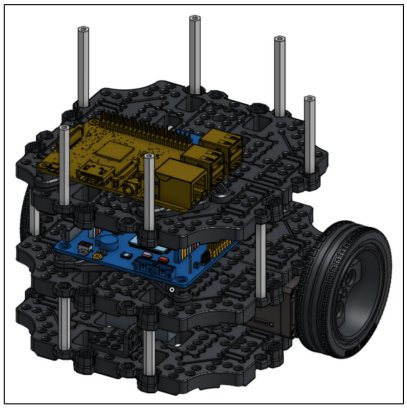

Figura 2.53: Capa 3. Implementación de la capa 1, capa 2, capa 3 y tarjeta Raspberry Pi 3.

Ya explicadas las distintas capas que conforman todo el sistema del robot, se muestra ensamble del robot en las Figuras 2.54 y 2.55 utilizando un total de 193 piezas en la parte estructural y 12 piezas que involucran la parte electrónica.

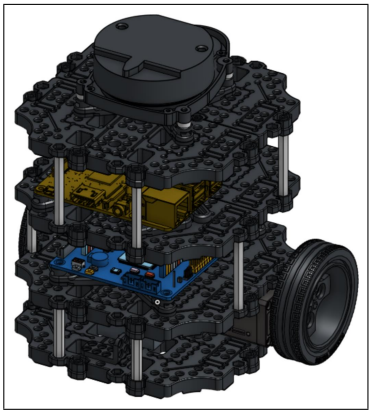

Figura 2.54: Vista isométrica del robot con todas las capas ensambladas.

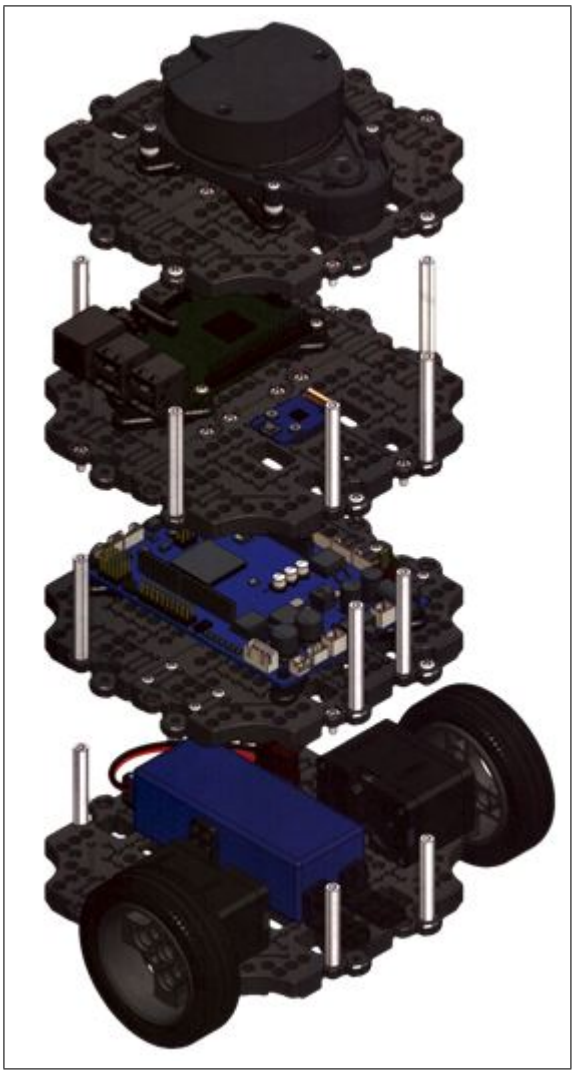

Figura 2.55: Explosionado y distribución de los componentes que conforman al robot.

2.2.6.1. Elementos de Software

Respecto a la parte de procesamiento, las áreas funcionales se desarrollan en diferentes nodos de ROS, es decir un programa no cumplirá una función completa, sino que dicha función estará dividida en distintos nodos, en cada una de las áreas funcionales se mencionaron los nombres de los nodos en los cuales se iban a desarrollar las áreas funcionales, la manera en que los diversos nodos se comunican es por

medio de tópicos, los cuales también fueron listados en la sección de transmisión de mensajes. Una manera simple de mostrar la integración de estos nodos es a través de los siguientes diagramas, los cuales presentan los nodos activos para cada Turtlebot 3 burger. Se proponen, para el líder y seguidor la interconexión de los siguientes nodos mostrados en las Figuras 2.56 y 2.57.

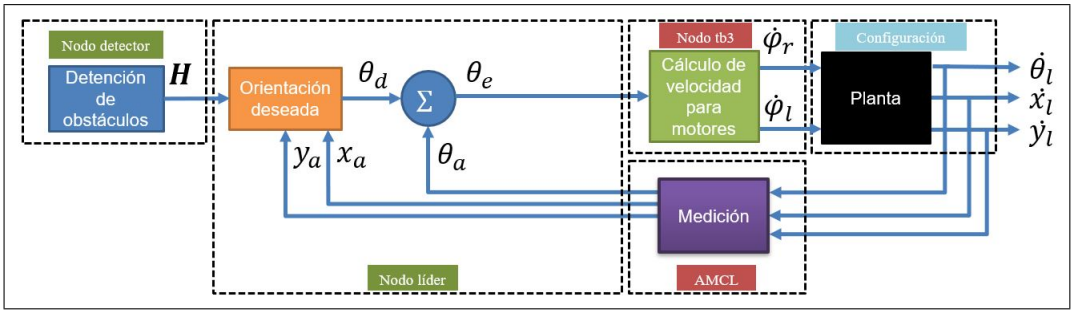

Figura 2.56: Esquema general de funcionamiento del sistema de control para el seguimiento de trayectorias y evasión de obstáculos.

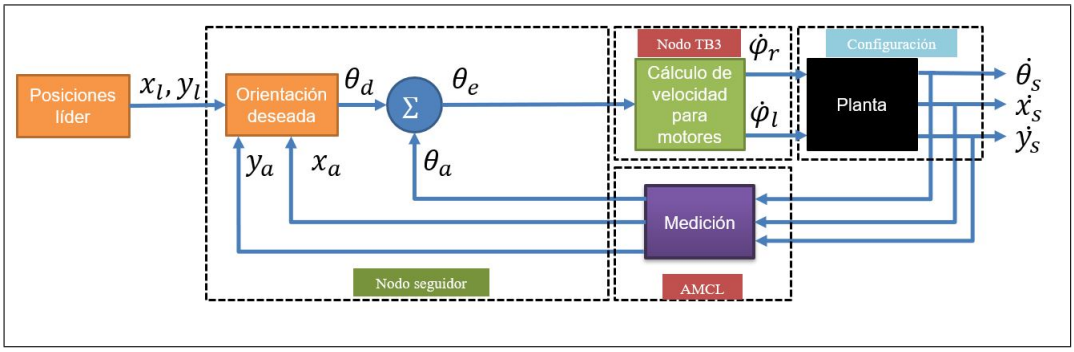

Figura 2.57: Esquema general de funcionamiento del sistema de control para el seguidor.

La manera de formar esta integración es muy directa, ya que dentro de cada nodo hay objetos que se suscriben a tópicos, y otros que publican la información procesada, por lo que estos de manera natural pueden y están hechos para trabajar en conjunto. Para mandar a llamar todos los nodos se hace uso de otro tipo de archivo de ROS, archivos con extensión .launch los cuales permiten mandar a llamar nodos

de manera automática. Para lograr la integración que se propone en los diagramas de las Figuras 2.56 y 2.57, se debe primero llamar el nodo que inicializa los motores y el sensor LIDAR, el cual viene por defecto en los Turtlebot 3 burger, posteriormente se debe mandar a llamar el mapa para la localización, una vez inicializado el mapa se llaman los nodos líder y seguidor, en los cuales se ejecuta la parte medular del proyecto, finalmente se llaman por medio de otro launch el localizador del líder y el localizador del seguidor, una vez inicializados todos los nodos antes mencionados se lleva a cabo la función principal del proyecto.

Los diagramas antes presentados muestran la integración de los nodos, sin embargo, dentro de cada nodo existen tópicos (variables compartidas) las cuales también forman parte de la integración de dichos códigos, el siguiente diagrama en la Figura 2.58 muestra como esos tópicos se comunican y cómo van transformando la información cada vez que pasan por algún nodo.

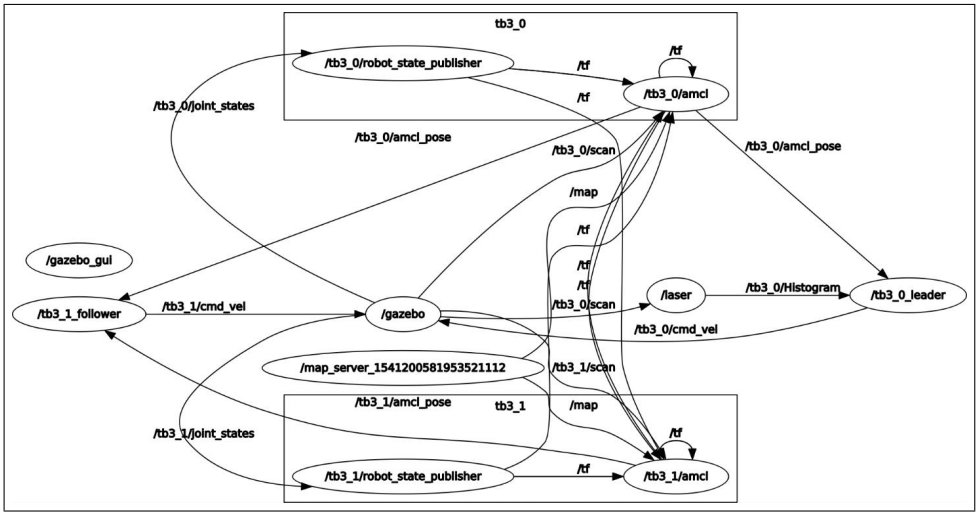

Figura 2.58: Comunicación entre tópicos de los distintos nodos.

2.2.7. HMI

La interfaz o HMI por sus siglas en inglés (Human Machine Interface) forma parte indirectamente del área funcional de Procesamiento como se puede observar

en la Figura 2.9 en la página 33, esto debido a que aunque la interfaz no se encuentre dentro de los robots, trabaja directamente con el procesamiento de los móviles, y se caracteriza por ser el puente de comunicación entre el usuario, el robot líder y el robot seguidor, en donde el usuario seleccionará y enviará a través de una computadora datos acerca de una tarea a ejecutar (en este caso la selección de una trayectoria), y el robot líder decodificará y ejecutará dicha tarea en conjunto con el robot seguidor al cual también se le dirá en que momento detenerse si así lo desea el usuario.

2.2.7.1. Comunicación peer to peer

Como ya se ha hablado con anterioridad en el glosario, el uso del término "nodo"dentro de ROS, ejecuta una serie de procesos conectados en un mismo tiempo de ejecución a un determinado número de anfitriones, dependiendo del número de nodos. Cuando muchos nodos se están ejecutando al mismo tiempo, se realiza una comunicación "peer-to-peer", que, en otras palabras, es la topología de enlace entre los distintos nodos. Este tipo de topología requiere de un maestro que permita que los procesos se encuentren los unos con otros. Esto se ejemplifica en la Figura 2.59.

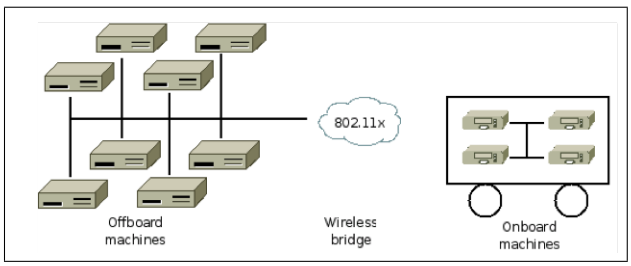

Figura 2.59: Conexión típica de ROS.

Para el desarrollo de la interfaz gráfica es importante entender como trabaja este tipo de comunicación, ya que la interfaz gráfica que se implementará desde una computadora funcionará como anfitrión (teniendo un único anfitrión), que enviará mensajes mediante la publicación en un tópico (entendiéndose tópico como el intercambio de información de los nodos) y llamando a los nodos interesados a

procedimientos remotos, suscribiendo su información a el tipo de tópico en proceso como se muestra en la Figura 2.60.

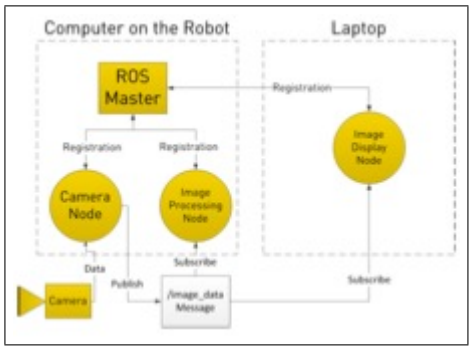

Figura 2.60: Ejemplo de comunicación de ROS con agentes externos.

La Figura 2.59, sirve como ejemplo para el desarrollo de la interfaz (agente externo) y su comunicación con ROS, ya que a diferencia de utilizar una cámara como se muestra en el ejemplo de la Figura 2.60, la propuesta para la recabación de los datos exteroceptivos de cada robot móvil, es mediante la implementación de un sensor LIDAR y que en la parte de percepción se aborda a detalle, pero la lógica del diagrama es la similar.

Dentro del Meta Sistema Operativo ROS, existe algunas herramientas para poder desarrollar e implementar interfaces gráficas siendo Qt un ejemplo de estas.

2.2.8. Validación y análisis de resultados

Es de suma importancia que en cualquier proceso de investigación y diseño antes de la construcción exista una validación, ya que se corre el riesgo de que las propuestas de diseño diverjan de la realidad, no sean factibles o simplemente infuncional, es por esta razón que se necesita realizar una simulación lo más cercana a la realidad en la que se puedan probar los algoritmos implementados en el área funcional de procesamiento. El objetivo que se persigue en esta validación es determinar si los algoritmos implementados en un Turtlebot 3 burger virtual cumplen su función o no.

2.2.8.1. Procesamiento

Cada validación consistió en la programación de los nodos propuestos para cada sección del área funcional en el lenguaje de programación Python; del cual es relevante mencionar que es compatible con las tarjetas de procesamiento de los Turtlebot 3 burger. Una vez hechos los programas fueron implementados en un ambiente de simulación llamado *Gazebo* incluyendo allí el modelo 3D del Turtlebot 3 burger, con sus características físicas y restricciones mecánicas como lo son velocidad máxima en cada motor. Posteriormente los datos de cada una de las simulaciones fueron grabados con el comando de ROS (Rosbags) y graficados con ayuda de Excel. Dichos datos son presentados graficados en las siguientes secciones junto con un breve análisis. Todos los datos aquí graficados pueden ser consultados en la parte de apéndices.

2.2.8.2. Seguimiento de trayectoria

Lo primero por validar en el proyecto es el funcionamiento del generador de la matriz de fuerza para esta validación se propusieron dos trayectorias, las cuales se muestran en las Figuras 2.61 y 2.62, donde cada eje coordenado tiene dimensiones en metros, una vez propuestas las trayectorias se llevó a cabo la generación de la matriz de fuerzas como se observa en las Figuras 2.63 y 2.64, estas figuras también tiene sus ejes coordenados en metros. Como se puede observar de las gráficas 2.63 y 2.64 los vectores generados siguen las trayectorias propuestas cuando estan dentro de la trayectoria y se orientan al punto más cercano cuando estan fuera de trayectoria, por lo cual se toma como válido el algoritmo.

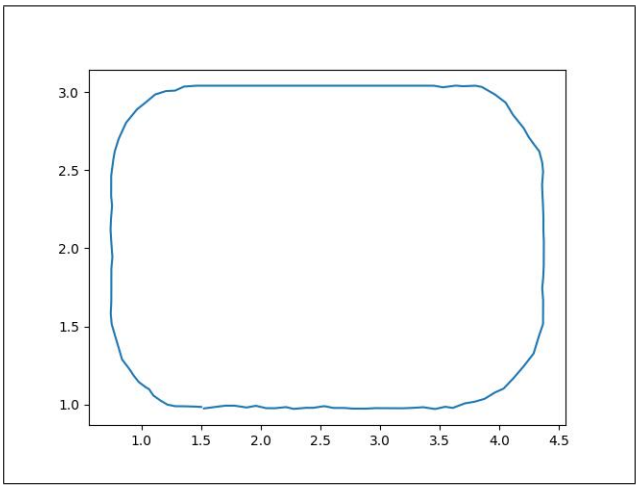

Figura 2.61: Trayectoria propuesta 1 (unidades en metros).

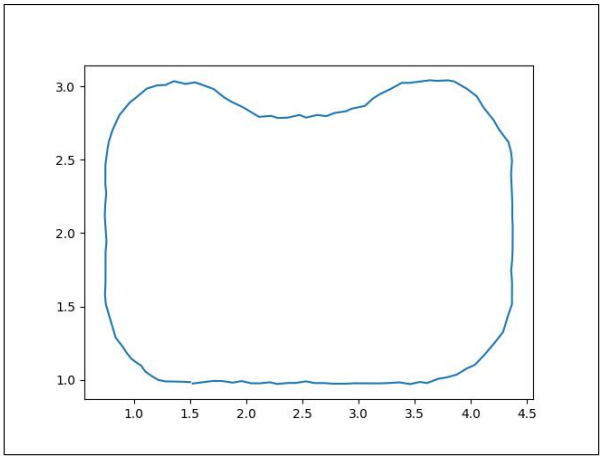

Figura 2.62: Trayectoria propuesta 2 (unidades en metros).

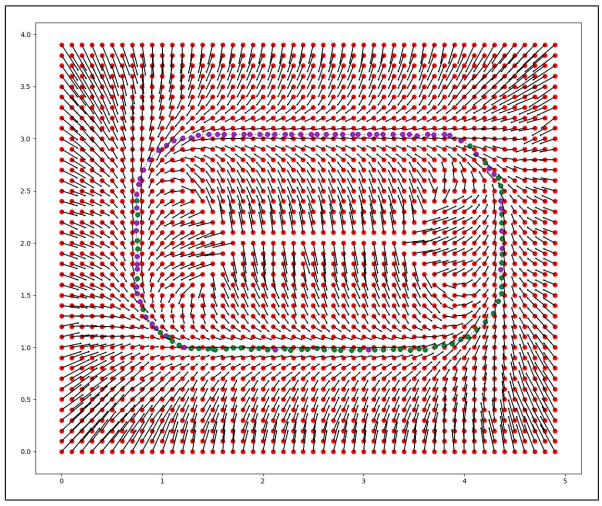

Figura 2.63: Matriz de fuerza para trayectoria propuesta 1 (unidades en metros).

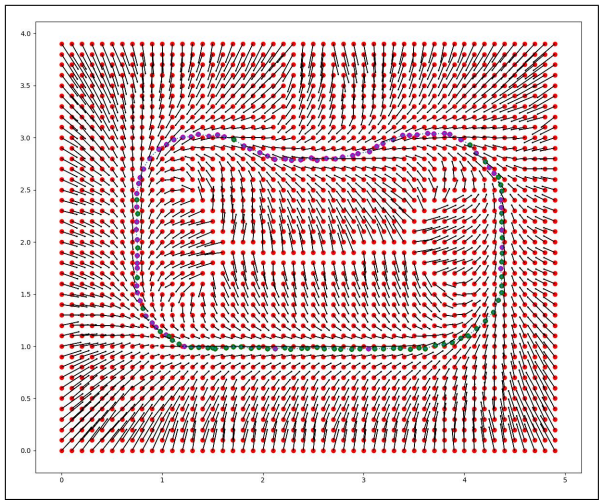

Figura 2.64: Matriz de fuerza para trayectoria propuesta 2 (unidades en metros).

A continuación se validará que el Turtlebot 3 burger en el ambiente virtual sea capaz de seguir la trayectoria de la Figura 2.61. Recordando que la simulación es realizada en Gazebo y los datos son grabados y posteriormente presentados en gráficas hechas con Excel. Para esta simulación se hizo uso del nodo líder, nodo AMCL y nodo tb3 virtual. Los datos obtenidos de la simulación se muestran en la Figura

2.66.

Figura 2.65: Resultados extraídos de la simulación de seguimiento de trayectoria (unidades en metros).

La Figura 2.66 muestra los puntos de ruta alcanzados por el líder durante la simulación, en dicha gráfica cada eje coordenado se representa en metros. Para validar esta parte de procesamiento se hace una comparación entre las Figuras 2.66 y 2.61, a simple vista se puede observar que los puntos representados en la Figura 2.66 se asemejan a la trayectoria propuesta en la Figura 2.61, por lo tanto, se valida el funcionamiento de algoritmo de seguimiento de trayectoria.

2.2.8.3. Líder Seguidor

Para validar el esquema líder seguidor se inició haciendo que el líder siguiera una de las trayectorias propuestas, con todos los nodos mencionados en la validación de seguimiento de trayectorias y después ejecutar el nodo del seguidor. Los resultados de la simulación son mostrados en las Figuras 2.66 y 2.67, las cuales tienen las mismas dimensiones. Comparando las coordenadas por las que el líder y seguidor pasan se valida el diseño de este nodo ya que las coordenadas son muy similares, entre líder y seguidor.

Figura 2.66: Resultados extraídos de la simulación del líder (unidades en metros).

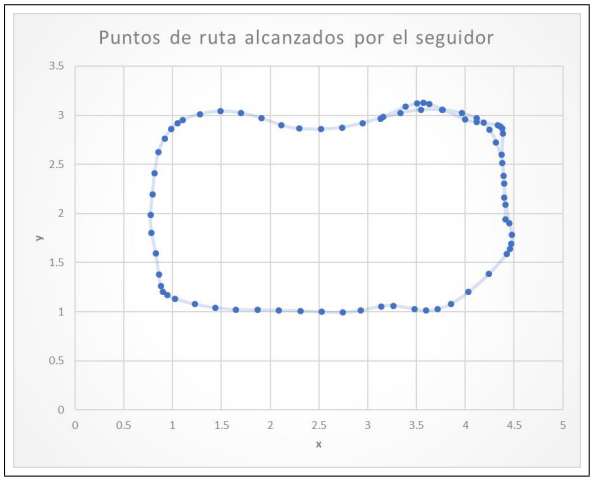

Figura 2.67: Resultados extraídos de la simulación del seguidor (unidades en metros).

2.2.8.4. Evasión de obstáculos

Para la evasión de obstáculos en el ambiente de simulación se colocaron dos objetos por donde se tenía previsto que pasara el robot líder, Figura 2.68, de acuerdo con la trayectoria que debía seguir, posteriormente se ejecutó el seguimiento de trayectoria y la evasión, los puntos por los que pasó el líder se muestran en Figura 2.69.

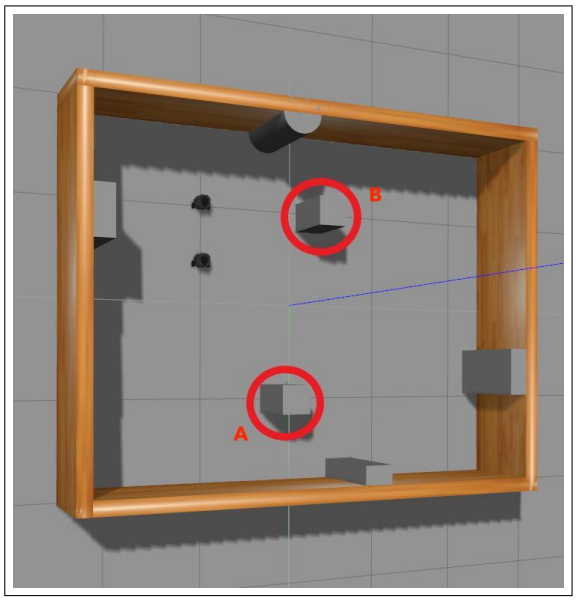

Figura 2.68: Obstáculos para la validación del algoritmo propuesto.

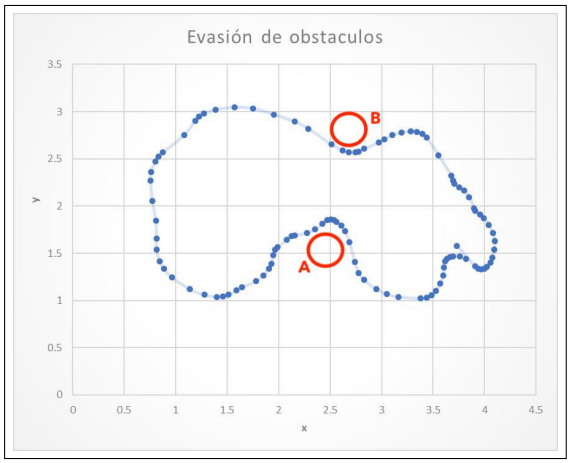

Figura 2.69: Puntos de ruta alcanzados por el líder durante la evasión de obstáculos (unidades en metros).

Durante la ejecución el robot líder fue capaz de evadir efectivamente los obstáculos, y cuando no había obstáculo se reincorporaba a la trayectoria nuevamente, por lo tanto se valida este algoritmo ya que cumple con la función de la evasión.

Integración del Sistema

3.1. Implementación del Área Funcional Estructura

La resolución de está área funcional estuvo relacionada con el armado del robot, que como ya se había mencionado posteriormente en capítulos pasados, está área funcional tenía como requerimiento alberga a todas las demás áreas funcionales que conforma a cada robot, lo cual no presentó mayor dificultad porque cada robot ya viene diseñado con tal objetivo para únicamente centrarse en armar.

A continuación se detalla cada parte del proceso.

3.1.1. Armado de los robots móviles

Figura 3.1: Caja con los diferentes componentes del robot.

Los componentes del turtlebot3 burger vienen en 4 cajas enumeradas y mostradas en la Figura 3.1, cada caja, con distintas piezas y componentes. En la *caja 1* como se muestra en la Figura 3.2, viene una tarjeta microSD de 16GB, una tarjeta Raspberry Pi 3, un par de motores Dynamixel y una tarjeta OpenCR Cortex M7. En la *caja 2* mostrada en la Figura 3.3 viene una bateria LiPo junto con su cargador, un desarmador, un sensor LIDAR, un conector USB2LDS y soportes para pcb's. En la *caja 3* como se muestra en la Figura 3.4 vienen todos los pisos del robot denominados "Waffle - Plate". En la *caja 4* mostrada en la Figura 3.5 viene el sistema de tracción, es decir, las llantas, sus respectivas ruedas, así como los tornillos y piezas

de ensamble de todo el robot, cables y una rueda loca.

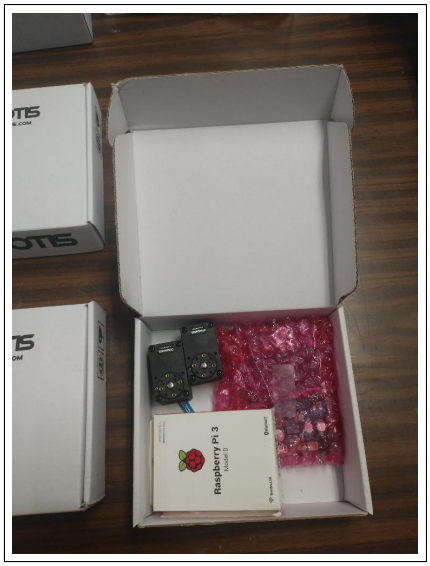

Figura 3.2: Caja de componentes 1. *MicroSD, Raspberry Pi 3, motores Dynamixel y Tarjera OpenCR Cortex M7*

Figura 3.3: Caja de componentes 2. *Bateria LiPo, cargador, desarmador, sensor LIDAR, conector USB2LDS y soportes para pcb's*

Figura 3.4: Caja de componentes 3. *"Waffle - Plate"pisos de los robots*

Figura 3.5: Caja de componentes 4. *Llantas, ruedas, tornillos, piezas de ensamble, cables y una rueda loca*

Además de estas cuatro cajas, también viene en una caja más no numerada donde viene la fuente de voltaje para el cargador con su respectivo cable como se muestra en la Figura 3.6.

Figura 3.6: Fuente de voltaje para bateria LiPo.

La primera parte del armado consistió en unir cada uno de los pisos con ayuda de sus respectivos tornillos, así como también de sus distintos componentes, como se muestra en las Figuras 3.7 y 3.8.

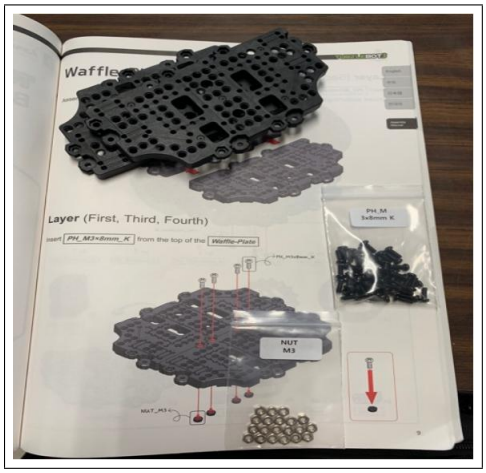

Figura 3.7: Unión de los pisos del robot.

Figura 3.8: Piezas Ensambladas.

Los problemas que en esta parte se presentaron fueron que a pesar de que en el instructivo de armado viene paso por paso como van las piezas y cada una de las tapas, muchas veces este no era tan claro aunado a que todas las partes del robot son muy parecidas, dando lugar a piezas mal colocadas y por ende mal ensambladas a la hora de unir cada uno de los pisos.

Figura 3.9: Ensamble del robot.

3.1.2. Instrucciones de Armado

Para el armado de estructura y de las partes electrónicas se siguieron las instrucciones tal y como se indican el manual de ensamble del turtlebot3 burger [28], con algunos consejos que a continuación se mencionan.

- Antes de armar el turtlebot3 burger tener todo el software de inicio instalado.
- En el caso particular de la Raspberry Pi 3 usar un cargador de 5.0V a 2A para instalar el sistema operativo.
- Tener un desarmador imantando como el proporcionado que sujete bien los tornillos.
- Ir verificando que cada capa este bien sujetada.
- Hacer las pruebas de hardware para cada capa.
- Colocar de manera correcta todos los cables de conexión por los orificios indicados.
- Armarla en un lugar seguro en cual no pierdan piezas pequeñas.
- Consultar el video de armado para un mejor armado.

3.1.3. Resultados del proceso de armado

En las Figuras 3.10, 3.11, 3.12 y 3.13 se puede observar como se ve en físico cada uno del los pisos del turtlebot3 burger así como el orden en el que fueron ensamblados. .

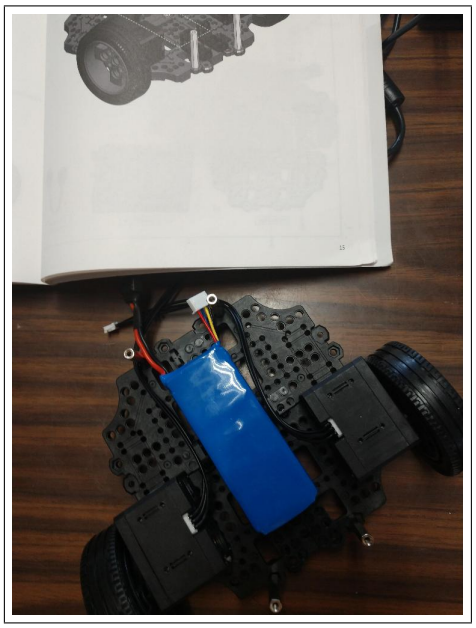

Figura 3.10: Ensamble del primer piso.

Figura 3.11: Ensamble del segundo piso.

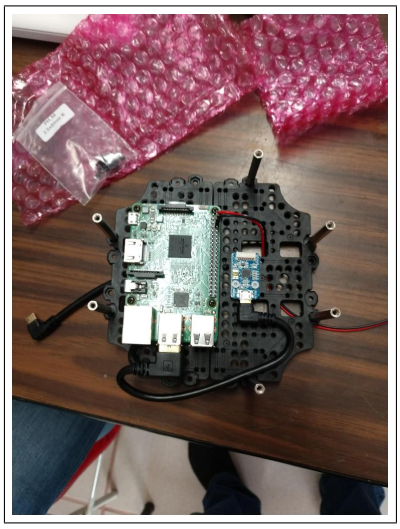

Figura 3.12: Ensamble del tercer piso.

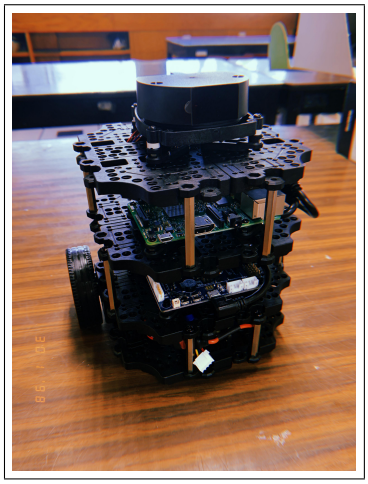

Figura 3.13: Ensamble final del robot.

Cabe destacar que el armado de cada robot representó un trabajo muy minucioso ya que cada tornillo y tuerca tenían que estar en el lugar adecuado, cada piso y componente por lo mismo la tarea duró aproximadamente 5 horas de trabajo contínuo para cada turtlebot3 burger.

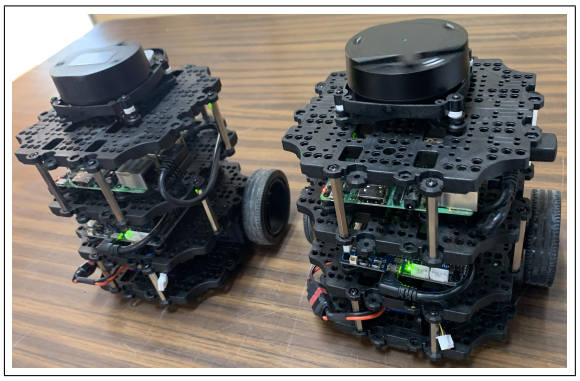

Figura 3.14: Robot líder y robot seguidor seguidor.

3.1.4. Verificación de ensamble del sistema

Una vez que el robot estuvo armado verificamos que todo estuviera bien conectado, es decir, que la batería LiPo alimentara a las dos tarjetas de procesamiento (Raspberry y OpenCR) y que estas alimentaran a los subsistemas que de ellas dependen como lo son los actuadores y el sensor LIDAR. El trabajo en conjunto del ensamble del robot junto con la programación e instalación de Raspbian a la Raspberry Pi 3.

3.2. Implementación del Área Funcional Movimiento

Esta área funcional es la encargada de la unión de la estructura junto con su configuración mecánica. En la implementación se utilizaron 2 actuadores Dynamixel por cada robot conectados a una tarjeta OpenCR denominada tarjeta de control. Para ajustar el PID encargado de mantener estable las velocidades de los robots no fue necesario realzar un calculo de valores, ya que los motores contaban con la opción de autotune que permitía de forma automática obtener los valores de PID necesarios para lograr un control de velocidad y estos ya estaban previamente configurados, con un valor **P** de 1.1, **D** de 0.03 e **I** de 0.11.

Figura 3.15: Area funcional movimiento, que cuenta con los dos motores Dynamixel y la OpenCR.

3.3. Implementación del Área Funcional Percepción

En la implementación de la tercera área funcional; percepción se utilizan 2 sensores (sensor LIDAR, encoder del actuador) que en conjunto logran percibir el entorno brindando así la posibilidad de conocer la posición de cada robot y determinar si hay un objeto cercano al robot para que este lo pueda esquivar.

3.3.1. Sensor LIDAR

El primer sensor involucrado es el sensor LIDAR, y por medio de este sensor se miden las distancias mínimas de cercanía hacia algún objeto (para este proyecto puede ser una pared u obstáculo), para cada uno de los 360 grados que rodean al robot.

Para su implementación este sensor fue colocado en la capa superior de cada móvil como se observa en Figura 3.16, destacando que cada uno de estos sensores es

independiente del otro y al verificar su correcto funcionamiento en robot se deben aplicar los mismos pasos para el otro robot, posteriormente se realizaron las pruebas en la interfaz gráfica proporcionada por el fabricante (previamente realizando la configuración básica del turtlebot3 burger a verificar, descrita a fondo en procesamiento) y con los resultados de la interfaz gráfica la cual muestra las distancias del sensor LIDAR reflejadas en un mapa 2D, podemos comprobar que el sensor esta trabajando correctamente.

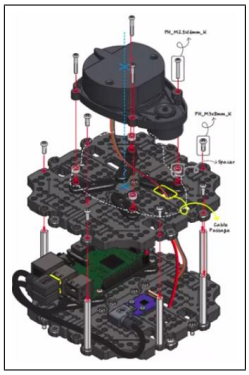

Figura 3.16: Montaje del sensor LIDAR en la estructura.

Como se puede observar en la Figura 3.16 se debe tener cuidado en la orientación en la que se coloca el sensor LIDAR, colocándolo de tal forma como se muestra en la Figura 3.16. Esto debido a que el sistema de referencia entre el móvil y los datos arrojados en la interfaz, los cuales son propuestos para que el sensor LIDAR tenga su origen en 0 grados en la parte frontal del móvil, sino respetara está condición provocaría un desfase en el ángulo lo cual afectaría en la realización del mapa del entorno y que posteriormente se vería reflejado en un desfase angular en los demás móviles que no tuvieran la misma orientación que el móvil con el cual se realizó el escaneo del mapa.

Es importante también señalar que el sensor LIDAR como el mostrado en la Figura 3.17 cuenta al frente con un indicador de fase, el cual le comunica al controlador del LIDAR que ha pasado por el 0 absoluto del sensor, esto se realiza ya que no se

sabe certeramente en que posición esta apuntando el del sensor LIDAR cuando se inicia el escaneo.

Figura 3.17: Sensor LIDAR del Turtlebot3 burger.

Para comprobar que el sensor esta funcionando correctamente se cuenta con un programa que inicializa los nodos necesarios para el funcionamiento básico del robot, el cual comunica el robot con la computadora principal además de obtener y mandar datos al sensor LIDAR y a los motores por medio de la tarjeta OpenCR.

Las instrucciones para correr esta verificación son las siguientes:

- roscore (En la PC principal)
- roslaunch turtlebot3 turtlebot3_bringup.launch (En la Raspberry Pi 3)
- roslaunch turtlebot3_slam turtlebot3_slam.launch (En la PC principal)

Al correr estos comandos habiendo configurado correctamente la comunicación por ROS entre el turtlebot3 y la PC principal y estando en la misma red con mismos parámetros se obtuvieron los resultados mostrados en la Figura 3.18, donde los puntos en amarillo son las distancias escaneadas por el LIDAR.

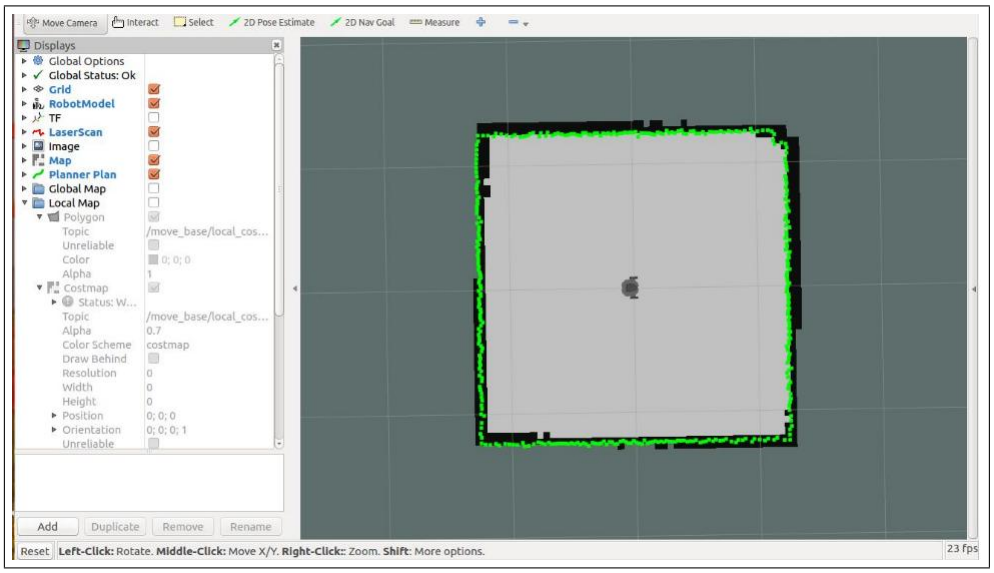

Figura 3.18: Distancias del LIDAR en RVIZ

Adicionalmente se pueden ver los valores de las distancias si ejecutamos el comando:

- rqt (En la PC principal)

Al abrirse la ventana como se muestra en la Figura 3.19 se podrán observar los valores de los mensajes publicados y si se marca la casilla se puede leer y ver su valor en pantalla, para verificar los datos que da el sensor se marca la casilla /scan, la cual si no aparece significa que se tiene problemas en el sensor LIDAR o en la ejecución de los comandos.

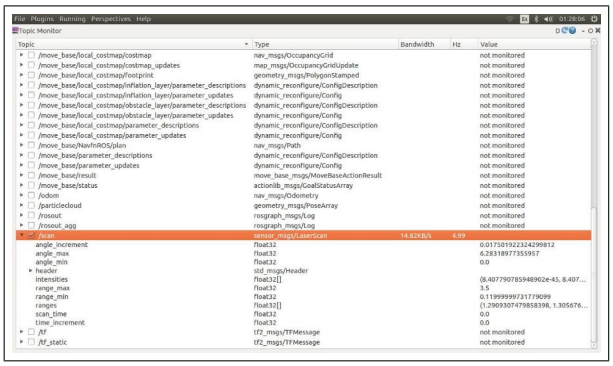

Figura 3.19: Visualización en rqt de los valores del sensor LIDAR.

3.3.2. Sensores del motor DYNAMIXEL

El segundo sensor involucrado es el encoder contenido en los motores DYNAMIXEL mostrado en la Figura 3.20, este enconder lleva a cabo la tarea en conjunto con el microcontrolador del motor DYNAMIXEL, de estimar del desplazamiento que ha tenido el robot.

Figura 3.20: Motores DYNAMIXEL XL430.

Para la implementación de este sensor se colocan los 2 motores en la estructura conectados a la tarjeta OpenCR como se muestra en la Figura 3.21.

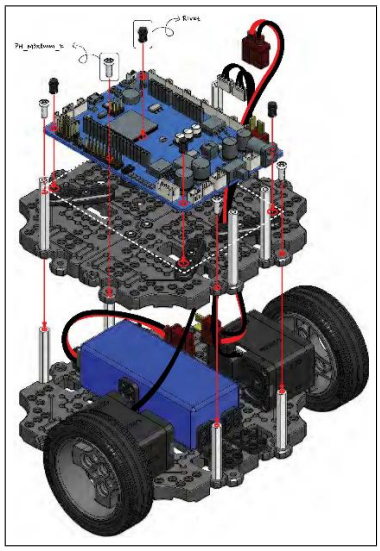

Figura 3.21: Ensamble de los motores, OpenCR y Batería LiPo.

Una vez ensamblada esa capa se procedió a verificar el funcionamiento de los motores así como él de los enconders por medio del programa base que se le carga a la OpenCR. Este programa hace dos pruebas con los motores una para lograr un desplazamiento lineal y otra para un desplazamiento angular (giro de 180° conforme al centro del robot) en la posición del robot.

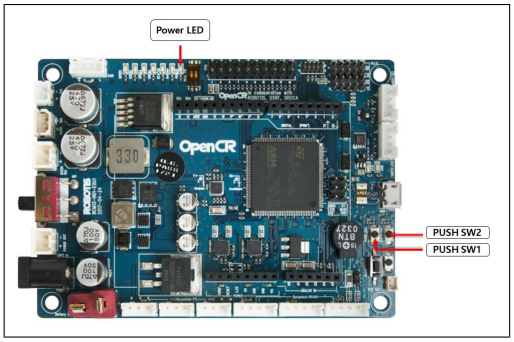

Figura 3.22: Localización de los botones de prueba en la tarjeta OpenCR.

El proceso de verificación del correcto funcionamiento de los encoders y el giro de los motores es el siguiente :

- Revisar que todos los elementos estén ensamblados al menos hasta la capa 2.
- Encender el Robot por medio del switch que trae consigo la tarjeta Open CR.
- Colocar el robot en una zona que no choque y a su vez no se caiga como una mesa.
- Presionar por 3 segundos el botón PUSH SW1 visto en la Figura 3.22 para verificar que el robot avance alrededor de 30 cm.
- Presionar por 3 segundos el botón PUSH SW2 visto en la Figura 3.22 para verificar que el robot gire 180° grados sobre su propio eje.

Si estas 2 pruebas se cumplen entonces los encoders están funcionando correctamente, ya que nos están dando la distancia desplazada real.

3.4. Implementación del Área Funcional Alimentación

Para la implementación de esta área funcional se usó una batería LiPo de11.1V y 1800mAH vista en la Figura 3.23, la cual se carga totalmente aproximadamente en 1 hora y media, y es capaz de mantener en operación al robot por aproximadamente 2 horas en funcionamiento con uso moderado.

Figura 3.23: Batería LiPo LB-012.

Para verificar su correcto funcionamiento solamente se conectó al móvil siguiendo los primeros pasos de ensamble como se muestra en la Figura 3.21 y se verificó que el Power LED de la OpenCR estuviera prendido, este LED se muestra en la Figura 3.22.

3.5. Implementación del Área Funcional Procesamiento

3.5.1. Pasos previos

La sección de procesamiento requiere una configuración previa de las tarjetas y de la computadora remota. Todos los pasos que son necesarios para tener en funcionamiento los turtlebot3 burger se encuentran en su manual online, a continuación, se presentan algunos consejos de implementación o pasos adicionales que se tuvieron que realizar.

En la computadora remota se instaló Ubuntu16.04, muchos son los tutoriales que explican cómo hacerlo y dependiendo de la computadora puede ser más o menos complejo, una buena fuente de referencia es la de la página web linuxtechi [16]. La mejor forma de instalar ROS sobre la computadora remota es con el método que propone el manual online de los turtlebot3, todo el proceso es automático, ya que a diferencia de los métodos convencionales donde es necesario ejecutar una seria de comandos manualmente, este manual proporciona un script que va ejecutando los pasos uno después de otro hasta finalizar la instalación, además de que descarga todos los paquetes necesarios para el funcionamiento de los robots al ejecutar todas las instrucciones que nos dice.

Para la Raspberry Pi 3 existen dos sistemas operativos compatibles con ROS, Raspbian y UbuntuMate, para este caso se optó por Raspbian ya que la forma de configurar el SSH está bien establecida en la documentación oficial de la Raspberry Pi [22], además en el mismo manual , se proporciona una versión de Raspbian con ROS instalado, lo cual reduce en aproximadamente en 2 horas el tiempo de instalación

por tarjeta.

Para la configuración de OpenCR, igual existen dos formas de configurarla, la primera por medio de de la computadora remota y sirve cuando se está ensamblando por primera vez el turtlebot3 burger y la segunda directo de la Raspberry Pi 3, la segunda opción es la mas sencilla cuando ya se tiene ensamblado el turtlebot3 burger pues no se tiene que desconectar. Para llevar a cabo este proceso en la Raspberry Pi 3 la cual está conectada por USB a la tarjeta OpenCR, se ejecuta el comando siguiente, el cual descarga e instala el programa que va sobre la tarjeta OpenCR:

https://github.com/ROBOTIS-GIT/OpenCR-Binaries/raw/master/turtlebot3/ROS1 /latest/opencr_update.tar.bz2&& tar -xvf opencr_update.tar.bz2 && cd ./opencr_update && ./update.sh $OPENCR_PORT $OPENCR_MODEL.opencr && cd ..

Para desarrollar la aplicación de una forma más fluida es necesario tener conocimientos previos en ROS y sus extensiones que facilitan las tareas de simulación y visualización de información como lo son Gazebo, Rviz y RQt. Todas estas extensiones se instalan predeterminadamente si se elige instalar la versión completa de ROS por lo que es recomendable hacerlo en la computadora principal. Las herramientas de ROS más usadas en el proyecto fueron los mensajes (messages), suscriptores (subscribers) y publicadores (publishers), y por medio de estas herramientas es como podemos modularizar y compartir información entre nodos.

Lo siguiente en la implementación fue hacer el modelo 3D del ambiente en que los robots virtuales serian simulados. Para ello se usó Gazebo, y los pasos para hacerlo fueron, en la opción de editar se seleccionó la figura que se asemeja a una pared, se trazó con las medias correctas y después en las opciones de cada una de las paredes, fue modificado su aspecto visual y altura, finalmente se guardó el diseño.

También fue necesario configurar RViz para poder visualizar los datos de los dos robots en tiempo de ejecución, la manera de configurarlo fue editar el archivo base que proporcionan las librerías de Turtlebot3 burger y eliminar algunos tópicos que

no eran necesarios para esta aplicación, los cuales son:

- /cost_map
- /global_cost_map

Además, se agregaron los tópicos del seguidor, que a continuación se enlistan:

- /tb3_1/robot_model
- /tb3_1/laser
- /tb3_1/amcl_particles

Adicionalmente con ayuda de AMCL, una de las librerías de ROS, se realiza el mapa del ambiente virtual y del espacio real, esto con el fin de usarlo para posteriormente obtener su posición con ayuda de los sensores. Los comandos, que también estan presentes en el manual y se ejecutan sobre la terminal de la computadora remota son:

- roslaunch turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
- roslaunch turtlebot3_teleop turtlebot3_teleop_key.launch

3.5.2. Comunicación

Para establecer la comunicación entre los robots y la computadora central, se siguieron los pasos como se indica en el manual online de los Turtlebot3 burger, también fue requerido configurar la red de la computadora remota como un punto de acceso. Lo primero por hacer es crear una nueva red, mostrada en la Figura 3.24.

Figura 3.24: Creación de una nueva red en Ubuntu.

Posteriormente se seleccionó el tipo de conexión como inalámbrica como se muestra en la Figura 3.25.

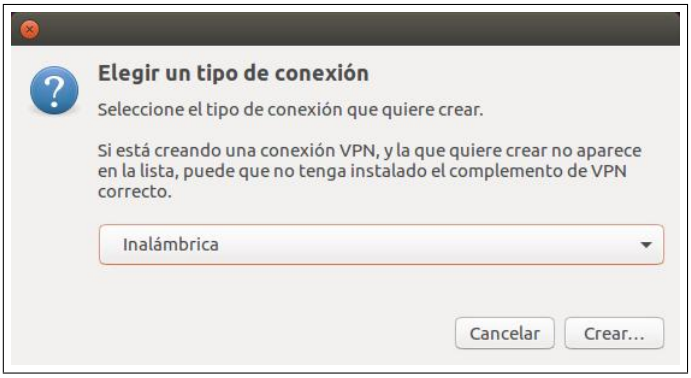

Figura 3.25: Elegir tipo de conexión.

Y por último se editaron las pestañas de las configuraciones inalámbricas y de seguridad como se muestran las Figuras 3.26 y 3.27.

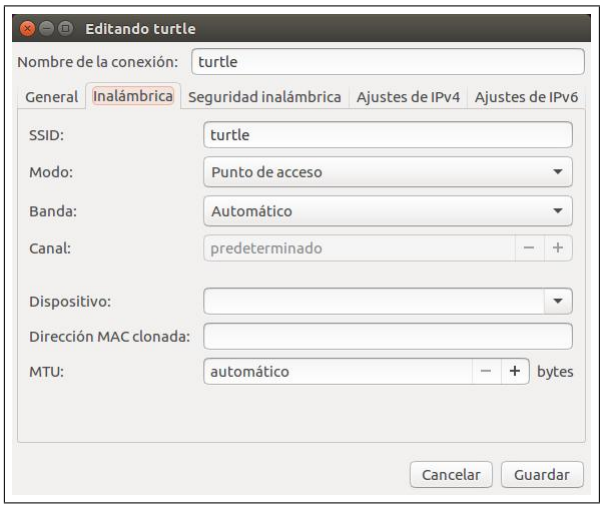

Figura 3.26: Configuración del Modo y Nombre.

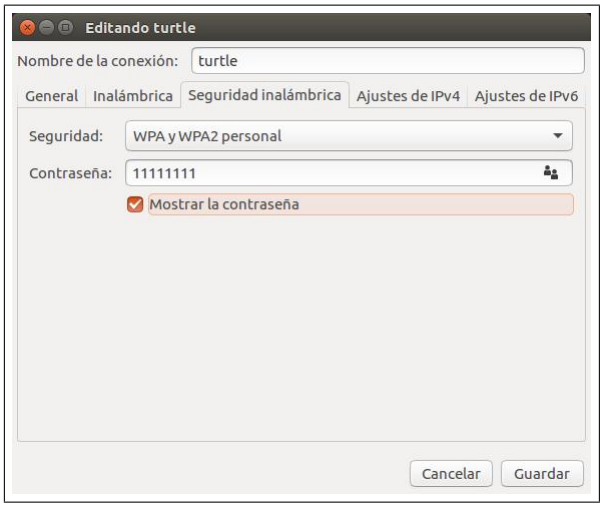

Figura 3.27: Configuración de la seguridad.

Una vez establecida la comunicación se procedió a realizar pruebas con el propósito de determinar si la comunicación fue establecida correctamente. Para ellos se utilizó el comando ping de la terminal de Linux que envía paquetes de forma continua a cada turtlebot3 burger como se muestra en la Figura 3.28.

```
bob@suse1:~> ping 192.168.198.130
PING 192.168.198.130 (192.168.198.130) 56(84) bytes of data.
64 bytes from 192.168.198.130: icmp_seq=1 ttl=64 time=6.14 ms
64 bytes from 192.168.198.130: icmp_seq=2 ttl=64 time=0.778 ms
64 bytes from 192.168.198.130: icmp_seq=3 ttl=64 time=0.599 ms
64 bytes from 192.168.198.130: icmp_seq=4 ttl=64 time=0.558 ms
64 bytes from 192.168.198.130: icmp_seq=5 ttl=64 time=0.615 ms
64 bytes from 192.168.198.130: icmp_seq=6 ttl=64 time=0.608 ms
64 bytes from 192.168.198.130: icmp_seq=7 ttl=64 time=0.645 ms
64 bytes from 192.168.198.130: icmp_seq=8 ttl=64 time=0.619 ms
64 bytes from 192.168.198.130: icmp_seq=9 ttl=64 time=0.698 ms
^C
--- 192.168.198.130 ping statistics ---
9 packets transmitted, 9 received, 0% packet loss, time 8000ms
rtt min/avg/max/mdev = 0.558/1.252/6.149/1.732 ms
```

Figura 3.28: Muestra del resultado correcto de la ejecución del comando ping.

Como se pudo apreciar en la Figura 3.28, la comunicación fue establecida correctamente ya que se recibieron paquetes correctamente. Finalmente se prueba la comunicación de algunos tópicos simples, para verificar que la configuración entre la red y ROS es correcta, para ello se inicializa por medio de la computadora remota un turtlebot3 burger.

Además con apoyo de RQt, se puede visualizó el valor actual de algunos tópicos, en este caso se eligió el estado de la batería.

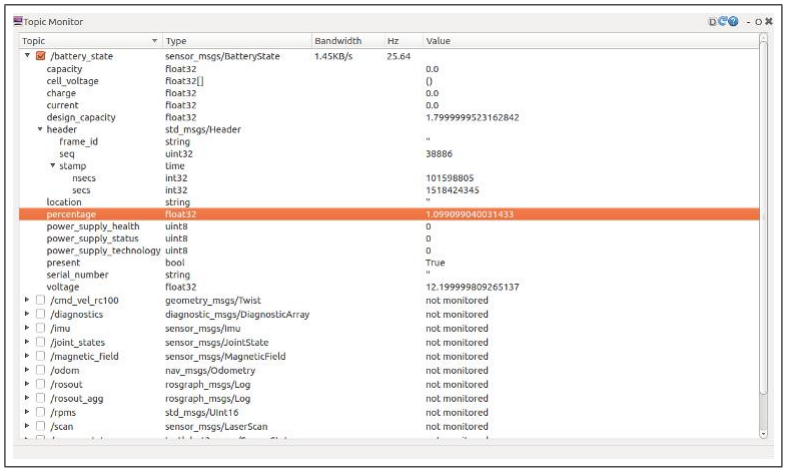

Figura 3.29: RQt mostrando los datos del tópico battery.

Para la implementación de los algoritmos de control, todos fueron programados en el lenguaje Python con la lógica que fue propuesta en los diagramas de flujo, posteriormente fueron almacenados en un mismo paquete al que llamamos LiderSeguidor,

con el objetivo de mantener un orden de la información para realizar la ejecución del algoritmo, el cual fue creado especialmente para este proyecto. Para manejar la información de forma independiente fue necesario agrupar por namespaces todos los tópicos propios de cada uno de los turtlebot3 burger, ya que en un principio cada turtlebot3 burger posee tópicos con el mismo nombre, en el caso de no usarse el namespace para nombrar los tópicos de forma distinta resulta muy complejo controlar los robots de forma individual ya que no se puede saber si la información se le enviá al líder o al seguidor.

3.5.3. Prueba A: Verificación de la matriz de fuerza

El objetivo de esta prueba fue verificar que la matriz fuese hecha correctamente, se prueba con un código que reconstruye la trayectoria a partir de la matriz, si la trayectoria tiene la forma de la original se puede asegurar que fue hecha correctamente. Las Figuras 3.30, 3.31 y 3.32 muestran la reconstrucción, mientras que las 3.33, 3.34 y 3.32, muestran la trayectoria original.

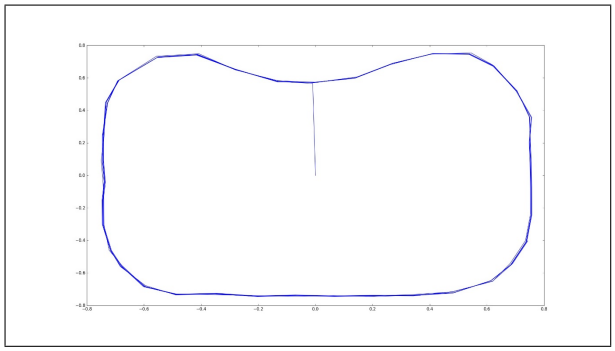

Figura 3.30: Reconstrucción de la trayectoria A.

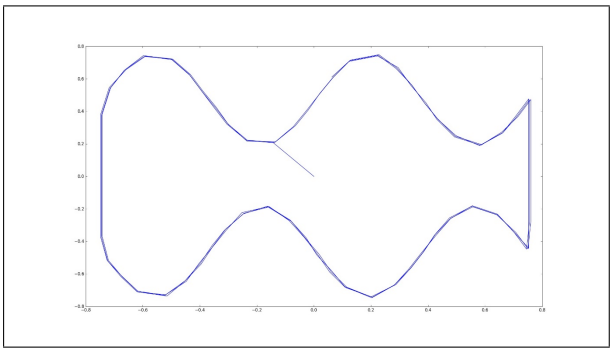

Figura 3.31: Reconstrucción de la trayectoria B.

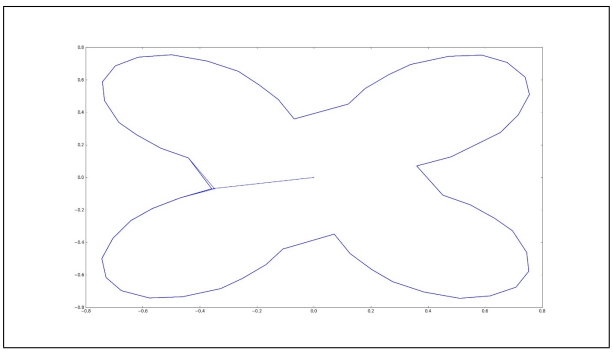

Figura 3.32: Reconstrucción de la trayectoria C.

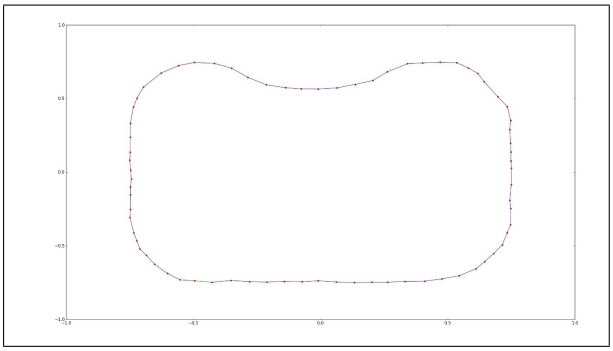

Figura 3.33: Puntos de trayectoria A.

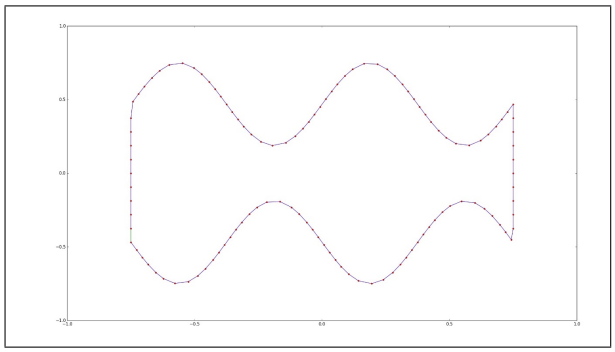

Figura 3.34: Puntos de trayectoria B.

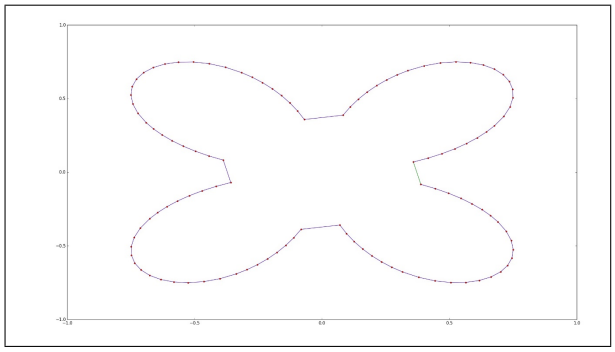

Figura 3.35: Puntos de trayectoria C.

3.5.3.1. Análisis de resultados

De las Figuras 3.30, 3.31, 3.32, 3.33, 3.34, 3.35 se logró observar que la reconstrucción conserva la forma de la trayectoria, lo cual indica que las matrices fueron hechas de forma correcta.

3.5.4. Prueba B: Validar el seguimiento de trayectoria

Una vez que la matriz de fuerza fue verificada, se procedió con la prueba del seguimiento de trayectoria. La prueba consistió en el usó del ambiente de simulación hecho en Gazebo para saber si funcionaba correctamente, en cuanto a la simulación se hizo la confirmación, ya que se pudo apreciar visualmente como el robot seguía

la trayectoria, posteriormente se procedió a la prueba física. Una vez hecho esto se ejecutó el programa con "roslaunch LiderSeguidor lider.launch"por medio de la computadora remota e inició la recabación de las posiciones con el comando ,"rosrecord tb3_0/amcl_pose tb3_1/amcl_pose", los datos que se recabaron son usados para calcular el error que existe entre los puntos por cuales fue definida la trayectoria y los puntos por los que el robot pasó, estos se muestran en las figuras 13, 14 y 15.

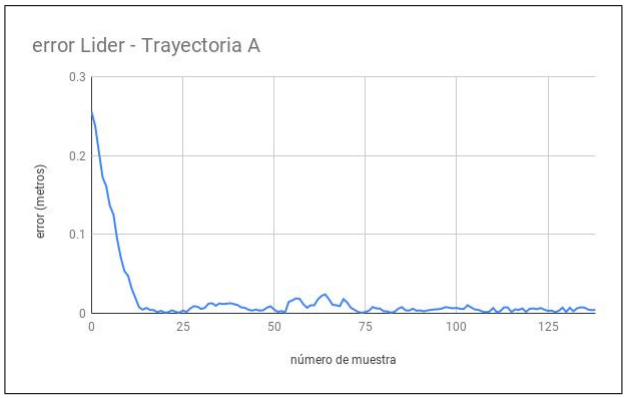

Figura 3.36: Error del líder en seguimiento de la trayectoria A.

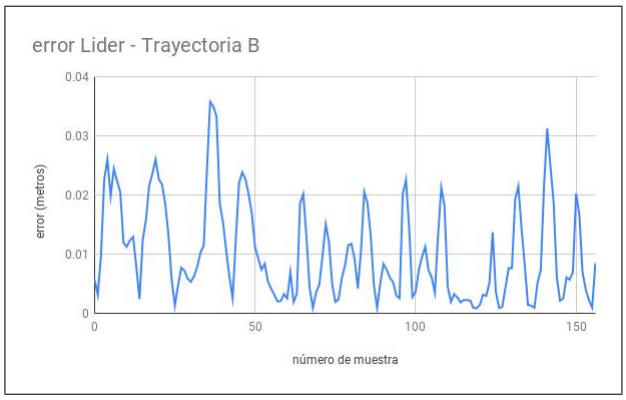

Figura 3.37: Error del líder en seguimiento de la trayectoria B.

.

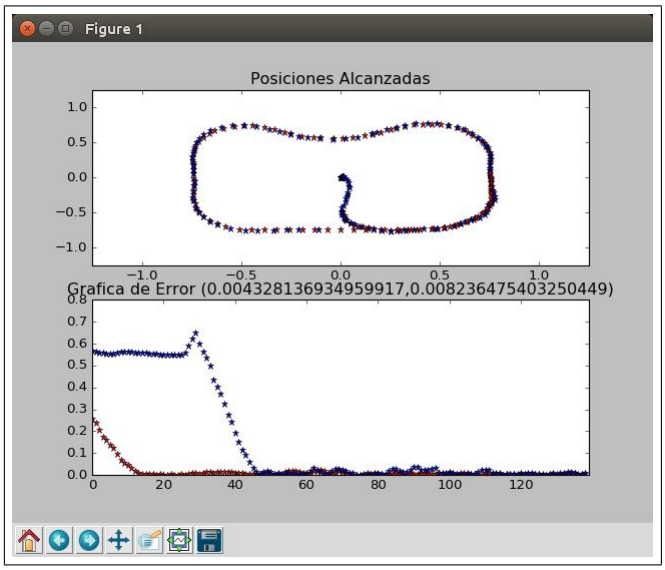

Figura 3.38: Gráfica de las posiciones alcanzadas y el error.

3.5.4.1. Análisis de resultados

Se puede observar el error disminuir y posteriormente mantenerse dentro de un umbral de 7 cm, lo cual indica que el seguimiento de trayectoria efectivamente es ejecutado y realmente se hace un seguimiento de trayectoria.

3.5.5. Prueba C: Validar el algoritmo Líder-Seguidor

Al realizar pruebas en la simulación del algoritmo líder-seguidor, se notaba como el robot seguidor en ocasiones realizaba los movimientos del líder con un desfase en la posición. Lo que se hizo para lograr un mejor seguimiento de los puntos de coordenada, fue aplicar una condición de distancia mínima (la cual se obtiene empíricamente) para poder cambiar la dirección del móvil una vez actualizada la posición del seguidor, esta condición se aplicó debido a que el algoritmo utilizado en el diseño detallado solo cambia su dirección con base a la actualización de la información, lo cual provoca que el seguidor solo cambie su orientación por el avance que ha tenido el líder, sin verse afectado por su posición actual, al asignar una condición de distancia mínima

entonces se logra que en el punto al que cambia su orientación el líder sea el mismo o muy parecido al del seguidor.

Figura 3.39: Error del seguidor en la trayectoria A.

Figura 3.40: Error del seguidor en la trayectoria B.

3.5.5.1. Análisis de resultados

Nuevamente se tomaron datos de posición en este caso del seguidor, obteniendo un error respecto a la trayectoria, dicho error, después de llegar a la trayectoria se mantiene en cierto umbral, validando el seguimiento al líder, el cual a su vez sigue la trayectoria.

3.5.6. Prueba D:Validar la evasión de obstáculos

El algoritmo de evasión de obstáculos fue la parte más compleja durante la implementación ya que, al reducir las dimensiones propuestas del espacio de trabajo, no lograba esquivar los objetos, el problema radicó en que todas las paredes eran leídas como obstáculos y no se lograba establecer un camino adecuado para la evasión. Para solucionar esta problemática, un nodo adicional fue hecho, este nodo se encargaba de calcular una distancia mínima para ciertos ángulos del robot, la función de este código mostrado en la Figura 3.41 es informarle al nodo principal si se debe leer o no el Histograma para posteriormente aplicar el algoritmo Vector Field Histogram (VFH). La forma de verificar el funcionamiento del código fue con apoyo del ambiente virtual hecho en Gazebo, el cual inicializaba el nodo y posteriormente corroboraba la información que enviaba el nodo con las posiciones de los obstáculos.

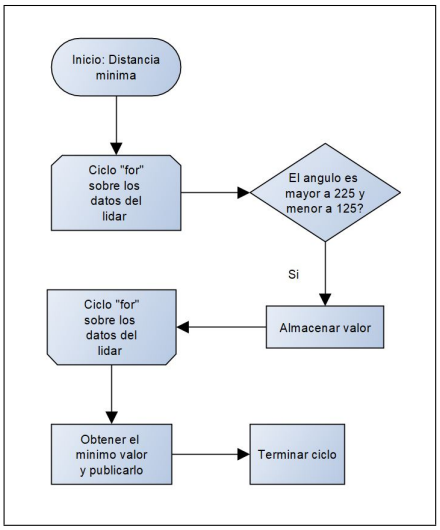

Figura 3.41: Nodo Laser_Distance.

En el programa donde se implementó el VFH fue necesario validarlo colocando obstáculos en el ambiente de simulación, y corroborando que el histograma efectivamente reflejara la posición angular de los obstáculos respecto al robot. Las figuras 3.42 y 3.43 muestran los resultados las pruebas.

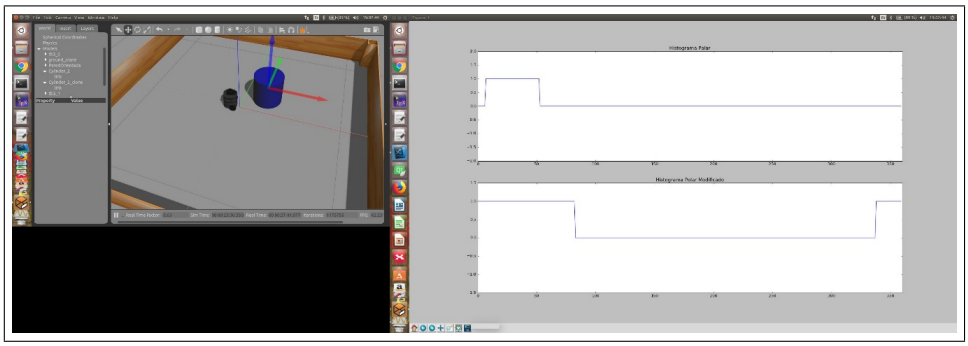

Figura 3.42: Detección del obstáculo cercano colocado en el primer cuadrante.

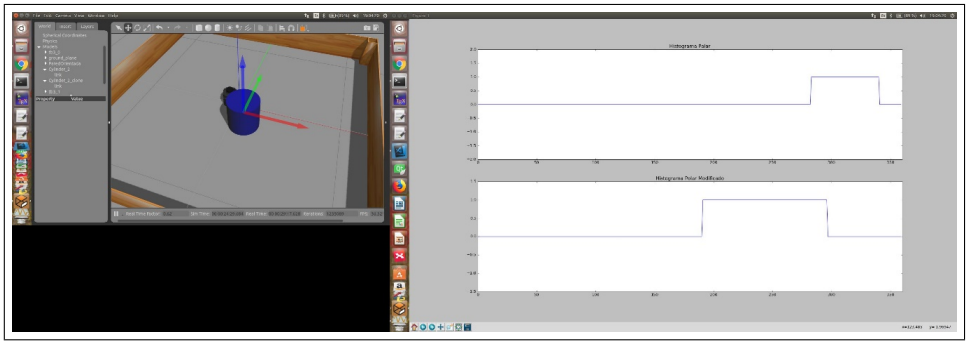

Figura 3.43: Detección del obstáculo cercano colocado en el segundo cuadrante.

Para integrar los nuevos nodos junto con el código del seguimiento de trayectoria fue necesario primeramente implementar la matriz de fuerza que permite al nodo saber la orientación que debe tomar el móvil para mantenerse dentro de la trayectoria propuesta, este algoritmo se programó en una función que llamamos Follow mostrado en la Figura 3.44, que sigue el siguiente algoritmo, obtiene su posición por el callback que genera el nodo amcl, una vez conocida la posición, se discretiza en un a matriz para poder ser accedida, dado que es una matriz discretizada para evitar un error de índices en la programación, se definen los límites máximos y mínimos, tomando en cuenta las dimensiónes del mapa. Con los índices definidos se procede a extraer la diferencia de la distancia entre el punto de origen del robot y el punto al que se desea llegar, en caso de que no se haya seleccionado una trayectoria el robot se detiene.

Para incluir la evasión de obstáculos se propuso tener una distancia mínima mostrado en la Figura 3.41, con la cual se inicia la evasión. En el caso en cual se tiene una distancia mayor a la propuesta, se ejecuta el algoritmo de la Figura 3.45, el cual tiene como propósito alinearse con el ángulo deseado y la orientación propia del robot, posteriormente este ángulo se compara con su complemento para determinar el sentido en que girar para tener movimientos más cortos, debido a que el sistema puede dar oscilaciones muy grandes y con esto dañar los motores, además se incluyen condiciones para que la velocidad angular quede dentro de un margen, así mismo si es muy grande el ángulo de error(45°) la velocidad lineal se vuelve cero. Estas velocidades son publicadas para que el robot tome estos valores, posteriormente se le da un retardo a la ejecución del código para tener tiempo suficiente de recopilar los datos.

En el caso en que la distancia es menor a la propuesta como se muestra en las Figuras 3.46 y 3.47 se verifica si no hay un obstáculo en la dirección por la que el robot de ir. De no haber obstáculo sobre el ángulo deseado se ejecuta la rutina anterior, en el caso de que si lo haya se hace una búsqueda para encontrar el primer camino libre en sentido antihorario, agregando 15° más para evitar colisionar, finalmente la nueva velocidad es publicada. Esto se ejecuta controlado por la interfaz que indica el avance o paro del robot.

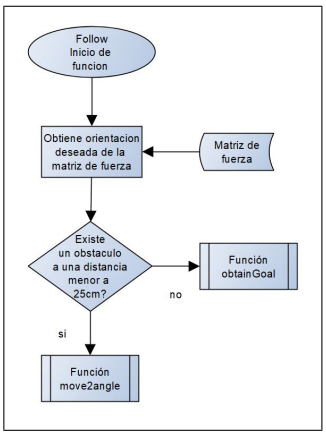

Figura 3.44: Función follow para decidir que ejecutar.

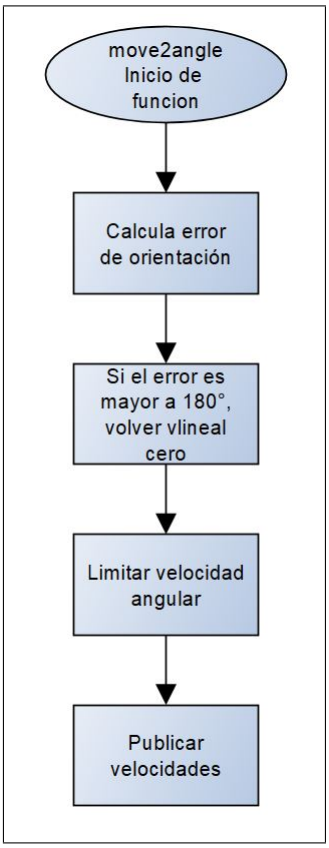

Figura 3.45: Función de seguimiento de trayectoria.

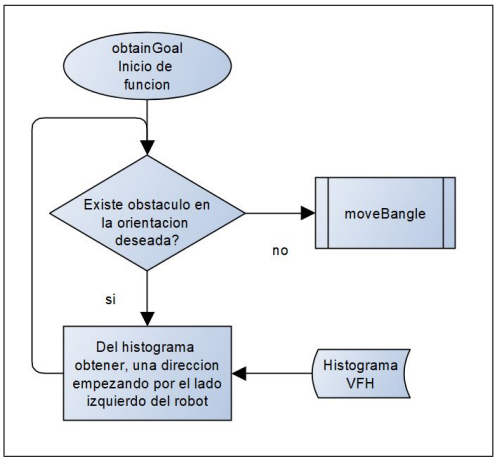

Figura 3.46: Algoritmo para aplicar VFH.

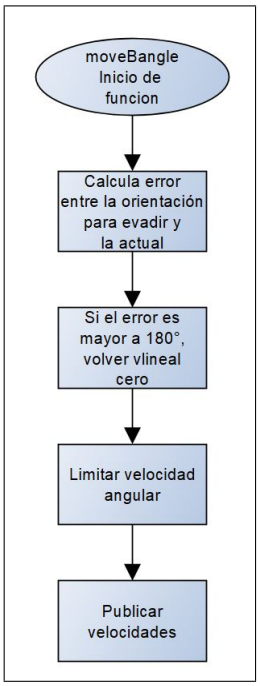

Figura 3.47: Función para ir a un sector libre.

3.5.7. Validación de seguimiento y evasión

La forma de validar los algoritmos de seguimiento de trayectorias y líder-seguidor consistió en una misma serie de pasos. Inicialmente fueron probados en un ambiente virtual, que incluye el espacio físico y un modelo funcional de los turtlebot3 burger, como si se tratase de los reales. De la simulación se verifican los programas, únicamente de forma visual, la prueba consiste en observar el comportamiento de los robots, y si estos siguen la trayectoria sin colisionar se confirma que fueron hechos correctamente. Posteriormente, los robots se prueban físicamente. Durante las pruebas físicas se guardan datos de posición, que junto con los datos de la trayectoria a seguir aplicando un algoritmo basado en un árbol kd se obtiene un error y con éste es finalmente validado, si existieron cambios son documentados.

Finalmente, para validar el algoritmo de evasión de obstáculos la prueba consiste en colocar dos obstáculos en la zona de pruebas y ejecutar los algoritmos de segui-

miento de trayectoria, líder-seguidor y evasión. Para cada trayectoria los robots dan 3 recorridos, si falla únicamente una sola vez se considera exitoso.

3.5.7.1. Análisis de resultados

Como se observo en las Figuras los algoritmos de evasión funcionan correctamente ya que el histograma muestra datos correctos, para los diferentes casos en que se presenta el obstáculo.

3.5.8. Prueba E: Validar la integración de evasión y seguimiento

El objetivo de esta prueba fue validar el funcionamiento del sistema en conjunto, líder-seguir y evasión. Se realizó una prueba en el ambiente físico, colocando ambos robots a medio metro de distancia y dos obstáculos en posiciones por las que se estaba seguro de que el líder intentaría pasar, debido al seguimiento de trayectoria, se realizaron 3 recorrido para cada trayectoria, si únicamente colisionaba una vez por trayectoria se consideró válido, adicionalmente mostramos las gráficas de error.

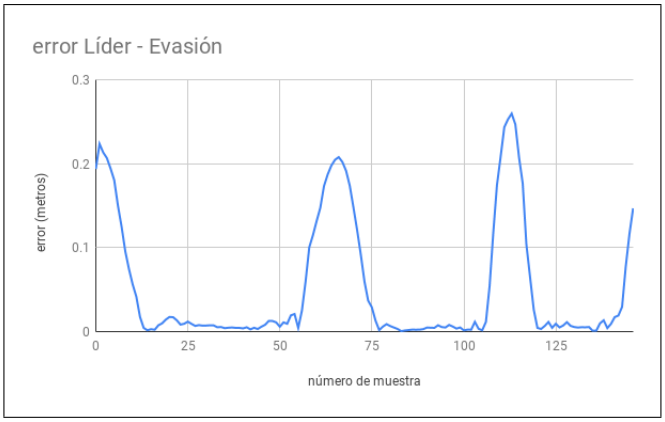

Figura 3.48: Error del Líder durante la evasión.

Figura 3.49: Error del Seguidor durante la evasión.

3.5.8.1. Análisis de resultados

Como se puede observar en las Figuras 3.48 y 3.49, el error disminuye cuando se mantiene dentro de la trayectoria y aumenta cuando requiere alejarse de la trayectoria e iniciar la evasión, para ambos robots el comportamiento es el mismo, lo cual valida que realmente existe una evasión ya que el líder toma como prioridad no chocar a mantenerse sobre la trayectoria, como el seguidor realiza las acciones del líder, el seguidor como era de esperar no colisionó.

3.5.9. Realización de la HMI

Para lograr que el sistema tenga una mejor interacción con el usuario se optó por intregrarle una hmi (human machine interface), para esto se buscaron varias posibilidades de frameworks de trabajo que fueran compatibles con ROS y que tuvieran una considerable información de consulta en páginas web.

Dadas estas características se utilizó el ambiente de trabajo QtCreator en el cual además de tener una extensa documentación e información de ayuda en internet, había sido utilizado para desarrollar una de las principales herramientas para el análisis de la información que posee el ecosistema de ROS, logrando estp por medio de la interfaz RQt realizada en Qt, en dicha interfaz se puden visualizar las conexiones

entre nodos, los valores de los distintos tópicos, mandar información para probar las variables de los tópicos de los nodos como la velocidad de los motores del turtlebot3 burger y graficar la información de los nodos a través del tiempo entre muchas otras funciones como las que en las Figuras 3.50 y 3.51.

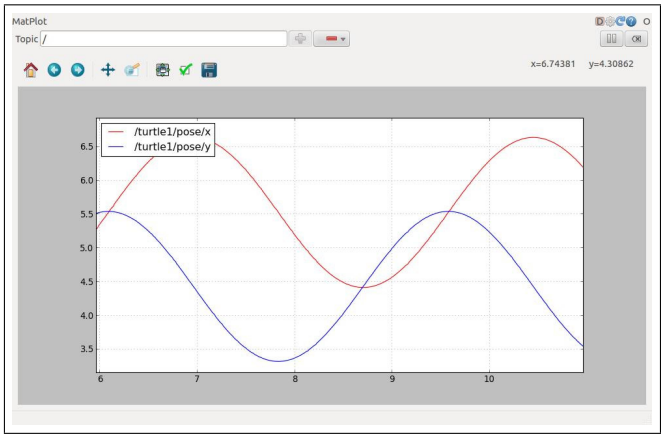

Figura 3.50: RQt para graficar distintos tópicos con respecto al tiempo.

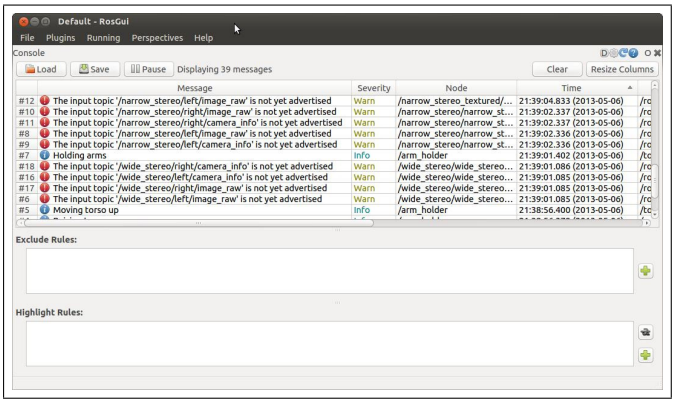

Figura 3.51: RQt mostrando y filtrando los distintos tipos de mensajes.

Con el objetivo de usar Qt como interfaz gráfica para mandar ordenes a los robots líder y seguidor se usaron los siguientes paquetes:

- qtcreator
- ros-kinetic-qt-create

- ros-kinetic-qt-build

Y para instalarlos se usaron las siguientes instrucciones:

- sudo apt-get install qtcreator
- sudo apt-get install ros-kinetic-qt-create
- sudo apt-get install ros-kinetic-qt-build

Una vez instalados se crearon los paquetes como se describen en la página web de la wiki de ros en la sección qt_build [34], asimismo cabe mencionar que para generar el paquete de ROS relacionado a la interfaz se usó la instrucción en la terminal en la carpeta catkin_ws de la siguiente forma:

- catkin_create_qt_pkg qgui

Donde "catkin_create_qt_pkg", es la instrucción para generar el paquete y "qgui", es el nombre del paquete asignado por el usuario.

Una vez compilado correctamente el paquete de la interfaz ahora se puede compilar como cualquier otro paquete usando la instrucción catkin_make y si esta instrucción al compilar lo hace correctamente ahora se puede editar y compilar el proyecto usando qtcreator. Para esto se ejecuta en la terminal el comando qtcreator y este comando abre el programa qtcreator como se muestra en la Figura 3.58.

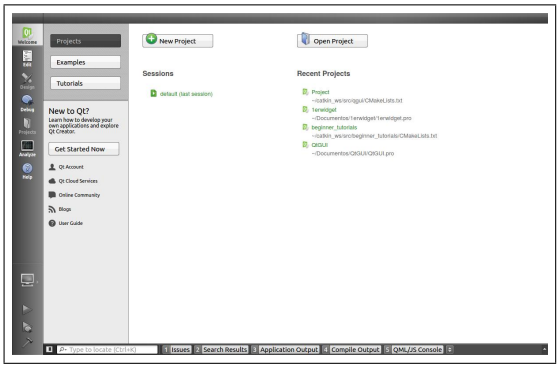

Figura 3.52: Pantalla de bienvenida de QtCreator.

Una vez dentro del programa se abre el proyecto dando clic en “File”luego “Open File or Project”, y se abre el archivo “CmakeList.txt”del paquete creado en este caso “qgui”, como se muestra en la Figura 3.53.

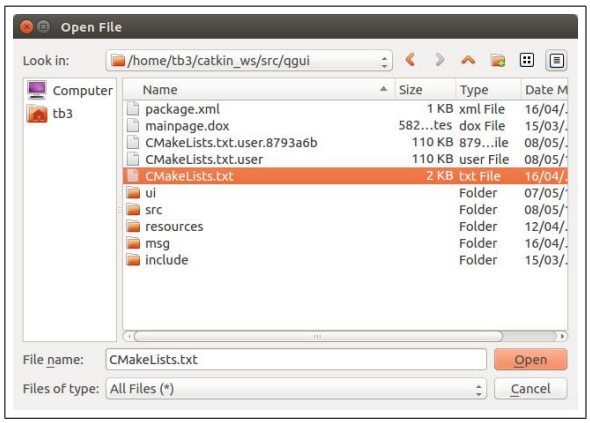

Figura 3.53: Ventana para compilar paquete en QtCreator.

Posteriormente se abre una ventana para elegir las opciones de compilación donde se selecciona la carpeta en la cual se guardará la compilación, aquí se selecciona la carpeta “build”dentro de la carpeta “catkin_ws”, como se muestra en la Figura 3.54.

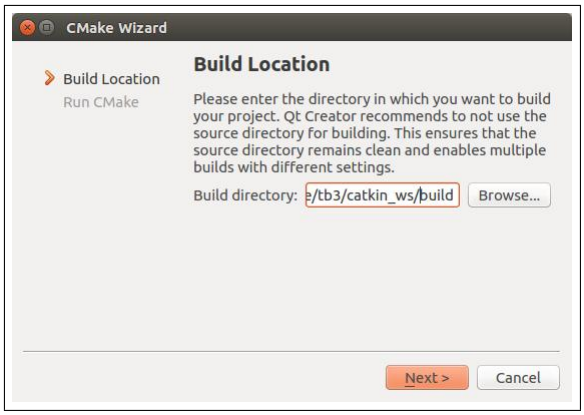

Figura 3.54: Ingreso de la carpeta de compilación.

Al darle “next”, aparecerá una ventana que pide el comando de compilación cmake, para que compile se le ingresa el siguiente comando en “Arguments”, el cual

le indica como y donde compilar:

- cmake ../src -DCMAKE_INSTALL_PREFIX=../install -DCATKIN_DEVEL_PREFIX=../devel

Este comando se puede consultar mas a detalle en la sección “catkin_make”de la wiki de ROS [33].

Luego se le da clic en el botón “Run CMake”, esta acción compila el proyecto y al darle “Finish”si se compiló correctamente se abren los archivos relacionados al proyecto para poderse editar finalmente como se muestra en la Figura 3.55.

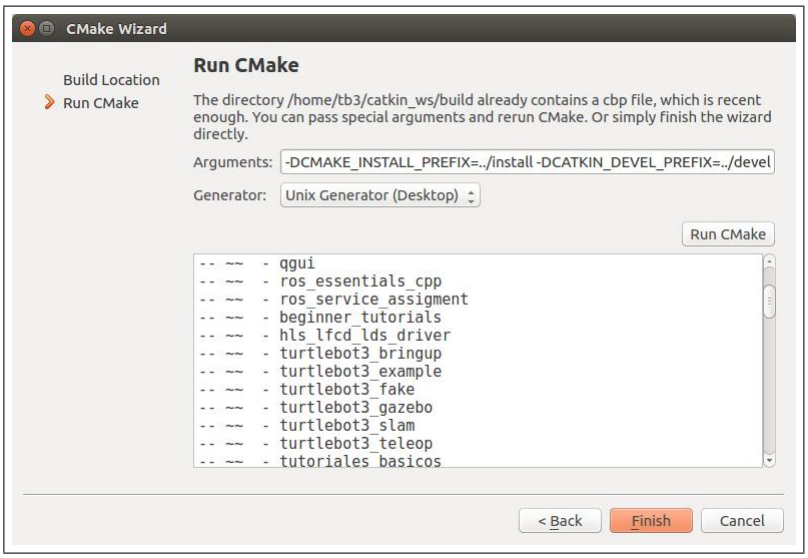

Figura 3.55: Ingreso y compilación con el comando CMake.

Una vez realizado esto exitosamente se muestra una ventana como la de la Figura 3.56, cabe mencionar que este procedimiento se puede aplicar para cualquier paquete del la carpeta “catkin_ws”.

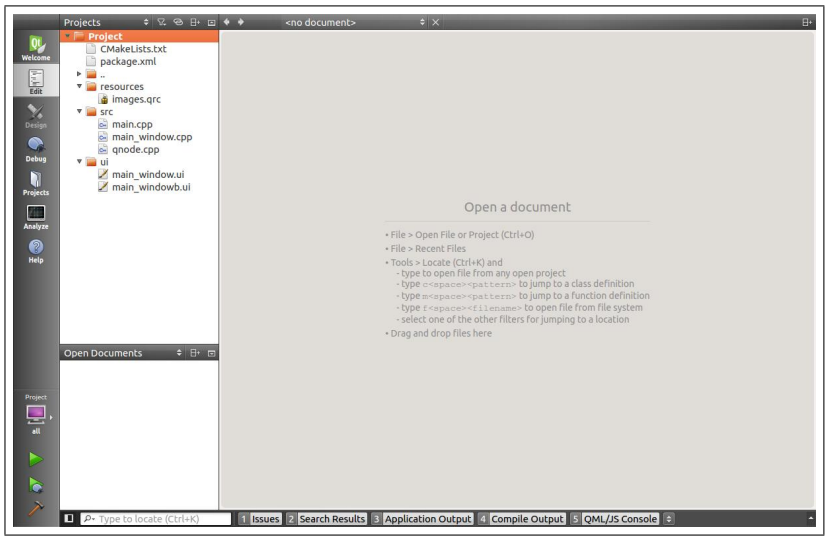

Figura 3.56: Ventana en Qt con el paquete qgui.

La edición del paquete se describe a detalle igualmente en la wiki de ROS en la sección de "qt_build"[34], pero a grandes rasgos este proyecto se divide en 4 partes:

- El programa principal del proyecto.
- El programa de la funcionalidad del código del la Ventana Gráfica.
- El programa de las funciones para interactuar con ROS.
- El framework para realizar la edición de la HMI gráficamente.

Estos 3 programas y el framework de edición trabajan en conjunto y cada uno está contenido dentro de otro de mayor nivel como se explica a continuación.

3.5.9.1. Programa Principal

El programa principal se encarga de crear el objeto de la ventana y que esta sea visible, además que cuando esta se cierra da la indicación que el programa ha terminado, esto al igual que los demás programas anteriormente mencionados este también usa hilos de ejecución por medio de la librería QThread, para que puedan ejecutarse

varios procesos a la vez dentro de un mismo código, logrando de tal forma que el usuario pueda interactuar con la interfaz al mismo tiempo que esta realiza distintos cálculos o enviá mensajes al ecosistema de ROS. Este programa en general contiene todo porque de aquí llaman a los demás programas anteriormente mencionados.

3.5.9.2. Ventana Gráfica

Este programa muestra la interfaz gráfica creada en el framework de edición de Qt y al ser un programa de c++ se compone de 2 archivos el "main_window.cpp", y el "main_window.hpp", en el primero se le informa que variables van a existir dentro de la clase y que métodos se van a definir y en el segundo se le da la funcionalidad a cada uno de los métodos, del tal manera en esta parte del programa se programan que acciones van a realizar los elementos colocados en la interfaz gráfica, por ejemplo cuando se selecciona una opción de la trayectoria en la interfaz creada para este proyecto se muestra una imagen de la trayectoria, implicando esto que al inicializarse esta ventana lee la imagen y luego la guarda para después mostrarla y en general este proceso de lectura, guardado y ejecución se implementa a lo largo de todo este programa. Cabe mencionar que este programa es afectado por lo que se edita en el framework o editor de la interfaz gráfica y este programa de la ventana gráfica está contenido dentro del Programa Principal.

3.5.9.3. Programa de ROS

Este programa al igual que el anterior se compone de 2 archivos el "qnode .cpp", y el "qnode.hpp", los cuales hacen una clase e igual dan funcionalidad y manejan definiciones respectivamente. En cuanto a lo que realiza este programa es inicializar la comunicación con ROS cuando le es pedido desde la interfaz gráfica, el cual al ser manejado por un hilo de ejecución no bloquea la interfaz gráfica, también aquí se crea el nodo para trabajar con ROS, se definen a que tópicos se suscribe este nodo, que tópicos publica y además se definen métodos públicos que pueden ser llamados

por la interfaz gráfica para realizar una cierta acción en el entorno de ROS. Cabe mencionar que este programa está contenido dentro del ventana gráfica y por defecto contiene el sistema de llamado de mensajes de consola de ROS.

3.5.9.4. Framework de Trabajo

El framework de trabajo es una herramienta muy útil ya que se puede editar fácilmente y de manera visual como se visualizará la interfaz, destacando 2 ventajas: la primera que queda visualmente como se bosqueja la interfaz gráfica en vez de modificar posiciones o contenedores para que se ajusten y la segunda que sin un gran conocimiento de todas las librerías de la interfaz gráfica se puedan usar los elementos provistos por el framework que además de todo mejoran los tiempos de desarrollo de la interfaz, ya que se auto genera el código de la visualización de la interfaz y solo queda por programar su funcionalidad. Cabe mencionar que este framework crea un archivo que está contenido en el programa de la ventana gráfica para que los elementos queden justo como en el editor de la interfaz el cual se muestra en la Figura 3.57.

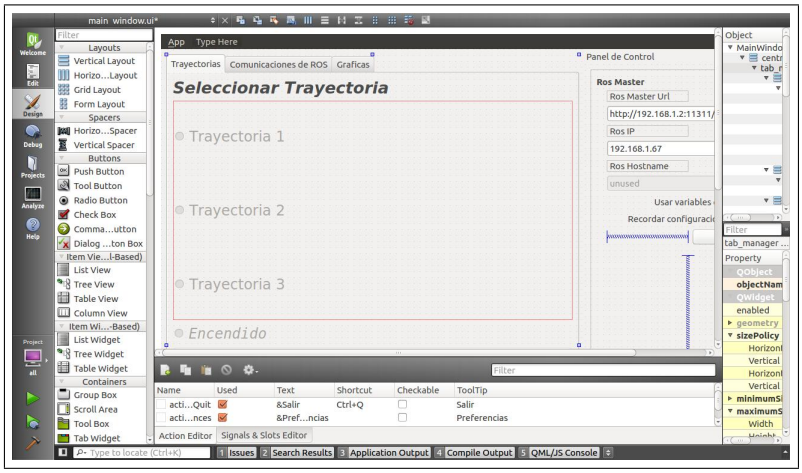

Figura 3.57: Framework de Qt para GUI.

3.5.9.5. Puntos importantes a considerar

Como recomendación se propone primero editar la interfaz gráfica sin intentar añadir un slot (método a ejecutar cada que hay un evento en cierto elemento de la interfaz; por ejemplo darle clic a un pushbutton) a los componentes ya que daña el proyecto, posteriormente compilarlo y después con este procedimiento ya aparecerán los nuevos componentes reconocidos automáticamente por el sistema para asi realizar una mejor edición del código. En el caso de usar slots para darle funcionalidad a cada componente se recomienda conectarlos manualmente como se muestra en la siguiente instrucción:

- QObject::connect(ui.pushButton_Graficar, SIGNAL(clicked()),
- this, SLOT(pushButton_Graficar_clicked()));

Donde "ui.pushButton_Graficar", es el botón en la interfaz gráfica, "clicked()", es la acción de darle clic a ese botón y "pushButton_Graficar_clicked()", es la acción que se ejecutará al darle clic en el botón "ui.pushButton_Graficar".

3.5.9.6. Modificación del Programa

Dado que este paquete ya venia con una funcionalidad predeterminada se módico el código base, así como la interfaz gráfica para ajustarlo a los requerimientos del trabajo, con lo cual siguiendo la lógica del programa descrita anteriormente, se consiguió enviar los tópicos necesarios para que se lograra interactuar con los robots así como con el graficador. Los tópicos que se crearon para dar dicha funcionalidad son los siguientes

- lane de tipo std_msgs::Int32 para seleccionar la trayectoria.
- turn_on de tipo std_msgs::Bool para decidir si los móviles avanzan.
- graph de tipo std_msgs::Bool para decidir si se grafica.

- clear_graph de tipo std_msgs::Bool para limpiar la gráfica y puntos almacenados.

- save_data de tipo std_msgs::Bool para guardar los datos graficados.

Es importante mencionar que para el correcto funcionamiento del programa primero se leyó y comentó lo que realizaba el código para tener una mejor comprensión del mismo y así poder editarlo correctamente sin presentar errores.

3.5.9.7. Resultados de la ejecución

A continuación se muestran los resultados de la ejecución de la interfaz, donde destacan 3 ventanas principales las cuales son:

- Ventana para elegir trayectoria y encender o apagar al robot.

- Ventana para visualizar el historial de la trayectoria elegida.

- Ventana para graficar, limpiar gráficas o guardar datos.

En la primera ventana como se muestra en las Figuras 3.58 y 3.59 se elige la trayectoria deseada por el usuario y con esta elección se podrá ver la imagen de la trayectoria a seguir, sin embargo esta se cargará hasta que se presione el botón cargar trayectoria, para que de esta manera se puedan inspeccionar libremente las trayectorias antes de cargarlas, cabe mencionar que la trayectoria predeterminadamente es la 0 en la cual nunca se avanza.

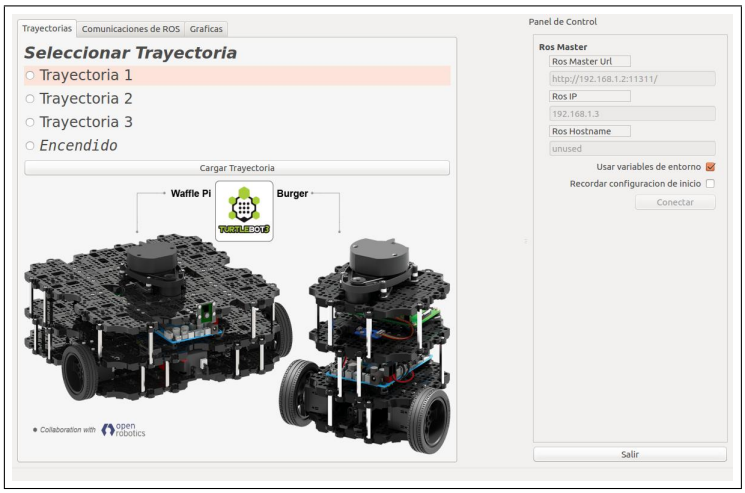

Figura 3.58: Ventana incial de la interfaz.

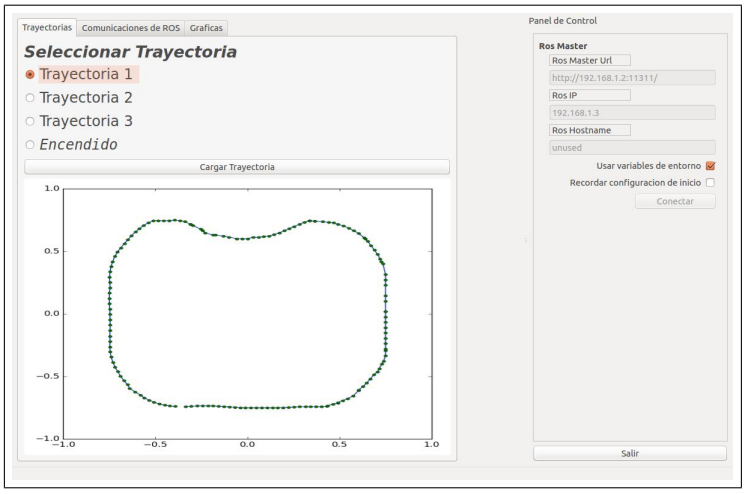

Figura 3.59: Configuración de la interfaz con la trayectoria 1 seleccionada y apagada.

Para hacer que el móvil avance se selecciona la opción encendido y en ese momento aparecerá la opción marcada y los móviles empezaran a seguir la trayectoria escogida como se muestra en las Figuras 3.60 y 3.61, para apagarlo basta con volver a presionar el botón.

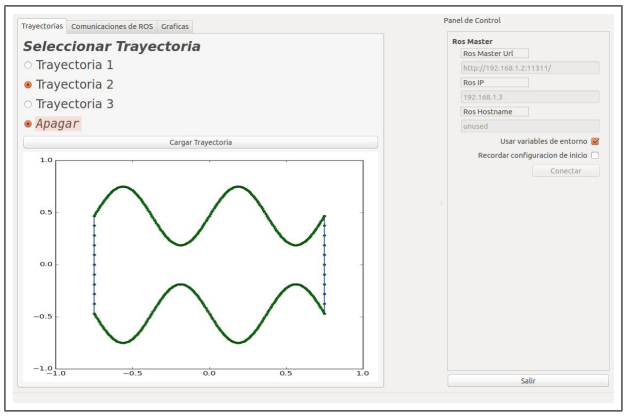

Figura 3.60: Configuración de la interfaz con la trayectoria 2 seleccionada y encendida.

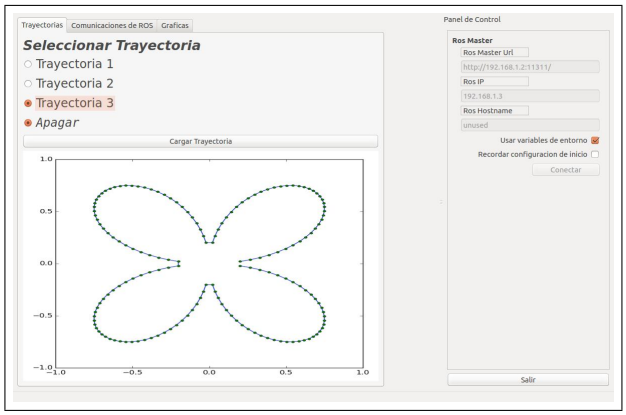

Figura 3.61: Configuración de la interfaz con la trayectoria 3 seleccionada y encendida.

En la segunda ventana se muestra el historial de todas las trayectorias cargadas, así como los mensajes de prueba para verificar que la interfaz este enviando bien la información a través de ROS, esto se muestra en la Figura 3.62 sin nunca haber cargado ninguna trayectoria y en la Figura 3.63 con un historial de trayectorias cargadas (donde además se puede consultar cual es la trayectoria a seguir actualmente) .

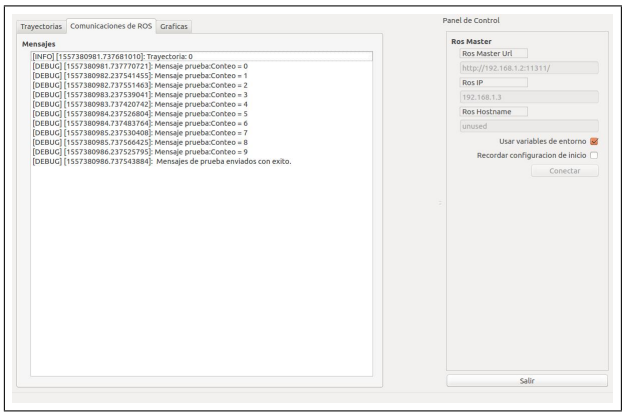

Figura 3.62: Interfaz con los mensajes de prueba enviados.

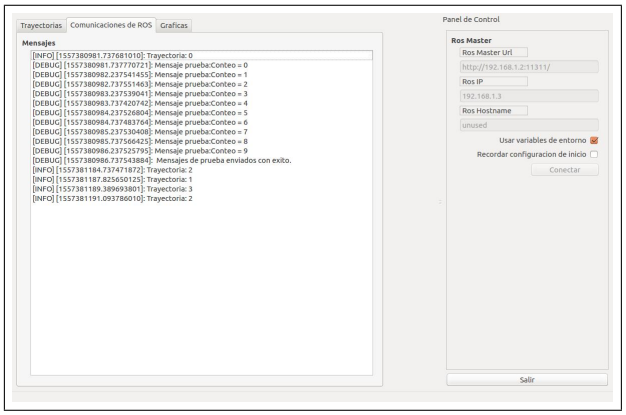

Figura 3.63: Interfaz con el historial de trayectorias seleccionadas.

En la tercera ventana como se observa en la Figura 3.64 se dan las opciones para controlar cuando se grafica o no, si se quiere limpiar la gráfica para realizar varios guardados sin repetir la información, y para guardar los datos de los valores de posición y error en un archivo con formato .npy y .xlsx .

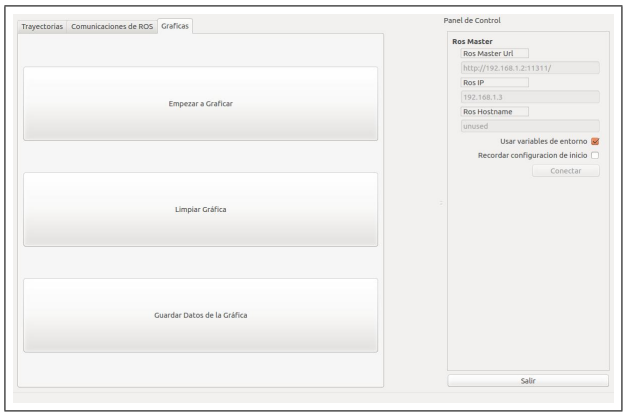

Figura 3.64: Interfaz mostrando las opciones de graficación.

3.6. Especificaciones del Sistema

En esta sección se habla de las especificaciones de algunos de los componentes de cada Área Funcional como los tiempos de las baterias LiPo, la velocidad de los motores, la resolución de los sensores (LIDAR y enconders), etcétera.

3.6.0.1. Especificaciones del Área de Trabajo

El Área de Trabajo fue fabricada de MDF y sus dimensiones son las siguientes: 2.4m x 2.4m x .30m (Longitud - Ancho - Altura).

3.6.0.2. Especificaciones de la Alimentanción

Cada bateria de los robots dura aproximadamente *dos horas* de uso continuo, y tarda en cargarse aproximadamente *una hora y media*.

3.6.0.3. Especificaciones del Procesamiento

Editar

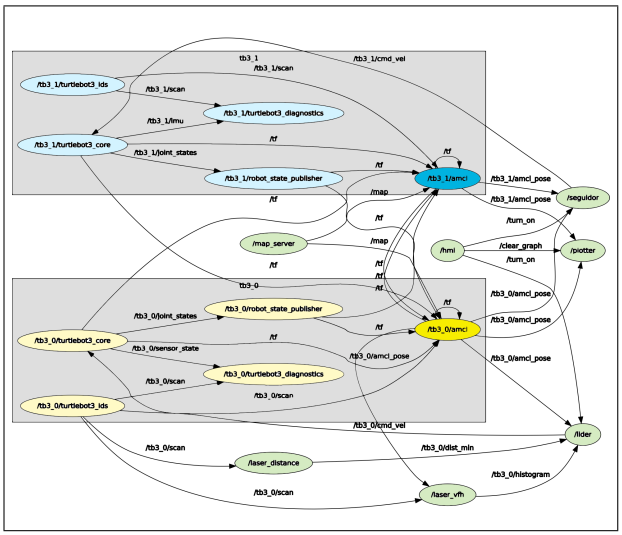

Figura 3.65: Nodos de Rosgraph.

Análisis de Costo

4.1. Costos variables

Los costos variables corresponden a los costos directos involucrados en la producción y venta de un articulo. Alineándonos a nuestro proyecto, no cubrimos gastos por ventas, pero si por producción, en donde, se derivan dos categorías: a) costos por materias primas y b) costos por mano de obra directa.

Los costos primarios corresponden a los costos por materia prima directa gastados. Ya que en nuestro trabajo no se manufacturó ni tampoco se transformó el material, nuestra materia prima corresponde a todo el material que se utilizó para llevar acabo el desarrollo del proyecto, desde la compra de los robots hasta la compra de los tornillos, pegamento, etc.

Articulo	Cantidad	Precio unitario	Precio total
Turtlebot3 burger	2	590 USD*	1180 USD*
Teclado y mouse Logitech	2	343.97 MXN	687.80 MXN
Cargado Energizer	1	206.03 MXN	206.03 MXN
Plan de remplazo	2	68.10 MXN	136.20 MXN
Adaptador de HDMI a VGA	2	188.70 MXN	188.70 MXN
Baterías	1	76.73 MXN	73.73 MXN
Placas de madera	3	170.00 MXN	510.00 MXN
Tira de pino	6	25.00 MXN	150.00 MXN
Otros (tornillos, pegamento, etc)	n	150.00 MXN	150.00 MXN
Total			24,808.75 MXN

Cuadro 4.1: Tabla de costos variables.

*La compra de los Turtlebot3 burger se hizo el 14 de septiembre de 2018, ese día el tipo de cambio promedio un total de 19.24 pesos mexicanos por cada dólar estadounidense.

4.2. Costos fijos

Horas trabajadas

TT1	Total de días trabajados	Horas trabajadas por día
	85	3
Total de horas trabajadas		255

Cuadro 4.2: Días trabajados TT1.

TT2	Total de días trabajados	Horas trabajadas por día
	95	4
Total de horas trabajadas		380

Cuadro 4.3: Días trabajados TT2.

Total de días trabajados	180 días
Total de horas trabajadas	635 horas

Cuadro 4.4: Total de días y horas trabajadas.

Nota: El total de días trabajados es ideal, ya que únicamente se están contabilizando los 5 primeros días de la semana, sin contar fines de semana, ni vacaciones, aunque se haya trabajado en fines de semana y vacaciones. Este análisis se hizo basándose en el calendario del Instituto Politécnico Nacional (semestres 2019 -1 y 2019 -2).

Los costos fijos son costos periódicos, es decir, suelen incurrirse a ellos por medio del tiempo transcurrido, como lo sería costos por renta, mantenimiento, etc., pero como fue en nuestro caso nosotros no cubrimos ninguno de esos gastos, porque son gastos cubiertos por el Instituto Politécnico Nacional. Consideramos como gastos fijos los gastos de transporte, comidas, y tiempo en la escuela por todos los miembros del equipo.

Tipo de gasto	Gasto por día [MXN]	Gasto por el total de días trabajados (180)
Pasajes	$140.00	$25,200.00
Comidas	$160.00	$28,800.00
Otros gastos	$15.00	$2,700.00
TOTAL		$56,700.00 MXN

Cuadro 4.5: Costo total de gastos fijos.

4.3. Costo total

El costo total de proyecto corresponde a la suma del costo fijo más la suma del costo variable.

$$\textit{Costo total} = \textit{Costo fijo} + \textit{Costo variable}$$

$$\text{Costo total del proyecto} = \$81{,}508.75.00$$

Este es el *precio neto* del proyecto por lo que si se pensará en comercializarlo se le podría agregar un porcentaje de ganancia con base en el costo total de proyecto.

Conclusiones

A lo largo del desarrollo del proyecto todos los diseños, códigos y pruebas dieron como resultado un sistema de dos robots móviles que desarrollan la tarea del seguimiento de trayectorias y evasión de obstáculos bajo un esquema líder-seguidor. Al tener el sistema funcionando correctamente se pudieron dar por cumplidos los objetivos particulares los cuales iban orientados a la adecuada selección e implementación de los subsistemas que componen el proyecto. Se logró seleccionar y tener funcionando correctamente todos los sensores para la correcta recabación de datos, así mismo satisfactoriamente diversos algoritmos fueron implementados para el correcto comportamiento de los robots, todo bajo el control de una HMI, cuyas funciones son la de seleccionar trayectorias y desplegar información propia del sistema.

Respecto a la implementación del proyecto muchos fueron los retos técnicos que se requirieron superar, modificaciones en el VFH para ser adaptado al sensor LIDAR, y la adición de un factor que aumentaba la dimensión de los obstáculos para evitar los choques debido a las dimensiones de los robots. La utilización de hilos de ejecución en la programación fue fundamental para el proyecto ya que permitió realizar distintos procesos de cómputo lo que aumento la velocidad de respuesta del sistema y dio paso a tener movimientos fluidos en ambos robots.

Al realizar un sistema de robots múltiples se dejó como contribución y al alcance

de la comunidad politécnica dos algoritmos totalmente funciones para ser usados en diversas aplicaciones relacionadas con la robótica cooperativa. Entre ellos está el VFH implementado en ROS y Python, haciendo uso de únicamente un sensor LIDAR, lo que permite ser aplicado con facilidad a otros robots o sistemas y el algoritmo lider-seguidor él cual puede ser aplicado para mas de dos robots.

Lo resultados de este proyecto dejan en la Unidad Profesional las bases para poder desarrollar aplicaciones prácticas, entre las cuales está el transponer del material con robots móviles autónomos, un sistema eficiente de flujo de tráfico aplicando el principio del lider-seguidor y la evasión de obstáculos con drones por medio de sensores LIDAR y el algoritmo de evasión VFH.

A continuación detallamos mejor las razones por las cuales consideramos exitoso el cumplimiento de cada uno de los objetivos particulares.

Seleccionar y construir dos sistemas de locomoción para los móviles que les permita moverse en superficies y ambientes controlados. Este objetivo se cumplió al analizar las distintas opciones que se tenían disponibles de acuerdo con el presupuesto y haber seleccionado una de estas, la segunda parte del objetivo se cumplió al construirlos con indicaban los manuales. Los detalles de implementación, se encuentra en la sección de Integración del área funcional de estructura.

Diseñar e implementar un control de seguimiento de trayectoria para el robot líder. Este objetivo se cumplió al diseñar un algoritmo de seguimiento el cual hizo uso de la matriz de fuerza, dicho elemento cumple con la función de orientar de forma correcta al líder, para así mantenerse sobre la trayectoria. La implementación fue hecha con el lenguaje de programación Python, ROS. Los detalles de implementación se mencionan en el área funcional de procesamiento, específicamente las pruebas A y B.

Diseñar e implementar un control de seguimiento al líder para el robot seguidor. Este objetivo se cumplió al diseñar un algoritmo de seguimiento al líder el cual hizo uso de la lista de puntos, que cumple con la función de orientar de forma correcta al líder, para así mantenerse sobre la trayectoria. La implementación fue

hecha con el lenguaje de programación Python, ROS. Los detalles de implementación se mencionan en el área funcional de procesamiento, específicamente la prueba C.

Establecer e implementar un método de comunicación apropiado para la configuración líder seguidor. El método de comunicación fue por medio de ROS que permite la transmisión de mensajes personalizados y entre distintas computadoras apoyado del protocolo de comunicación WIFI. Los detalles de implementación se mencionan en el área funcional de procesamiento, específicamente la parte de comunicación.

Diseñar e implementar un algoritmo para la evasión de obstáculos fijos. El algoritmo fue diseñado a partir del VFH, del cual tuvo que ser diseñado ajustado a los sensores y lenguajes de programación usados, que a su vez fueron parte de la implementación. Los detalles de implementación se mencionan en el área funcional de procesamiento, específicamente la prueba D.

Seleccionar e implementar un sistema de sensado en el robot líder, para su propia localización y detección de obstáculos. El sistema de sensado fue seleccionada juntos los robots móviles, la implementación consistió en lograr correctas mediciones para detectar obstáculos de forma correcta y lograr su localización donde la mayor parte del trabajo consistió en ajustes por medio de software. Los detalles de implementación se mencionan en el área funcional de percepción, y la parte concerniente a software en prueba D de procesamiento.

Seleccionar e implementar un sistema de sensado en el robot seguidor, para la identificación del líder. El sistema de sensado fue seleccionada juntos los robots móviles, la implementación consistió en lograr correctas mediciones para localizar al líder, esta sección también fue en su mayoría ajustes por medio de software. Los detalles de implementación se mencionan en el área funcional de percepción.

Diseñar e implementar una GUI para el monitoreo del sistema, la selección de una trayectoria a seguir y delimitar el área de trabajo. El diseño de la GUI fue presentado durante la sección de diseño detallado y posteriormente modificada durante la implementación para mejorar el aspecto visual y concordar con

las funciones el objetivo menciona. Los detalles de implementación se profundizan en la seccione titulada realización de la HMI.

Referencias

[1] Alvaro, G. (2013). Ros: Robot operating system. *Universidad Politécnica de Cartagena*.

[2] AutoModelCar (2016). *Robotis e-Manual*. http://emanual.robotis.com/docs/en/platform/turtlebot specifications.

[3] Berlin, F. U. (2018). *Automodel Car*. Dahlem Center for Machine Learning and Robotics, Freie Universität Berlin. https://github.com/AutoModelCar/AutoModelCarWiki/wiki/Navigation.

[4] Bishop, R. H. (2008). *The mechatronics handbook*. London.

[5] Borenstein, J. and Koren, Y. (1991). The vector field histogram-fast obstacle avoidance for mobile robots. *IEEE Transactions on Robotics and Automation*, 7(3):278 to 288.

[6] Bugarin Carlos, E. and Aguilar Bustos, A. Y. (2014). Control visual para la formación de robots móviles tipo uniciclo bajo el esquema líder seguidor. *UNAM*.

[7] Darpa (2007). Route network definition file (rndf) and mis-

sion data file (mdf) formats. *Grandchallenge*, 1:4 to 9. https://www.grandchallenge.org/grandchallenge/docs/RNDF_MDF_Formats_031407.pdf.

[8] García Tovar, V. E., Guzmán Martínez, S., Pérez Nicolás, P., and Terán Chapul, R. E. (2017). Robot repartidor autonónomo.

[9] Gérard Bruno, o. H. (2005). *El Proceso de Análisis Jerárquico (AHP) como Herramienta para la Toma de Decisiones en la Selección de Proveedores.* PhD thesis, Univesidad Mayor de San Marcos. http://sisbib.unmsm.edu.pe/bibvirtualdata/Tesis/Basic/toskano_hg/cap3.PDF.

[10] Gerkey, B. P., Lu, D. V., Ferguson, M., and Hoy, A. (2018). Amcl. http://wiki.ros.org/amcl.

[11] GTK (2018). *Batería de polímero de litio.* Shenzhen Foxell Technologies Co., Ltd. https://es.aliexpress.com/item/1-unids-11-1-V-5C-1800-mAh-li-po-bater-a-de-pol-mero-de/32830921404.html.

[12] Harashima, F., Tomizuka, M., and Fukuda, T. (1996). *Mechatronics, What is it, why and how?* IEEE/ASME Transactions on Mechatronics.

[13] Jet (2017). Tx1 jet robot kit. https://www.servocity.com/tx1-jet-robot-kit.

[14] Juárez Palafox, J., Muro Maldonado, D., and Franco González, A. (2005). Construcción y control de un robot móvil.

[15] Juliá Cristóbal, M. (2005). Seguimiento de robots mediante visión artificial. aplicación al mantenimiento de formaciones basada en comportamientos.

[16] Kumar, P. (2016). *Install Ubuntu.* https://www.linuxtechi.com/install-ubuntu-16-04-with-screenshots/.

[17] Lagunes Fortiz, M. A. (2014). Sistema de navegación evasor de obstaáculos en un robot móvil.

[18] Madhevan, B. and Sreekumar, M. (2013). Tracking algorithm using leader follower approach for multi robots. *Procedia Engineering*, 64:1426 to 1435.

[19] Molina Villa, M. A. and Rodríguez Vásquez, E. L. (2014). Flotilla de robots para trabajos en robotica cooperativa.

[20] Petrinic, T. and Petrovic, I. (2013). A leader-follower approach to formation control of multiple non-holonomic mobile robots. In *2013 36th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO)*, page 931 to 935.

[21] Rangel, Cortina, A., Park, J. Y., and Lloyd (2016). Turtlebot 3 burger [us].

[22] Raspberrypi.org (2013). *SSH (Secure Shell)*. https://www.raspberrypi.org/documentation/remote-access/ssh/.

[23] Ribeiro, M. I. (2005). Obstacle avoidance. http://users.isr.ist.utl.pt/ mir/pub/ObstacleAvoidance.pdf.

[24] Riveros Guevara, A. and Solaque Guzmán, L. E. (2013). Formación de robots móviles mediante el uso de controladores. *USBMed*, page 63 to 64.

[25] Ro-Botica (2016). *DYNAMIXEL XL430-W250*. Accensit. http://emanual.robotis.com/docs/en/dxl/x/xl430-w250/control-table-of-eeprom-area.

[26] Ro-Botica (2018). *Motores DYNAMIXEL*. Accensit. http://robotica.com/tienda/ROBOTIS-DYNAMIXEL.

[27] Robotis (2016a). *e-Manual Features*. http://emanual.robotis.com/docs/en/platform/turtlebot3/fe

[28] Robotis (2016b). *e-Manual specifications*. http://emanual.robotis.com/docs/en/platform/turtlebot3/specifications/.

[29] Robotis (2016c). *install-ubuntu-on-remote-pc*. Jekyll and Minimal Mistakes. http://emanual.robotis.com/docs/en/platform/turtlebot3/pcsetup/install-ubuntu-on-remote-pc.

[30] Robotis (2016d). Raspberry pi 3 model b. https://www.raspberrypi.org/products/raspberry-pi-3-model-b/.

[31] Robotis (2018a). *LIPO Battery 11.1V*. Purdue University. http://www.robotis.us/lipo-battery-11-1v-1800mah-lb-012/.

[32] Robotis (2018b). Opencr. http://emanual.robotis.com/docs/en/parts/controller/opencr10/.

[33] ROS (2014a). catkin make. http://wiki.ros.org/catkin/commands/catkin_make.

[34] ROS (2014b). qt-build. http://wiki.ros.org/qt_ros.

[35] RullyFoote (2019). Topics. http://wiki.ros.org/topics.

[36] Siegwart, R., Nourbakhsh, I. R., and Scaramuzza, D. (2011). *Introduction to autonomous mobile robots*. MIT.

[37] Store, H. (2017). Rosbot 2.0. https://store.husarion.com/collections/dev-kits/products/rosbot?variant=8799150276662.

[38] Thrun, S., Burgard, W., and Fox, D. (2010). *Probabilistic robotics*. MIT Press.

[39] Ulrich, I. and Borenstein, J. (1998). Vfh : reliable obstacle avoidance for fast mobile robots. *Proceedings. 1998 IEEE International Conference on Robotics and Automation (Cat. No.98CH36146)*, page 1572 to 1577. https://ieeexplore.ieee.org/document/677362.

[40] Vidal, R., Shakernia, O., and Sastry, S. (2003). Formation control of nonholonomic mobile robots with omnidirectional visual servoing and motion segmentation. *ieeexplore*.

[41] Xiaohai, L., Jizong, X., and Zijun, C. (2005). Backstepping based multiple mobile robots formation control. *ieeexplore*.

[42] YoonSeok, P., HanCheol, C., RyuWoon, J., and TaeHoon, L. (2017). *ROS Robot Programming*. ROBOTIS Co,Ltd.

Apéndices

Apéndice 1

3. Nodes

3.1 amcl

`amcl` takes in a laser-based map, laser scans, and transform messages, and outputs pose estimates. On startup, `amcl` initializes its particle filter according to the parameters provided. Note that, because of the defaults, if no parameters are set, the initial filter state will be a moderately sized particle cloud centered about (0,0,0).

3.1.1 Subscribed Topics

`scan` (sensor_msgs/LaserScan)
Laser scans.

`tf` (tf/tfMessage)
Transforms.

`initialpose` (geometry_msgs/PoseWithCovarianceStamped)
Mean and covariance with which to (re-)initialize the particle filter.

`map` (nav_msgs/OccupancyGrid)
When the `use_map_topic` parameter is set, AMCL subscribes to this topic to retrieve the map used for laser-based localization. **New in navigation 1.4.2.**

3.1.2 Published Topics

`amcl_pose` (geometry_msgs/PoseWithCovarianceStamped)
Robot's estimated pose in the map, with covariance.

`particlecloud` (geometry_msgs/PoseArray)
The set of pose estimates being maintained by the filter.

`tf` (tf/tfMessage)
Publishes the transform from `odom` (which can be remapped via the ~odom_frame_id parameter) to `map`.

3.1.3 Services

`global_localization` (std_srvs/Empty)
Initiate global localization, wherein all particles are dispersed randomly through the free space in the map.

`request_nomotion_update` (std_srvs/Empty)
Service to manually perform update and publish updated particles.

`set_map` (nav_msgs/SetMap)
Service to manually set a new map and pose.

Figura 1: Configuración de nodos en ROS parte A

3.1.5 Parameters

There are three categories of ROS Parameters that can be used to configure the `amcl` node: overall filter, laser model, and odometery model.

Overall filter parameters

`~min_particles` (`int`, default: 100)

Minimum allowed number of particles.

`~max_particles` (`int`, default: 5000)
Maximum allowed number of particles.

`~kld_err` (`double`, default: 0.01)
Maximum error between the true distribution and the estimated distribution.

`~kld_z` (`double`, default: 0.99)
Upper standard normal quantile for (1 - p), where p is the probability that the error on the estimated distrubition will be less than `kld_err`.

`~update_min_d` (`double`, default: 0.2 meters)
Translational movement required before performing a filter update.

`~update_min_a` (`double`, default: π/6.0 radians)
Rotational movement required before performing a filter update.

`~resample_interval` (`int`, default: 2)
Number of filter updates required before resampling.

`~transform_tolerance` (`double`, default: 0.1 seconds)
Time with which to post-date the transform that is published, to indicate that this transform is valid into the future.

`~recovery_alpha_slow` (`double`, default: 0.0 (`disabled`))
Exponential decay rate for the slow average weight filter, used in deciding when to recover by adding random poses. A good value might be 0.001.

`~recovery_alpha_fast` (`double`, default: 0.0 (`disabled`))
Exponential decay rate for the fast average weight filter, used in deciding when to recover by adding random poses. A good value might be 0.1.

`~initial_pose_x` (`double`, default: 0.0 meters)
Initial pose mean (x), used to initialize filter with Gaussian distribution.

`~initial_pose_y` (`double`, default: 0.0 meters)
Initial pose mean (y), used to initialize filter with Gaussian distribution.

Figura 2: Configuración de nodos en ROS parte B

~initial_pose_a (double, default: 0.0 radians)
Initial pose mean (yaw), used to initialize filter with Gaussian distribution.

~initial_cov_xx (double, default: 0.5*0.5 meters)
Initial pose covariance (x*x), used to initialize filter with Gaussian distribution.

~initial_cov_yy (double, default: 0.5*0.5 meters)
Initial pose covariance (y*y), used to initialize filter with Gaussian distribution.

~initial_cov_aa (double, default: (π/12)*(π/12) radian)
Initial pose covariance (yaw*yaw), used to initialize filter with Gaussian distribution.

~gui_publish_rate (double, default: -1.0 Hz)
Maximum rate (Hz) at which scans and paths are published for visualization, -1.0 to disable.

~save_pose_rate (double, default: 0.5 Hz)
Maximum rate (Hz) at which to store the last estimated pose and covariance to the parameter server, in the variables ~initial_pose_* and ~initial_cov_*. This saved pose will be used on subsequent runs to initialize the filter. -1.0 to disable.

~use_map_topic (bool, default: false)
When set to true, AMCL will subscribe to the map topic rather than making a service call to receive its map. **New in navigation 1.4.2**

~first_map_only (bool, default: false)
When set to true, AMCL will only use the first map it subscribes to, rather than updating each time a new one is received. **New in navigation 1.4.2**

Laser model parameters

Note that whichever mixture weights are in use should sum to 1. The beam model uses all 4: z_hit, z_short, z_max, and z_rand. The likelihood_field model uses only 2: z_hit and z_rand.

~laser_min_range (double, default: -1.0)

Minimum scan range to be considered; -1.0 will cause the laser's reported minimum range to be used.

~laser_max_range (double, default: -1.0)
Maximum scan range to be considered; -1.0 will cause the laser's reported maximum range to be used.

~laser_max_beams (int, default: 30)
How many evenly-spaced beams in each scan to be used when updating the filter.

~laser_z_hit (double, default: 0.95)
Mixture weight for the z_hit part of the model.

~laser_z_short (double, default: 0.1)
Mixture weight for the z_short part of the model.

Figura 3: Configuración de nodos en ROS parte C

~laser_z_max (double, default: 0.05)
Mixture weight for the z_max part of the model.

~laser_z_rand (double, default: 0.05)
Mixture weight for the z_rand part of the model.

~laser_sigma_hit (double, default: 0.2 meters)
Standard deviation for Gaussian model used in z_hit part of the model.

~laser_lambda_short (double, default: 0.1)
Exponential decay parameter for z_short part of model.

~laser_likelihood_max_dist (double, default: 2.0 meters)
Maximum distance to do obstacle inflation on map, for use in likelihood_field model.

~laser_model_type (string, default: "likelihood_field")
Which model to use, either beam, likelihood_field, or likelihood_field_prob (same as likelihood_field but incorporates the beamskip feature, if enabled).

Odometry model parameters

If ~odom_model_type is "diff" then we use the sample_motion_model_odometry algorithm from Probabilistic Robotics, p136; this model uses the noise parameters odom_alpha_1 through odom_alpha4, as defined in the book.

If ~odom_model_type is "omni" then we use a custom model for an omni-directional base, which uses odom_alpha_1 through odom_alpha_5. The meaning of the first four parameters is similar to that for the "diff" model. The fifth parameter capture the tendency of the robot to translate (without rotating) perpendicular to the observed direction of travel.

A bug was found and fixed. But fixing the old models would have changed or broken the localisation of already tuned robot systems, so the new fixed odometry models were added as new types "diff-corrected" and "omni-corrected". The default settings of the odom_alpha parameters only fit the old models, for the new model these values probably need to be a lot smaller, see http://answers.ros.org/question/227811/tuning-amcls-diff-corrected-and-omni-corrected-odom-models/.

~odom_model_type (string, default: "diff")

Which model to use, either "diff", "omni", "diff-corrected" or "omni-corrected".

~odom_alpha1 (double, default: 0.2)
Specifies the expected noise in odometry's rotation estimate from the rotational component of the robot's motion.

~odom_alpha2 (double, default: 0.2)
Specifies the expected noise in odometry's rotation estimate from translational component of the robot's motion.

~odom_alpha3 (double, default: 0.2)
Specifies the expected noise in odometry's translation estimate from the translational component of the robot's motion.

~odom_alpha4 (double, default: 0.2)
Specifies the expected noise in odometry's translation estimate from the rotational component of the robot's motion.

Figura 4: Configuración de nodos en ROS parte D

~laser_model_type (string, default: "likelihood_field")
Which model to use, either beam, likelihood_field, or likelihood_field_prob (same as likelihood_field but incorporates the beamskip feature, if enabled).

Odometry model parameters

If ~odom_model_type is "diff" then we use the sample_motion_model_odometry algorithm from Probabilistic Robotics, p136; this model uses the noise parameters odom_alpha_1 through odom_alpha4, as defined in the book.

If ~odom_model_type is "omni" then we use a custom model for an omni-directional base, which uses odom_alpha_1 through odom_alpha_5. The meaning of the first four parameters is similar to that for the "diff" model. The fifth parameter capture the tendency of the robot to translate (without rotating) perpendicular to the observed direction of travel.

A bug was found and fixed. But fixing the old models would have changed or broken the localisation of already tuned robot systems, so the new fixed odometry models were added as new types "diff-corrected" and "omni-corrected". The default settings of the odom_alpha parameters only fit the old models, for the new model these values probably need to be a lot smaller, see http://answers.ros.org/question/227811/tuning-amcls-diff-corrected-and-omni-corrected-odom-models/.

~odom_model_type (string, default: "diff")

Which model to use, either "diff", "omni", "diff-corrected" or "omni-corrected".

~odom_alpha1 (double, default: 0.2)
Specifies the expected noise in odometry's rotation estimate from the rotational component of the robot's motion.

~odom_alpha2 (double, default: 0.2)
Specifies the expected noise in odometry's rotation estimate from translational component of the robot's motion.

~odom_alpha3 (double, default: 0.2)
Specifies the expected noise in odometry's translation estimate from the translational component of the robot's motion.

~odom_alpha4 (double, default: 0.2)
Specifies the expected noise in odometry's translation estimate from the rotational component of the robot's motion.

~odom_alpha5 (double, default: 0.2)
Translation-related noise parameter (only used if model is "omni").

~odom_frame_id (string, default: "odom")
Which frame to use for odometry.

~base_frame_id (string, default: "base_link")
Which frame to use for the robot base

~global_frame_id (string, default: "map")
The name of the coordinate frame published by the localization system

~tf_broadcast (bool, default: true)
Set this to false to prevent amcl from publishing the transform between the global frame and the odometry frame.

Figura 5: Configuración de nodos en ROS parte E

Anexos

Anexo 1. Criterios de selección en Áreas funcionales

	resolución	costo computacional	alcance de mediciones		
	C_1	C_2	C_3	Suma	vector prom
C_1	1	4	2	7	0.5
C_2	0.25	1	0.25	1.5	0.10714286
C_3	0.5	4	1	5.5	0.39285714
				14	1
resolucion					
	ROSBot	Ttb3	Jet		vector prom
ROSBot	1	0.5	2	3.5	0.28571429
Tb3	2	1	4	7	0.57142857
Jet	0.5	0.25	1	1.75	0.14285714
				12.25	1
cost comp					
	ROSBot	Ttb3	Jet		vector prom
ROSBot	1	0.5	0.25	1.75	0.13207547
Tb3	2	2	0.5	4.5	0.33962264
Jet	4	2	1	7	0.52830189
				13.25	1
alcane					
	ROSBot	Ttb3	Jet		vector prom
ROSBot	1	2	4	7	0.53503185
Tb3	0.5	1	3	4.5	0.34394904
Jet	0.25	0.333333333	1	1.583333333	0.12101911
				13.08333333	1
	res	proce	alca	total	
Rosbot	0.28571429	0.132075472	0.53503185	0.367199169	
Ttb3	0.57142857	0.339622642	0.34394904	0.457225265	
Jet	0.14285714	0.528301887	0.12101911	0.175575566	
Pondera	0.5	0.107142857	0.39285714		

Cuadro 1: Tabla de selección para Percepción

	velocidad	compatibi	documentacion		
	C_1	C_2	C_3	Suma	vector prom
C_1	1.0000	0.5000	2.0000	3.5000	0.3500
C_2	2.0000	1.0000	1.0000	4.0000	0.4000
C_3	0.5000	1.0000	1.0000	2.5000	0.2500
				10.0000	1.0000
vel					
	ROSBot	Ttb3	Jet		vector prom
ROSBot	1.0000	2.0000	0.1250	3.1250	0.1437
Tb3	0.5000	1.0000	0.1250	1.6250	0.0747
Jet	8.0000	8.0000	1.0000	17.0000	0.7816
				21.7500	1.0000
com					
	ROSBot	Ttb3	Jet		vector prom
ROSBot	1.0000	0.5000	1.0000	2.5000	0.2500
Tb3	2.0000	1.0000	2.0000	5.0000	0.5000
Jet	1.0000	0.5000	1.0000	2.5000	0.2500
				10.0000	1.0000
doc					
	ROSBot	Ttb3	Jet		vector prom
ROSBot	1.0000	2.0000	1.0000	4.0000	0.4000
Tb3	0.5000	1.0000	2.0000	3.5000	0.3500
Jet	1.0000	0.5000	1.0000	2.5000	0.2500
				10.0000	1.0000
	vel	com	doc	total	
Rosbot	0.1437	0.2500	0.4000	0.2503	
Ttb3	0.0747	0.5000	0.3500	0.3136	
Jet	0.7816	0.2500	0.2500	0.4361	
Pondera	0.3500	0.4000	0.2500		

Cuadro 2: Tabla de selección para Estructura

	modular	adaptable	peso		
	C_1	C_2	C_3	Suma	vector prom
C_1	1.0000	1.0000	0.2500	2.2500	0.1915
C_2	1.0000	1.0000	0.5000	2.5000	0.2128
C_3	4.0000	2.0000	1.0000	7.0000	0.5957
				11.7500	1.0000
modu					
	ROSBot	Ttb3	Jet		vector prom
ROSBot	1.0000	0.2500	1.0000	2.2500	0.1915
Tb3	4.0000	1.0000	2.0000	7.0000	0.5957
Jet	1.0000	0.5000	1.0000	2.5000	0.2128
				11.7500	1.0000
adap					
	ROSBot	Ttb3	Jet		vector prom
ROSBot	1.0000	0.5000	2.0000	3.5000	0.2857
Tb3	2.0000	1.0000	4.0000	7.0000	0.5714
Jet	0.5000	0.2500	1.0000	1.7500	0.1429
				12.2500	1.0000
peso					
	ROSBot	Ttb3	Jet		vector prom
ROSBot	1.0000	0.5000	2.0000	3.5000	0.2857
Tb3	2.0000	1.0000	4.0000	7.0000	0.5714
Jet	0.5000	0.2500	1.0000	1.7500	0.1429
				12.2500	1.0000
	res	proce	alca	total	
Rosbot	0.1915	0.2857	0.2857	0.2677	
Ttb3	0.5957	0.5714	0.5714	0.5761	
Jet	0.2128	0.1429	0.1429	0.1562	
Pondera	0.1915	0.2128	0.5957		

Cuadro 3: Tabla de selección para Procesamiento

	max current	modos de operación	torque		
	C_1	C_2	C_3	Suma	vector prom
C_1	1.0000	0.2500	1.0000	2.2500	0.2000
C_2	4.0000	1.0000	1.0000	6.0000	0.5333
C_3	1.0000	1.0000	1.0000	3.0000	0.2667
				11.2500	1.0000
max cu					
	ROSBot	Ttb3	Jet		vector prom
ROSBot	1.0000	0.2500	1.0000	2.2500	0.1915
Tb3	4.0000	1.0000	2.0000	7.0000	0.5957
Jet	1.0000	0.5000	1.0000	2.5000	0.2128
				11.7500	1.0000
modos					
	ROSBot	Ttb3	Jet		vector prom
ROSBot	1.0000	0.5000	1.0000	2.5000	0.2500
Tb3	2.0000	1.0000	2.0000	5.0000	0.5000
Jet	1.0000	0.5000	1.0000	2.5000	0.2500
				10.0000	1.0000
torque					
	ROSBot	Ttb3	Jet		vector prom
ROSBot	1.0000	2.0000	2.0000	5.0000	0.5000
Tb3	0.5000	1.0000	1.0000	2.5000	0.2500
Jet	0.5000	1.0000	1.0000	2.5000	0.2500
				10.0000	1.0000
Motores y controladores					
	max	modos	torque	total	
Rosbot	0.1915	0.2500	0.5000	0.3050	
Ttb3	0.5957	0.5000	0.2500	0.4525	
Jet	0.2128	0.2500	0.2500	0.2426	
Pondera	0.2000	0.5333	0.2667		

Cuadro 4: Tabla de selección para Motores