



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

Profesora Titular

Dra. Blanca Rosa Briseño Tepepa

M. en C. Griselda Sánchez Otero

DEDICATORIA

“La única diferencia que existe entre el éxito y el fracaso está en los ojos con los que te miras”.

Esta obra está dedicada a nuestros compañeros, amigos, familiares, maestros y a ti, amigo lector, sin embargo, además dado el mérito y esfuerzo involucrados directa e indirectamente decidimos dedicar esta obra a la sociedad, donde más allá de solamente reflejar nuestros conocimientos técnicos sobre la ingeniería mecatrónica, también les compartimos un ideal de proseguir en el beneficio de la humanidad y correr hasta la meta de la vida sin mirar atrás. A lo largo de este trabajo pasamos por diversas pruebas que enfrentamos en equipo, practicando la empatía y luchando contra la desesperación y el estrés, experimentando en cada una de las fases la bendición de Dios tanto en el ámbito profesional como en el psicológico, complementando de esta manera día a día nuestro desarrollo personal.

Esperamos que esta obra te sea de mucha utilidad, que te sirva como un punto de partida superior del que nosotros comenzamos y que encuentres valioso el conocimiento aquí reflejado muchas bendiciones.

Dichoso aquel a quien tú, SEÑOR, corriges; aquel a quien instruyes en tu ley, para que enfrente tranquilo los días de aflicción, mientras al impío se le cava una fosa.

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

En primer lugar, me gustaría agradecerle a DIOS, por todas las situaciones en las que nos guio, nos puso entre las personas correctas, nos dio el conocimiento para resolver varios problemas del proyecto y de dotarnos de la tenacidad para proseguir hasta el final de la meta.

A mis padres por ayudarme a lo largo de toda mi formación profesional, de criarme, educarme y mostrarme su afecto y compasión en cada una de las etapas de

mi vida, esto para mí son obras incorruptibles que no pueden ser compradas por oro o plata, en las cuales encuentro un verdadero tesoro.

A mis hermanos Francisco y Daniel por darme su ayuda en distintas ocasiones a lo largo de mis estudios donde sentía consuelo de no estar sólo y poder recibir un buen consejo.

A mis compañeros de equipo Mauricio y Toro por ser empáticos, colaborativos y dedicar todo su tiempo, empeño y esfuerzo a lo largo del trabajo terminal, compartiendo no solo experiencias laborales sino también de amistad.

A mis profesores de TT por ayudarnos en la revisión y corrección de nuestro trabajo al profesor Mata en particular por proponernos y financiarnos el proyecto y su valioso conocimiento en control, a la maestra Griselda por todos los consejos sobre el escrito que nos daba y al profesor Mauricio por su ayuda en la implementación del código y gusto por la programación, por último, también me gustaría agradecerle al profesor Juan Carlos por su ayuda genuina en la edición del documento.

A mis profesores por todas las enseñanzas, el valor del arduo trabajo, y la amistad que me dieron, haciendo mención especial a los profesores Álvaro Gordillo, Emilio Brito, Benjamín Trejo, Erick Huitrón, Samuel Domínguez, German Escoto, Sergio Garduza, Rafael Calzada Trujillo, Fermín Acosta, Sandra Martínez, Armando Mejía, Octavio Montes, Guillermo Carrasco, Sergio Ramírez, José Pablo García y David Eguiarte.

A mis amigos por brindarme su amistad, energía, momentos alegres, inspiración, experiencias únicas y una estancia de confianza y fraternidad a lo largo de mi vida estudiantil, la cual siempre tendré en mis recuerdos teniendo en mente a mis amigos Saul Salazar, César Contreras, Itan Zoali, Francisco Villanueva, Alejandro Ramírez, Mike Rodríguez, Isaí García, Guillermo Alessandro Alva, Luis Ángel Méndez, Exel Dávila, Francisco Morales, Rogelio Caballero, Aldo Arenales, Rafael Guzmán, Brandon Quintín, Carlos García, Rafael Trovamala, Alberto Rivera, Luis Fernando Almanza, David Menchaca, Miguel Cuellar, Edmundo Morado, Aaron Ramírez, Luis Sánchez, Karla Rodríguez, Laura Maruri, Irari Jiménez, Uriel Meneses, Iván Ma-

rroquín y en especial Arturo Clemente y Efraín Paredes porque además de ser muy bueno amigos fueron un ejemplo para mí de cómo seguir a Cristo.

A mis familiares por siempre darme el ánimo para proseguir en mis estudios, brindarme consejos, pasar momentos agradables con ellos, por sus felicitaciones, por su apoyo económico y por muchas más cosas que viví con ellos, haciendo mención especial a mí tíos Martha Aguilar, Luz María Aguilar, Angélica Pérez, María Zarazua, Edith Zarazua, Martha Reyes, a mis tíos Enrique Aguilar, Ramon Aguilar, Joaquín Aguilar, Jorge Zarazua, Javier Zarazua, a mis primos Ricardo Saloma, Gerardo Saloma, José Luis Guzmán, Norma Angélica, Jorge Reyes, Ariana Trejo, Luis Trejo, Susana Reyes y todos mis demás primos, tíos y por supuesto mis abuelitos.

Luis Fernando Zarazua Aguilar

Contenido

Nomenclatura	XI
Glosario	XV
Resumen/Abstract	XIX
Objetivos	XXI
Introducción	XXIII
Antecedentes	XXIV
Nacionales	XXIV
Internacionales	XXVI
Planteamiento del problema	XXVII
Organización del reporte	XXVIII
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

CONTENIDO



1.1.3.	AMCL	4
1.1.4.	VFH (Vector Field Histogram)	5
1.1.4.1.	Primer nivel: <i>Mapa 2D</i>	6
1.1.4.2.	Segundo nivel: <i>Histograma polar</i>	6
1.1.4.3.	Tercer nivel: <i>Comando de dirección</i>	7
1.1.5.	Esquemas líder - seguidor	8
1.2.	Marco Procedimental	9
1.2.1.	Sistema mecatrónico	9
1.2.2.	Diseño mechatrónico	10
1.2.3.	Proceso de análisis jerárquico (AHP)	10
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: <i>Estructura</i>	17
2.1.2.2.	Área Funcional 2: <i>Procesamiento</i>	17
2.1.2.3.	Área Funcional 3: <i>Percepción</i>	18
2.1.2.4.	Área Funcional 4: <i>Alimentación</i>	18
2.1.2.5.	Área Funcional 5: <i>Movimiento</i>	18
2.1.3.	Características y/o necesidades principales de cada área funcional	19
2.1.4.	Características principales de los robots considerados. ROS-bot, Turtlebot, TX1 “Jet” Robot Kit	20
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.	<i>Amabilidad</i>	26
2.1.6.3.	<i>Interactivida</i>	26
2.1.6.4.	<i>Redundancia óptima en el proceso de comunicación</i>	26
2.1.6.5.	<i>Eficiencia y utilidad</i>	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: <i>Estructura (Soporte de componentes)</i>	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	39
2.2.2.	Área Funcional 2: <i>Procesamiento (Comunicación de componentes)</i>	39
2.2.2.1.	Primera parte: <i>Dispositivos destinados para el procesamiento</i>	40
2.2.2.2.	Segunda parte: <i>Funciones a desarrollar en cada tarjeta</i>	45
2.2.2.3.	Tercera parte: <i>Diseño del software</i>	46
2.2.3.	Área Funcional 3: <i>Percepción (Comunicación entorno - robot)</i>	62
2.2.3.1.	Datos propioceptivos	63
2.2.3.2.	Encoders rotatorios	63
2.2.3.3.	Encoder óptico rotatorio de tipo relativo o incremental	63
2.2.3.4.	Encoder óptico absoluto	64
2.2.3.5.	Acelerómetros	65
2.2.3.6.	Datos exteroceptivos	65
2.2.3.7.	Sensores láser	65
2.2.3.8.	Sensor LIDAR LDS-01	67

CONTENIDO



2.2.3.9. Dimensiones	68
2.2.3.10. Acondicionamiento	69
2.2.4. Área Funcional 4: <i>Alimentación (Suministro de energía)</i>	70
2.2.4.1. Características Técnicas de la batería LiPo.	72
2.2.4.2. Ensamble de la batería LiPo con la estructura	72
2.2.5. Área Funcional 5: <i>Movimiento (Desplazamiento del robot)</i>	73
2.2.5.1. Actuadores Dynamixel	74
2.2.5.2. Análisis interno del motor	75
2.2.5.3. Características del motor	77
2.2.6. Integración del Sistema	83
2.2.6.1. Elementos de Software	85
2.2.7. HMI	88
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	95
3.1. Implementación del Área Funcional Estructura	95
3.1.1. Recomendaciones a considerar	95
3.1.2. Armado de los robots móviles	96
3.1.3. Resultados del proceso de armado	102
3.1.4. Verificación de ensamble del sistema	105
3.2. Implementación del Área Funcional Movimiento	105
3.3. Implementación del Área Funcional Percepción	106
3.3.1. Sensor LIDAR	106
3.3.2. Sensores del motor DYNAMIXEL	110



3.4. Implementación del Área Funcional Alimentación	112
3.5. Implementación del Área Funcional Procesamiento	113
3.5.1. Pasos previos	113
3.5.2. Comunicación	116
3.5.3. Prueba A: Verificación de la matriz de fuerza	119
3.5.3.1. Análisis de resultados	121
3.5.4. Prueba B: Validar el seguimiento de trayectoria	121
3.5.4.1. Análisis de resultados	123
3.5.5. Prueba C: Validar el algoritmo Líder-Seguidor	123
3.5.5.1. Análisis de resultados	124
3.5.6. Prueba D:Validar la evasión de obstáculos	125
3.5.6.1. Validación de seguimiento y evasión	129
3.5.6.2. Análisis de resultados	130
3.5.7. Prueba E: Validar la integración de evasión y seguimiento . .	130
3.5.7.1. Análisis de resultados	131
3.5.8. Realización de la HMI	131
3.5.8.1. Programa Principal	136
3.5.8.2. Ventana Gráfica	137
3.5.8.3. Programa de ROS	137
3.5.8.4. Framework de Trabajo	138
3.5.8.5. Puntos importantes a considerar	139
3.5.8.6. Modificación del Programa	139
3.5.8.7. Resultados de la ejecución	140
3.6. Especificaciones del Sistema	144
3.6.0.1. Especificaciones del Área de Trabajo	144
3.6.0.2. Especificaciones de la Alimentación	144
3.6.0.3. Especificaciones del Procesamiento	144

CONTENIDO



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 [9]. . .	xxv
2.	Ejecución del Sistema de Navegación Evasor de Obstáculos implementando un robot Turtlebot2 [19].	xxv
3.	Control de la formación en un plano. Trabajo de investigación de la Universidad de California [42].	XXVII
1.1.	<i>Ciclo: Ve, piensa, actúa.</i> Componentes que conforman un robot móvil autónomo.	2
1.2.	Marco de referencia global y el marco de referencia local del robot. .	2
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 [39].	21
2.3.	Robot móvil Turtlebot3 Burger [23].	22
2.4.	Robot móvil TX1 “Jet” [15].	23

ÍNDICE DE FIGURAS



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 [3].	39
2.13. Tarjeta ARM Cortex M7 [34].	44
2.14. Tarjeta Raspberry PI 3, Modelo B [32].	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 el punto meta”.	48
2.18. Diagrama UML del objeto <i>MovementLeader</i> a implementar en el nodo del líder.	48
2.19. Diagrama de flujo del programa principal del nodo del líder.	49
2.20. Diagrama de flujo del método “Follow”.	49
2.21. Diagrama de flujo del método “VectorFieldHistogram”.	50
2.22. Diagrama de flujo del método “Move to angle”.	51
2.23. Diagrama de flujo de los “Callbacks”.	52
2.24. Representación de las distancias entre el líder y el seguidor.	52
2.25. Diagrama UML del objeto <i>FollowerRobot</i> a implementar en el nodo seguidor.	53
2.26. Diagrama de flujo del programa principal del nodo seguidor.	54
2.27. Diagrama de flujo de los Callbacks para obtención de datos.	54
2.28. Diagrama de flujo del método <i>FollowerCallback</i>	55
2.29. Diagrama UML para nodo de distancia mínima.	56
2.30. Diagrama de flujo del programa principal del nodo de distancia mínima.	56
2.31. Diagrama de flujo método para obtener la distancia mínima al obstáculo.	57

ÍNDICE DE FIGURAS



2.32. Diagrama UML para nodo LaserVFH.	58
2.33. Diagrama de flujo del programa principal del nodo LaserVFH.	58
2.34. Diagrama de flujo para generar histograma binario.	59
2.35. Configuración de la IP para la comunicación vía WiFi [23].	61
2.36. Ejemplo de configuración WIFI [31].	61
2.37. Funcionamiento del encoder absoluto.	64
2.38. Funcionamiento de un sensor láser.	66
2.39. Funcionamiento del sensor LIDAR.	66
2.40. Dimensiones del sensor LIDAR. Vista superior y lateral.	68
2.41. Vista Frontal del sensor LIDAR.	68
2.42. Pines del sensor LIDAR.	69
2.43. Ubicación de la batería de alimentación. Primera capa del Turtlebot3. .	71
2.44. Batería LiPo, con su entrada y salida [33].	71
2.45. Ensamble de la batería LiPo en la primera capa del Turtlebot3. . . .	72
2.46. Ensamble del rin y la llanta de los robots.	73
2.47. Estructura de la locomoción de los robots.	74
2.48. Ensamble de los actuadores con la estructura.	74
2.49. Explosión del motor del Dynamixel.	75
2.50. Características del actuador.	76
2.51. Sistema de locomoción en conjunto con los actuadores Dynamixel XL 430 - W250.	76
2.52. Sistema de locomoción en conjunto con los actuadores Dynamixel XL 430 - W250. Vista frontal e inferior.	77
2.53. Nomenclatura de los actuadores Dynamixel [28].	77
2.54. Gráfica de funcionamiento del motor DYNAMIXEL [28].	81
2.55. Interfaz de programación de los actuadores Dynamixel.	82
2.56. Capa 1. Implementación de la capa con los actuadores y la batería LiPo. .	83
2.57. Capa 2. Implementación de la capa 1, capa 2 y tarjeta Open CR. . .	83

ÍNDICE DE FIGURAS



2.58. Capa 3. Implementación de la capa 1, capa 2, capa 3 y tarjeta Raspberry Pi 3	84
2.59. Vista isométrica del robot con todas las capas ensambladas.	84
2.60. Explosión y distribución de los componentes que conforman al robot.	85
2.61. Esquema general de funcionamiento del sistema de control para el seguimiento de trayectorias y evasión de obstáculos.	86
2.62. Esquema general de funcionamiento del sistema de control para el seguidor.	86
2.63. Comunicación entre tópicos de los distintos nodos.	87
2.64. Conexión típica de ROS.	88
2.65. Ejemplo de comunicación de ROS con agentes externos.	89
2.66. Trayectoria propuesta 1 (unidades en metros).	91
2.67. Trayectoria propuesta 2 (unidades en metros).	91
2.68. Matriz de fuerza para trayectoria propuesta 1 (unidades en metros).	91
2.69. Matriz de fuerza para trayectoria propuesta 2 (unidades en metros).	92
2.70. Resultados extraídos de la simulación de seguimiento de trayectoria (unidades en metros).	92
2.71. Resultados extraídos de la simulación del líder (unidades en metros).	93
2.72. Resultados extraídos de la simulación del seguidor (unidades en metros).	93
2.73. Obstáculos para la validación del algoritmo propuesto.	94
2.74. Puntos de ruta alcanzados por el líder durante la evasión de obstáculos (unidades en metros).	94
3.1. Caja con los diferentes componentes del robot.	96
3.2. Caja de componentes 1. <i>MicroSD, Raspberry Pi 3, motores Dynamixel y Tarjeta OpenCR Cortex M7</i>	97
3.3. Caja de componentes 2. <i>Bateria LiPo, cargador, desarmador, sensor LIDAR, conector USB2LDS y soportes para pcb's</i>	98
3.4. Caja de componentes 3. “Waffle - Plate”pisos de los robots	98

ÍNDICE DE FIGURAS



3.5. Caja de componentes 4. <i>Llantas, ruedas, tornillos, piezas de ensamble, cables y una rueda loca</i>	99
3.6. Fuente de voltaje para batería LiPo.	100
3.7. Unión de los pisos del robot.	100
3.8. Piezas Ensambladas.	101
3.9. Ensamble del robot.	101
3.10. Ensamble del primer piso.	102
3.11. Ensamble del segundo piso.	103
3.12. Ensamble del tercer piso.	103
3.13. Ensamble final del robot.	104
3.14. Robot líder y robot seguidor.	105
3.15. Implementación del área funcional movimiento.	106
3.16. Montaje del sensor LIDAR en la estructura.	107
3.17. Sensor LIDAR del Turtlebot 3 burger.	108
3.18. Distancias del LIDAR en RVIZ	109
3.19. Visualización en rqt de los valores del sensor LIDAR.	110
3.20. Motores DYNAMIXEL XL430.	110
3.21. Ensamble de los motores, OpenCR y batería LiPo.	111
3.22. Localización de los botones de prueba en la tarjeta OpenCR.	111
3.23. Batería LiPo LB-012.	113
3.24. Creación de una nueva red en Ubuntu.	116
3.25. Elegir tipo de conexión.	116
3.26. Configuración del Modo y Nombre.	117
3.27. Configuración de la seguridad.	117
3.28. Muestra del resultado correcto de la ejecución del comando ping.	118
3.29. RQt mostrando los datos del tópico battery.	118
3.30. Reconstrucción de la trayectoria A.	119
3.31. Reconstrucción de la trayectoria B.	120
3.32. Reconstrucción de la trayectoria C.	120

ÍNDICE DE FIGURAS



3.33. Puntos de trayectoria A.	120
3.34. Puntos de trayectoria B.	121
3.35. Puntos de trayectoria C.	121
3.36. Error del líder en seguimiento de la trayectoria A.	122
3.37. Error del líder en seguimiento de la trayectoria B.	122
3.38. Gráfica de las posiciones alcanzadas y el error.	123
3.39. Error del seguidor en la trayectoria A.	124
3.40. Error del seguidor en la trayectoria B.	124
3.41. Método para calcular la distancia mínima y publicarla.	125
3.42. Detección del obstáculo cercano colocado en el primer cuadrante. . .	126
3.43. Detección del obstáculo cercano colocado en el cuarto cuadrante. . .	126
3.44. Función Follow para ejecutar VFH o seguimiento de trayectoria. . .	127
3.45. Función de seguimiento de trayectoria por dirección.	128
3.46. Algoritmo para aplicar VFH.	129
3.47. Error del Líder durante la evasión.	130
3.48. Error del Seguidor durante la evasión.	131
3.49. RQt para graficar distintos tópicos con respecto al tiempo.	132
3.50. RQt mostrando y filtrando los distintos tipos de mensajes.	132
3.51. Pantalla de bienvenida de QtCreator.	133
3.52. Ventana para compilar paquete en QtCreator.	134
3.53. Ingreso de la carpeta de compilación.	134
3.54. Ingreso y compilación con el comando CMake.	135
3.55. Ventana en Qt con el paquete qgui.	136
3.56. Framework de Qt para GUI.	138
3.57. Ventana inicial de la interfaz.	141
3.58. Configuración de la interfaz con la trayectoria 1 seleccionada y apagada.	141
3.59. Configuración de la interfaz con la trayectoria 2 seleccionada y encen-	
dida.	142

ÍNDICE DE FIGURAS



3.60. Configuración de la interfaz con la trayectoria 3 seleccionada y encen- dida.	142
3.61. Interfaz con los mensajes de prueba enviados.	143
3.62. Interfaz con el historial de trayectorias seleccionadas.	143
3.63. Interfaz mostrando las opciones de graficación.	144
3.64. Nodos de Rosgraph.	145
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 [34]	41
2.9.	Características de la Tarjeta ARM Cortex M7 [34].	42
2.10.	Características de la Tarjeta ARM Cortex M7 [34].	43
2.11.	Características de la tarjeta Raspberry Pi 3, Modelo B [32].	44
2.12.	Características del sensor LIDAR.	67
2.13.	Descripción de los pines del sensor LIDAR.	69
2.14.	Descripción de los pines del motor del sensor LIDAR.	70
2.15.	Comandos para operación del LIDAR.	70
2.16.	Especificaciones técnicas de la batería. [12]	72

ÍNDICE DE TABLAS



2.17. Características técnicas del actuador Dynamixel XL 430 W250 [27]..	78
2.18. Características técnicas del actuador Dynamixel XL 430 W250 [27]..	79
2.19. Características técnicas del actuador Dynamixel XL 430 W250 [27]..	80
 4.1. Tabla de costos variables.	148
4.2. Días trabajados en 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 - Dos dimensiones

3D - Tres 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

Nomenclatura



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

FPU - Floating Point Unit

SSH - Secure SHell

DFU - Device Firmware Upgrade

GPIO - General Purpose Input/Output

PWM - Pulse-Width Modulation

I2C - Inter-Integrated Circuit

SPI - Serial Peripheral Interface

OTG - On The Go

RTC - Real Time Clock

SMPS - Switched-Mode Power Supply

LED - Light-Emitting Diode

TTL - Transistor-Transistor Logic

CAN - Controller Area Network

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 [44].

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 la 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, habrá un nodo más que proporcione una vista gráfica del sistema y otro que manipule todos



los datos obtenidos del sensando y así, de la misma manera 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*, el cual, literalmente se puede ver como un tema de conversación [44]. Cuando un nodo está interesado en un determinado tipo de dato se *subscribe* 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 [2].

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 [37]. En otras palabras, la forma en que un nodo envía o recibe información es a través de un *mensaje*.

Namespace

Un *namespace* es un identificador que se le pone a los tópicos con el fin de diferenciar uno del otro cuando se tienen tópicos con un mismo nombre. Por ejemplo, cuando se tienen más de una variable con mismo nombre como en el caso de dos o más baterias baterias o dos o más actuadores, se les ponen un índice identificado para diferenciar uno del otro.

Callback

Callback o *devolución de llamada*, es una función en ROS que 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.

Framework

Un *framework* es una estructura compuesta de componentes personalizables e intercambiables para el desarrollo de un programa o aplicación. En otras palabras, una framework se puede considerar una aplicación genérica incompleta y configurable a la que le podemos añadir las últimas piezas para construir una aplicación concreta [13].

Filtrado de partículas

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

RViz

Es una herramienta de visualización 3D que pose ROS cuya principal característica es mostrar mensajes en 3D, permitiendo la verificación visual de los datos [44].



Gazebo

Es un simulador 3D independiente que soporta ROS y que permite el diseño de robots, la prueba de algoritmos, el análisis de regresiones, el entrenamiento de inteligencia artificial, etc., y el cual es capaz de soportar a varios robots. La principal diferencia entre RViz está en que RViz es una herramienta de visualización la cual no permite obtener los cambios físicos en los robots como inercia, toque, colisión etc., ni tampoco obtener información del entorno circundante en tiempo real [44].

Rqt

Es una herramienta para la creación de GUIs (por sus siglas en inglés *Graphical User Interface*) que posee alrededor de 30 herramientas para el desarrollo y que viene integrada a partir de la versión ROS Fuerte [44].

Qt

Es un framework multiplataforma utilizado ampliamente en la programación de GUIs y del cual, rqt esta basado [44].

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, VFH.

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*”,

Resumen/Abstract



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.

Objetivos



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 la robótica, la cual, ofrece una solución para los problemas que el ser humano desea resolver en diversos escenarios de su cotidianidad. Desafortunadamente no todas las tareas pueden ser realizadas por un solo robot y es ahí, donde surge la robótica cooperativa, proporcionando métodos, teoremas e investigaciones para su desarrollo [26].

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 [7], como son: 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; permitiendo 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 [21].



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 manera 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, en donde encontramos entre las más conocidas: las *estructuras virtuales*, las configuraciones *líder-seguidor* y las *aproximaciones basadas en comportamientos* [26]. Cada una de estas técnicas ha venido siendo estudiada, y aplicada por diferentes sectores de la sociedad, destacando 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 ROS (Robotic Operating System) y se ejecutó a través de una tarjeta Nvidia Jetson y un sensor LIDAR para la recabación de datos. El trabajo consiste en un robot capaz de cargar hasta 15 kg y el cual dentro de un sistema de navegación (área de trabajo) genera una trayectoria de manera autónoma con la finalidad de llegar a trayectorias “meta”, transportando cualquier producto que no sobrepase la carga máxima que es 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 inglés *Monte Carlo Localization*) para la ubicación del robot dentro del área de trabajo, al igual que la generación de campos estáticos para su navegación autónoma.

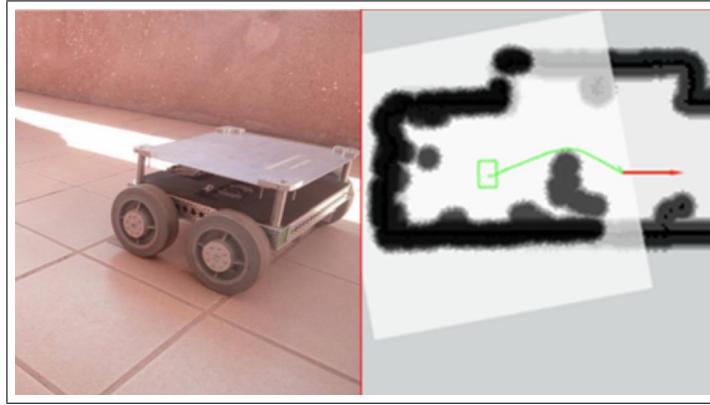


Figura 1: (a) Robot Repartidor Autónomo. (b) Generación de trayectoria [9].

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 implementa un robot Turtlebot2 que tiene como objetivo llegar de un punto inicial a un punto final, evadiendo obstáculos estáticos en el camino. En este trabajo se implementan distintos algoritmos de control, como: algoritmo generador de celdas de ocupación probabilística en un mapa 2D, algoritmo de planeamiento global utilizando información 2D del área de trabajo, junto con un algoritmo de planeamiento local reactivo.

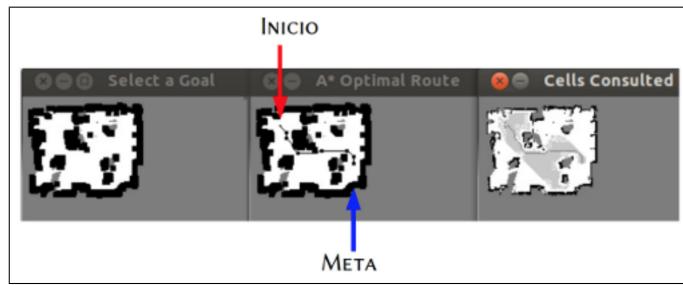


Figura 2: Ejecución del Sistema de Navegación Evasor de Obstáculos implementando un robot Turtlebot2 [19].

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”, la cual habla sobre propuesta de un control visual para la formación de robot móviles, considerando una sola cámara fija que observa el espacio de trabajo de los robots y da información sobre los movimientos y las posiciones de los robots, cuya dicha información es compartida tanto para el líder como para el seguidor [7].

Internacionales

En la Universidad Militar Nueva Granada (Bogotá, Colombia) en el 2014, llevó a cabo una investigación que se tituló “*Flotilla de robots para trabajos en robótica cooperativa*” la cual habla del diseño de un sistema de control descentralizado, basado en el control líder - seguidor que permite orientar un grupo de robots mientras se desplazan, manteniendo una formación predeterminada [21].

Por parte del IEEE (Instituto de Ingeniería Eléctrica y Electrónica) se llevó a cabo un trabajo titulado “*Backstepping based multiple mobile robots formation control*” (formación de múltiples robots móviles basado en el control Backstepping) el cual es un trabajo de investigación acerca del control de formación de múltiples robots móviles, liderados por un robot denominado “líder”, y en el cual se presenta un modelo cinemático para el sistema “líder-seguidor”, haciendo uso de coordenadas cartesianas en lugar de coordenadas polares y tomando la idea del integrador backstepping, derivándose en un control global estable para todo el sistema[42].

La Universidad Politécnica Superior de Elche desarrolló un trabajo de investigación llamado “*Seguimiento de robots mediante visión artificial*”, el cual es un trabajo aplicado al mantenimiento de formaciones y en donde se describe el desarrollo e implementación de algoritmos de navegación que permite a un robot móvil seguir la ruta realizada por otro robot, 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[17].

En el Departamento de Ingeniería Electrónica y Ciencia de la Comunicación de La Universidad de California se desarrolló un trabajo de investigación llamado “*Formation control of nonholonomic mobile robots with omnidirectional visual servoing and motion segmentation*” (Control para la formación de robots móviles no-holonómicos con servo visión omnidireccional y movimiento segmentado) el cual habla de un conjunto de robots móviles omnidireccionales basado en el diseño líder-seguidor usando visión omnidireccional, y en donde se especifica la formación deseada en un plano y esta después es traducida en un control servo visual para cada seguidor. En el trabajo también se muestra 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 [42].

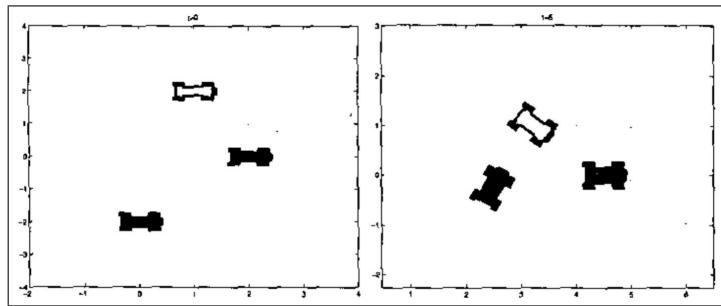


Figura 3: Control de la formación en un plano. Trabajo de investigación de la Universidad de California [42].

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 pro-



porcionada 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 a cabo un análisis del costo total de proyecto, partiendo de sus costos fijos y costos variables.

En el sexto y último capítulo *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 son necesarias 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 los algoritmos informáticos de inteligencia artificial y de la teoría probabilística para el cumplimiento de tareas de manera autónoma.

Todos estos elementos forman el llamado ciclo “*Ve, piensa, actúa*”, tal como se muestra en la Figura 1.1 [38].

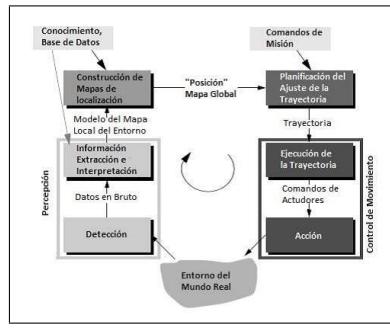


Figura 1.1: *Ciclo: Ve, piensa, actúa.* Componentes que conforman un robot móvil autónomo.

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 [38].

En la Figura 1.2 la posición de “ P ” en el marco de referencia global se especifica mediante las coordenadas “ x ” y “ y ”, y la diferencia angular entre los marcos de referencia global y local viene dada por “ θ ”, pudiéndose describir la posición del robot (ξ_I) 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.

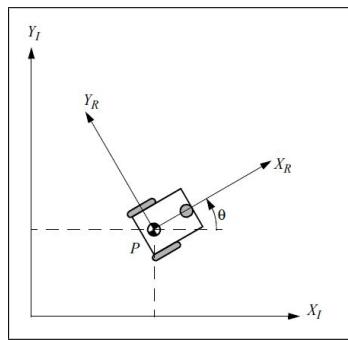


Figura 1.2: Marco de referencia global y el marco de referencia local del robot.



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

ξ_I Posición conforme a un marco de referencia I .

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 del robot al global y viceversa. Las expresiones para obtener el movimiento conforme a cada marco de referencia son:

$$\dot{\xi}_R = R(\theta) \dot{\xi}_I$$

$$\dot{\xi}_I = R(\theta)^{-1} \dot{\xi}_R$$

$\dot{\xi}_R$: Velocidad conforme al marco de referencia global.

$\dot{\xi}_I$: 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.

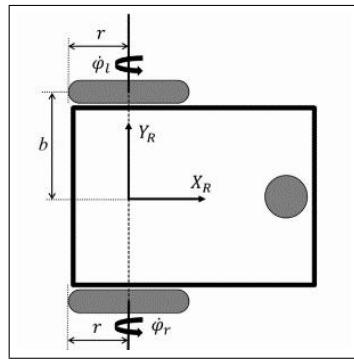


Figura 1.3: Parámetros de una configuración diferencial.

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, 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 [40], junto con los siguientes algoritmos.

```

1: Algorithm Augmented_MCL( $\mathcal{X}_{t-1}, u_t, z_t, m$ ):
2:   static  $w_{\text{slow}}, w_{\text{fast}}$ 
3:    $\bar{\mathcal{X}}_t = \mathcal{X}_t = \emptyset$ 
4:   for  $m = 1$  to  $M$  do
5:      $x_t^{[m]} = \text{sample\_motion\_model}(u_t, x_{t-1}^{[m]})$ 
6:      $w_t^{[m]} = \text{measurement\_model}(z_t, x_t^{[m]}, m)$ 
7:      $\mathcal{X}_t = \mathcal{X}_t + \langle x_t^{[m]}, w_t^{[m]} \rangle$ 
8:      $w_{\text{avg}} = w_{\text{avg}} + \frac{1}{M} w_t^{[m]}$ 
9:   endfor
10:   $w_{\text{slow}} = w_{\text{slow}} + \alpha_{\text{slow}}(w_{\text{avg}} - w_{\text{slow}})$ 
11:   $w_{\text{fast}} = w_{\text{fast}} + \alpha_{\text{fast}}(w_{\text{avg}} - w_{\text{fast}})$ 
12:  for  $m = 1$  to  $M$  do
13:    with probability  $\max(0.0, 1.0 - w_{\text{fast}}/w_{\text{slow}})$  do
14:      add random pose to  $\mathcal{X}_t$ 
15:    else
16:      draw  $i \in \{1, \dots, N\}$  with probability  $\propto w_t^{[i]}$ 
17:      add  $x_t^{[i]}$  to  $\mathcal{X}_t$ 
18:    endwith
19:  endfor
20:  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 y 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* sin interferir con el objetivo principal, como el de seguir una trayectoria. El método toma como entrada los datos adquiridos 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 dividido por celdas donde se guardan 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 este está cercano.



1.1.4.1. Primer nivel: *Mapa 2D*

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

$$m_{(i,j)} = (c_{(i,j)}^*)^2(a - bd_{(i,j)}^2) \quad (1.1)$$

$c_{(i,j)}^*$: 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 cumplen $a - bd_{max} = 0$

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 la distancia mayor a la celda más lejana al robot, obteniéndose “ a ” y “ b ” de manera tal que, la distancia más alejada devuelva un valor de 0 y la distancia más cercana devuelve el valor máximo posible y a su vez se mantenga 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*, dando como primer paso la definición de 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*) el cual nos indica la distancia más cercana al objetivo para cada determinados grados (por lo general 5). Para cada sector k , el histograma polar h_k es calculado siguiendo la *ecuación 1.2*.

$$h_k = \sum_{i,j} m_{i,j} \quad (1.2)$$

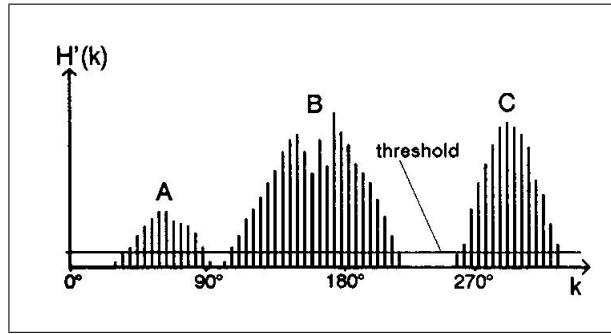


Figura 1.5: Ejemplo de histograma polar.

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 y k_l y según un umbral s_{max} (definido por el usuario y la cual depende de 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.

$$c_d = \frac{k_r + k_l}{2} \quad (1.3)$$

$$c_r = k_r + \frac{s_{max}}{2} \quad (1.4)$$

$$c_l = k_l - \frac{s_{max}}{2} \quad (1.5)$$

$$c_t = k_t \text{ si } k_t \in [c_r, c_l] \quad (1.6)$$

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

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

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 del control bajo un esquema líder-seguidor, se puede dar uno 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.

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 [22].

Dicha definición es muy parecida a lo que también es conocido como estructuras virtuales [21] dentro de los MRS (por sus siglas en inglés Multi Robot Systems).

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 usarán el trayecto definido para alcanzar una meta o lograr una tarea definida [20].



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 en la 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 de 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 [14]*”.

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.



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 como 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 [5]:

- 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 Process*) es un proceso diseñado e implementado por Thomas L. Saaty [10]. 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 todas las necesidades que el cliente tenga. 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 de este trabajo 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 o construcción 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 pudieran estar dentro de la 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 la trayectoria y la configuración líder–seguidor.

Tabla 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 con la posibilidad de ser adaptada 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.
- 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 en el control líder-seguidor.

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*, la cual permite la visualización de una tarea complicada, en tareas o funciones más sencillas que contribuyen a resolver el problema principal.

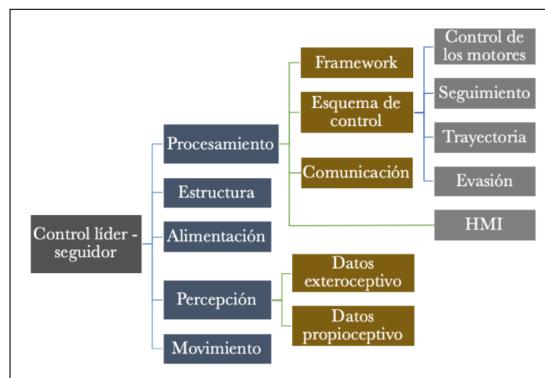


Figura 2.1: Áreas Funcionales propuestas.



2.1.2.1. Área Funcional 1: *Estructura*

La primera área de interés es la plataforma o estructura del móvil, donde se albergan todas las herramientas que cumplen con funciones o tareas posteriores. Esta área es la encargada de soportar todos los circuitos, sensores, actuadores, mecanismos de locomoción, fuentes de energía y tarjetas de procesamiento, capaces de satisfacer las necesidades de los robots y en particular del proyecto.

Es importante mencionar que no se realiza algún diseño de esta, sino que, se pretenden analizar las propuestas comerciales existentes o con acceso a un manual para su construcción. La razón es que al estar en el mercado se tiene la certeza que soportan todos los elementos que lleva un móvil, con un proceso de diseño concurrente, una línea de producción, un sistema de pruebas, documentación y al ser comerciales, con costos atractivos.

Dentro de los móviles que se seleccionaron cada uno posee características específicas, aunque similares, y es ahí donde entra la metodología mecatrónica para la selección de la opción más adecuada que se encuentre disponible en el mercado o que tenga la documentación suficiente para construirse, teniendo en cuenta los requerimientos antes mencionados, como 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*”. El control, en términos de un diagrama IDEF0 es la selección del algoritmo a usar para el cumplimiento de cada una de las funciones, adicionalmente se necesita establecer algún tipo de comunicación (preferentemente inalámbrica) entre ambos móviles y la computadora central, la cual permita al robot seguidor conocer los puntos de interés por alcanzar proporcionados por el robot líder. Así mismo, monitorear diversos parámetros durante la ejecución del programa.



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 a cabo 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 implementa 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 lo tanto, es necesario que la fuente de alimentación pueda ser portable y que a su vez, pueda ser capaz de suministrar una potencia suficiente al móvil, principalmente por los motores que son los que mayor cantidad de energía requieren. Asimismo, al ser portable debe de ser una batería con una capacidad de energía considerable para que los móviles puedan realizar su tarea totalmente sin dejarla a medias o en algún momento interferir con el proceso.

2.1.2.5. Área Funcional 5: *Movimiento*

La parte del movimiento, tiene como característica fundamental la elección del tipo de locomoción ya sea Ackerman, triciclo, diferencial, etc. Este tipo de locomoción tiene que estar lo suficientemente documentada para poder satisfacer los requerimientos del proyecto. Otro parámetro importante en esta función, son los actuadores y sus controladores, los cuales son una parte fundamental para lograr tener un funcionamiento adecuado, refiriéndose a la capacidad de transformar los comandos generados por los esquemas de control en acciones que además tengan un control de velocidad y dirección lo suficientemente preciso para asegurar que los



móviles sigan la trayectoria de una manera correcta y consistente.

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.
 - Herramientas de simulación, visualización de información y desarrollo de interfaces gráficas.
 - Facilidad de programación, física y digital.
 - Compatibilidad con 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. ROS-bot, Turtlebot, TX1 “Jet” Robot Kit

Para la selección de los robots móviles se hizo una búsqueda contemplando los criterios antes mencionados, las opciones 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 [39].

En la *Figura 2.2* se muestra el *Rosbot* el cual es un robot basado en el sistema operativo ROS (Robotic Operating System) que cuenta con un sensor LIDAR 360º (RPLIDAR A2) y una cámara RGBD. En 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. La segunda es un controlador 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 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 [23].

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. 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

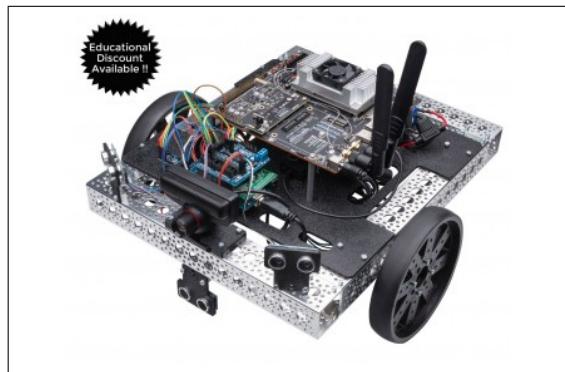


Figura 2.4: Robot móvil TX1 “Jet” [15].

En la *Figura 2.4* se muestra el *TX1 “Jet”* el cual es un robot que utiliza componentes Acobotronics 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	

Tabla 2.2: Tabla de Selección: Procesamiento.

Percepción				
	Resolución de sensores	Costo de pro- cesamiento	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	

Tabla 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	

Tabla 2.4: Tabla de Selección: Estructura.

Motores y Controladores				
	Corriente Máxima	Modos de ope- ración 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	

Tabla 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 consultar los 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 se seleccionaron los más importantes. Estos criterios para el diseño y selección de interfaces gráficas son mencionados a continuación.

2.1.6.2. *Amabilidad*

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

2.1.6.3. *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. *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. *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		

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



2.1.6.7. Lista de opciones de interfaz

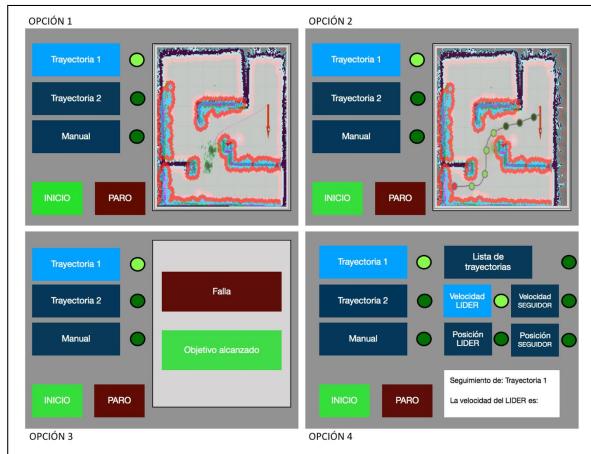


Figura 2.5: Opciones de interfaces gráficas

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.

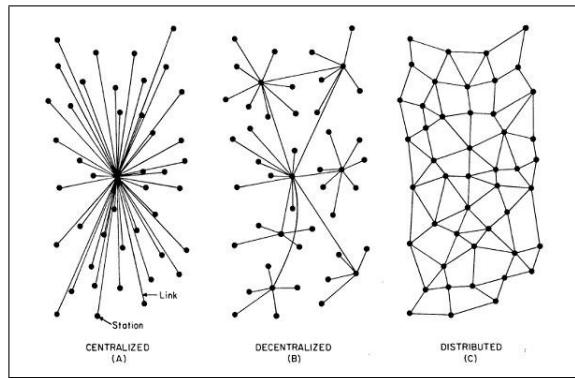


Figura 2.6: Tipos de Arquitecturas

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

	Centralizado	Descentralizado	Distribuido
Características	<ul style="list-style-type: none"> ■ Fácil de implementar. ■ Un solo servidor. ■ Difícil escalar. ■ Un solo punto de falla 	<ul style="list-style-type: none"> ■ 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. 	<ul style="list-style-type: none"> ■ Sistemas distribuidos, con poder de cómputo a través de los múltiples nodos, en lugar de 1.

Tabla 2.7: Comparación entre Arquitecturas



Sin embargo, para el caso en particular, 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 la 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 realiza el procesamiento en cada uno de los robots móviles, así como en la computadora de monitoreo. Cada elemento dentro del sistema procesará partes de código específicas, relacionadas con sus acciones de manera autónoma sin necesidad de consultar todas sus acciones con un nodo central.

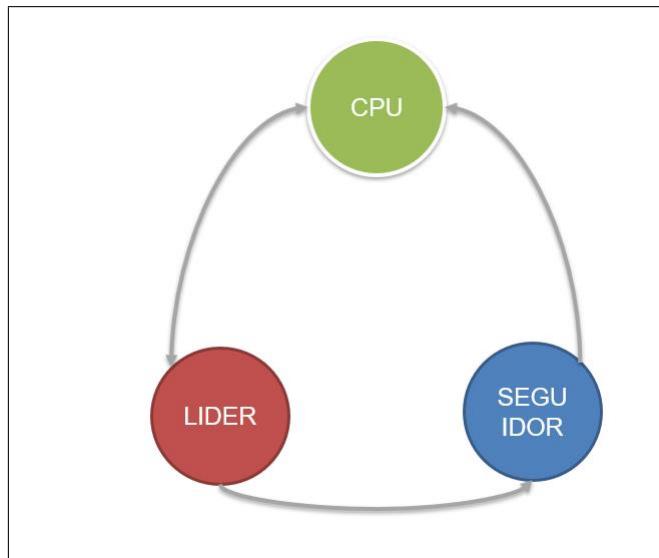


Figura 2.7: Arquitectura de software propuesta para el proyecto

Adicionalmente cada robot móvil tienen 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

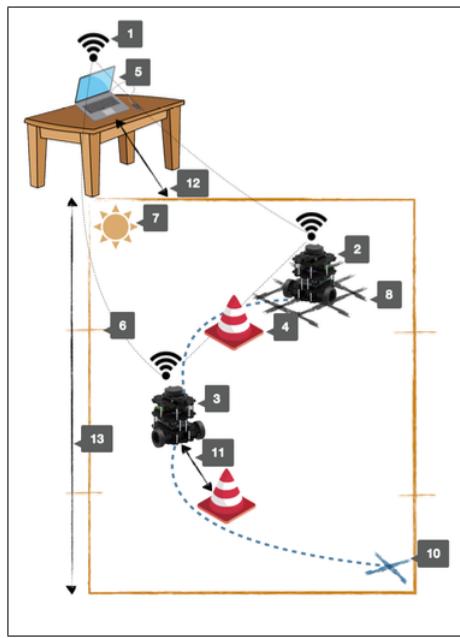


Figura 2.8: Concepto final

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 más a profundidad cada una de las áreas funcionales previamente analizadas 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 conforman a los robots, tal y como se puede observar en la *Figura 2.9*.

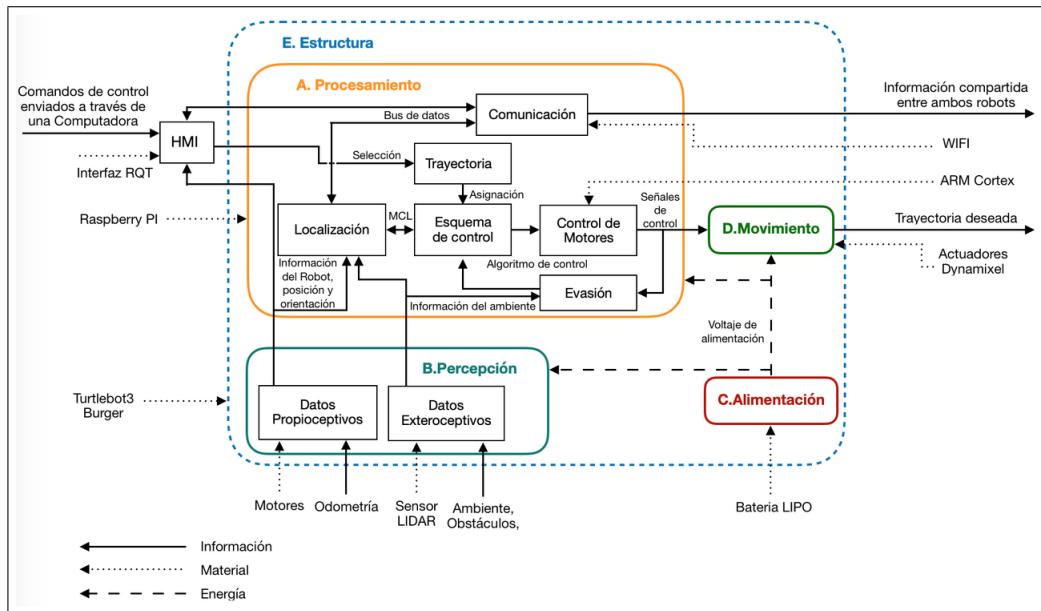


Figura 2.9: Integración de todas las Áreas Funcionales.

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 forman a los robots que se utilizan para llevar a cabo 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 estas estén 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 aplicaciones. Asimismo cabe mencionar que la estructura en conjunto con la batería LiPo y el sensor LIDAR es de apenas 1 kg [29] [3] [4].

Por otro lado, al tener una estructura modular dentro del robot, y estando clasificado el TurtleBot3 Burger como un robot móvil de *Código Abierto* con flexibilidad en el diseño mecánico, se puede adaptar a cualquier tipo de locomoción, aunque cabe mencionar que el desarrollo del proyecto se lleva a cabo en la configuración original con la que vienen integrados los robots, que es de tipo diferencial y de la cual en la sección de *Movimiento* se habla 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. Por otra parte, sirven para adaptar diferentes componentes, haciendo los pisos



de la estructura universales para cualquier requerimiento, adaptación o aplicación.

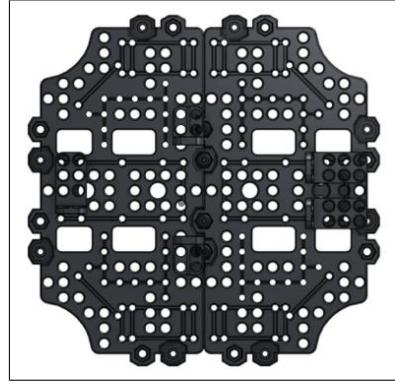


Figura 2.10: Ensamble de 2 pisos o chapas en forma de celda (base estructural).

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 manejabilidad 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 la tarea que sea requerida por el robot.

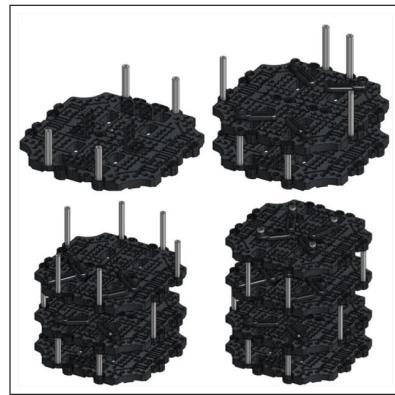


Figura 2.11: Estructura tipo multiplataforma.



2.2.1.2. Elementos utilizados en cada piso

Elementos generales

Para el ensamblaje de cada piso de la estructura, se utilizan 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 colocan 4 tornillos M3 de 8mm en la parte superior de la chapa, y por la parte inferior para ajustar se usan 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 batería 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 burger*, 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*.

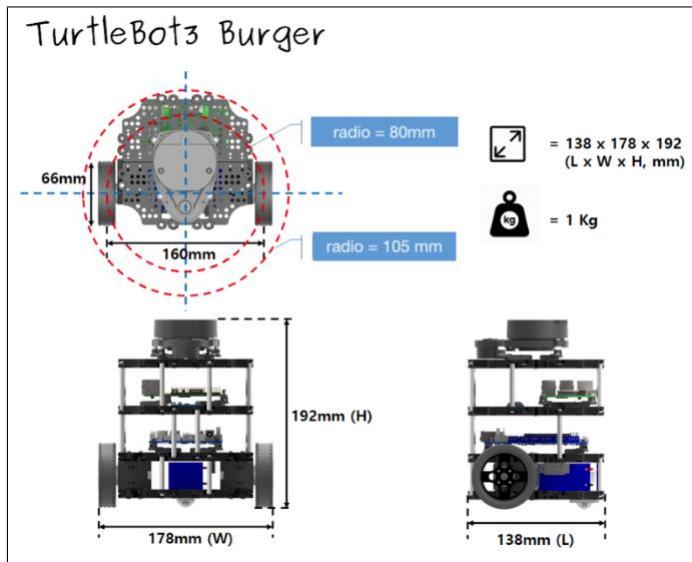


Figura 2.12: Dimensiones del Turtlebot3 [3].

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

El *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 el mayor número de subáreas, es una de las partes principales donde recae todo el desarrollo del esquema de control líder - seguidor.

El desarrollo de esta área funcional se dividió en tres secciones con el objetivo de volver más detallado el proceso de diseño. La primera parte consiste en la *descripción de las tarjetas encargadas del procesamiento*, en donde 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 propone 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, y es a través de este pensar que los robots pueden ejecutar las acciones que el procesamiento solicite. Son cuatro los dispositivos donde existe o se lleva 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	<ul style="list-style-type: none"> 1. STM32F746ZGT6 / 32-bit ARM Cortex-M7 con FPU (216MHz, 462DMIPS).
Sensores	<ul style="list-style-type: none"> 1. Giroscopio 3 ejes. 2. Acelerómetro 3 ejes. 3. Magnetómetro 3 ejes (MPU9250).
Programador	<ul style="list-style-type: none"> 1. ARM Cortex 10 pines JTAG/SWD conector serial USB. 2. Dispositivo de Actualización de Firmware (Device Firmware Upgrade - DFU).
Digitales I/O	<ul style="list-style-type: none"> 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.
Ing. Mecatrónica	UPIITA



Elemento	Características
Entradas analógicas	<ul style="list-style-type: none"> 1. 6 canales de Convertidores Analógicos - Digitales de 12 bits de resolución.
Puertos de comunicación	<ul style="list-style-type: none"> 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	<ul style="list-style-type: none"> 1. LD2 (rojo/verde) : comunicación USB. 2. LED de usuario x 4 : LD3 (rojo), LD4 (verde), LD5 (azul). 3. Botón de usuario x 2.

Tabla 2.9: Características de la Tarjeta ARM Cortex M7 [34].



Elemento	Características
Alimentación	<ul style="list-style-type: none"> 1. Fuente de entrada externa. 2. 5 V (USB VBUS), 7-24 V (Batería o SMPS). 3. Batería por defecto : LiPo 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 batería 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 (para reset de la tarjeta). 10. Botón de encendido o apagado x 1.
Dimensiones	<ul style="list-style-type: none"> 1. 105(W) X 75(D) mm.
Masa	<ul style="list-style-type: none"> 1. 60 gramos.

Tabla 2.10: Características de la Tarjeta ARM Cortex M7 [34].

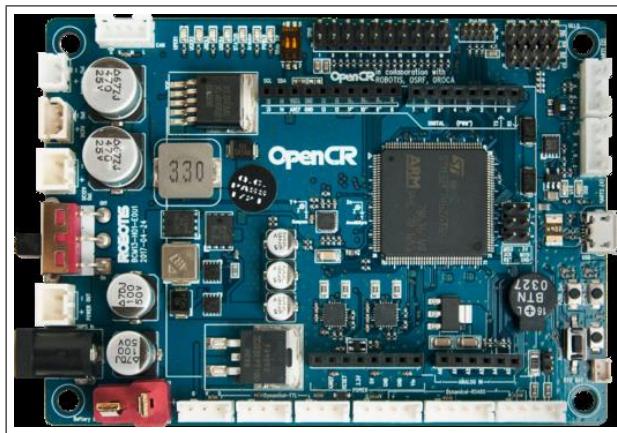


Figura 2.13: Tarjeta ARM Cortex M7 [34].

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

Tabla 2.11: Características de la tarjeta Raspberry Pi 3, Modelo B [32].

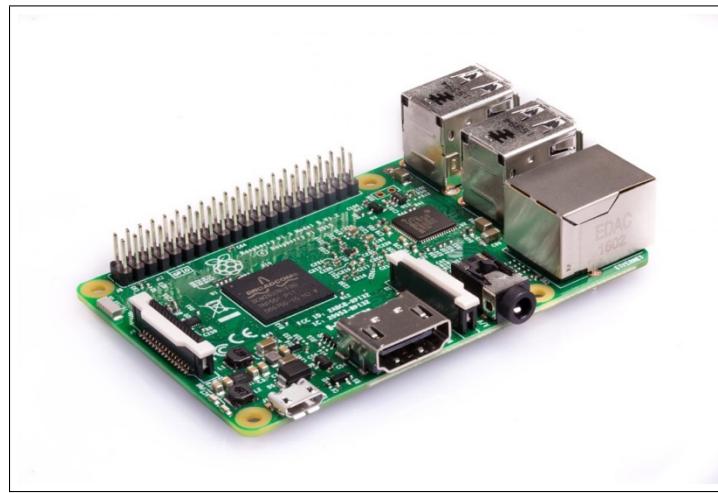


Figura 2.14: Tarjeta Raspberry PI 3, Modelo B [32].

2.2.2.2. Segunda parte: *Funciones a desarrollar en cada tarjeta*

Una vez definidos los elementos necesarios, sigue preguntarse ¿qué 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 la que se procesan los datos y el funcionamiento del sistema en general.

Esquemas de control: Son los encargados de definir la dirección de los robots 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 mensajes entre los distintos 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 “¿cómo se cumplirán?”.

2.2.2.3. Tercera parte: *Diseño del software*

Seguimiento de trayectorias Para el seguimiento de trayectorias se ha optado por seleccionar la *Matriz de fuerzas* (mapa de vectores que cubre toda el área del lugar) [4], como método de seguimiento por donde el líder se desplaza. El algoritmo probabilístico llamado *AMCL* permite conocer la posición de los robots con una incertidumbre de $2.5cm^2$, facilitando 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 líder, esto se hace discretizando la trayectoria propuesta y enlistando dichos puntos en un archivo con extensión .txt. 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. En la *Figura 2.16* se observa el algoritmo usado en el programa principal para generar la matriz de fuerza y en la *Figura 2.17* se muestra como se calcula para cada punto de la matriz el elemento destino o meta.

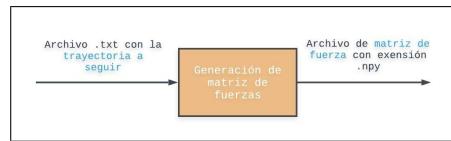


Figura 2.15: Entrada y salida del programa que genera la matriz de fuerza.

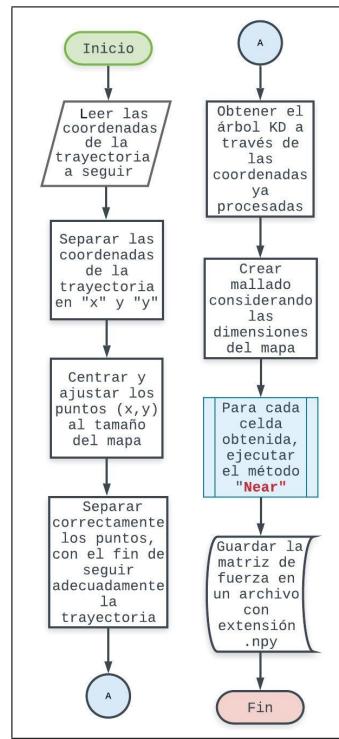


Figura 2.16: Algoritmo Generador de la Matriz de Fuerza.

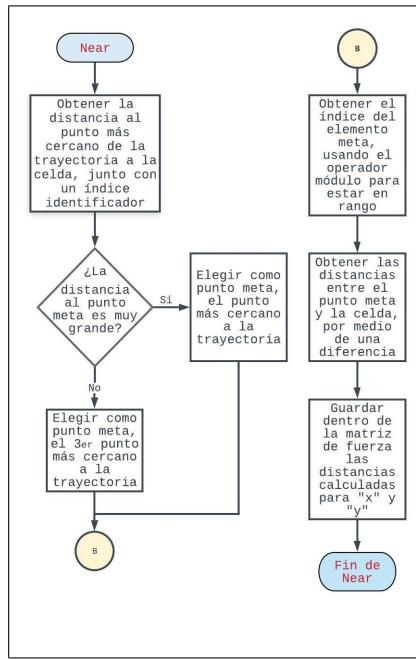


Figura 2.17: Diagrama de Flujo para la rutina “Obtener el punto meta”.

Una vez obtenida la matriz de fuerza se debe implementar un código para que el líder pueda seguir dicha trayectoria. La implementación del 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, 2.20, 2.21, 2.22* y *2.23* se muestran los diagramas de flujo que explican la funcionalidad de los métodos de la clase propuesta.

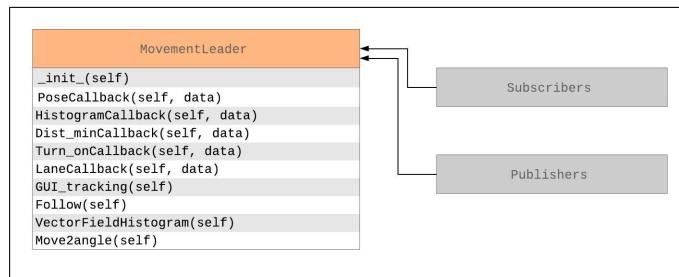


Figura 2.18: Diagrama UML del objeto *MovementLeader* a implementar en el nodo del líder.



En la *Figura 2.19* se observa el programa principal del nodo encargado de asignarle el movimiento al líder, este programa al construir el objeto crea las variables para publicadores y subscriptores que se mencionan en el Diagrama UML de la *Figura 2.18*, y además configura todas las condiciones y funciones a ejecutar. Como acción principal, este programa ejecuta el método “*Follow*”, brindando de la capacidad de movimiento a los robots, en caso de ser necesario.

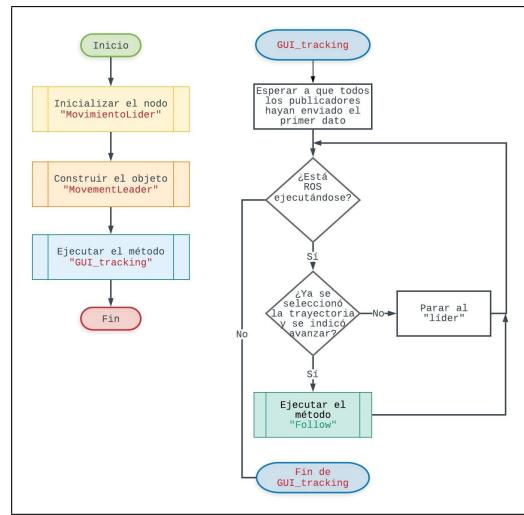


Figura 2.19: Diagrama de flujo del programa principal del nodo del líder.

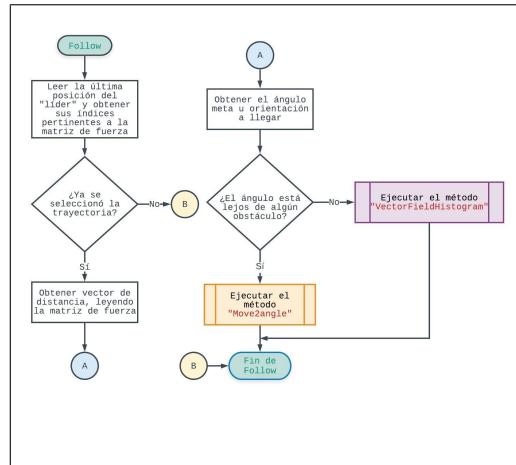


Figura 2.20: Diagrama de flujo del método “*Follow*”.



En la *Figura 2.20* se muestra el diagrama de flujo del método “*Follow*”, el cual tiene como propósito principal decidir si ejecuta el seguimiento de trayectoria o esquiva algún obstáculo cercano en caso de tenerlo. Adicionalmente incluye controles sobre la trayectoria seleccionada.

En la *Figura 2.21* se muestra el diagrama de flujo del método “*Vector Field Histogram*”, para su ejecución en el líder. Esta es la segunda parte del algoritmo donde ya se cuenta con el histograma generado. Este método tiene como función encontrar el ángulo más cercano a la trayectoria por donde pueda pasar el móvil sin chocar, y para ello encuentra el primer ángulo libre (más cercano a la trayectoria) donde se sabe que no hay obstáculo y se propone como dirección a llegar u objetivo.

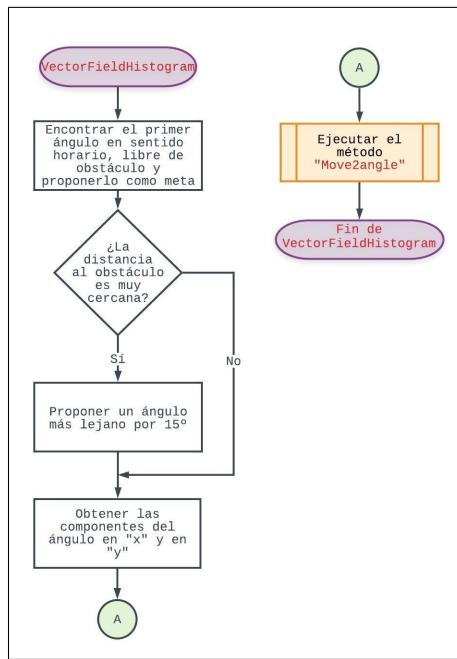


Figura 2.21: Diagrama de flujo del método “*VectorFieldHistogram*”.

En la *Figura 2.22* se muestra el diagrama de flujo del método “*Move2angle*”, el cual, a partir de la dirección a seguir, ya sea la que se da directamente del seguimiento de la trayectoria o a partir del VFH (en caso de tener un obstáculo cercano), orienta al robot líder al ángulo meta por medio del control de su velocidad angular. Cabe

mencionar que para realizar un mejor seguimiento cuando el móvil está desorientado por más de 45° , la velocidad lineal es cero para garantizar la rotación en su propio eje, lo cual reduce el error al tener una mejor maniobrabilidad.

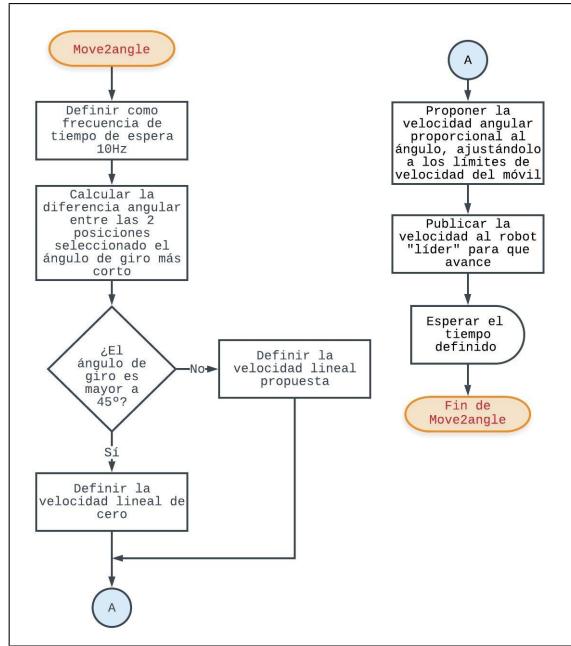


Figura 2.22: Diagrama de flujo del método “Move to angle”.

En la *Figura 2.23* se muestra los diagramas de flujo de los “Callbacks”, los cuales son métodos que interrumpen la ejecución del programa principal cada vez que se recibe un dato de los publicadores. El objetivo general de los callbacks es recibir y guardar datos como variables globales para que posteriormente puedan ser utilizados. En el caso específico del *histograma*, recibe un vector indicando si hay obstáculo o no, con el cual el método de “Follow” da la instrucción de seguir la trayectoria o esquivar el obstáculo. Para evitar que una variable no este actualizada, se prende una bandera que indica que ya fue recibido al menos un dato.

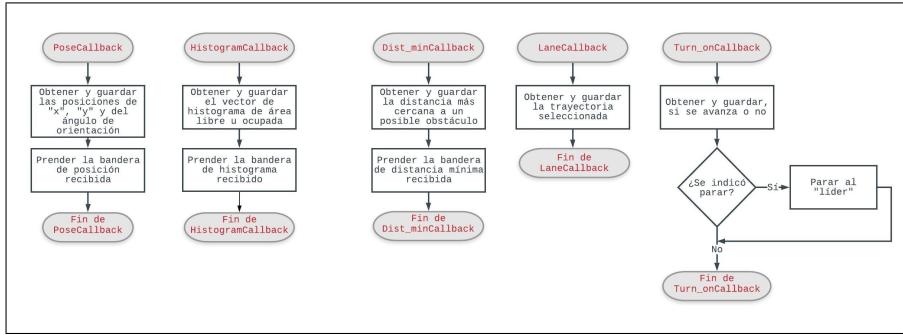


Figura 2.23: Diagrama de flujo de los “Callbacks”.

Este nodo es el encargado de suscribirse al algoritmo AMCL y con base a ello, reorientarse para mantener la trayectoria junto con la ayuda de la matriz de fuerza.

Esquema Líder - Seguidor El implementar uno de los esquemas de control líder - seguidor propuestos en el Marco de referencia, implica mantener una posición relativa al robot líder (d_1) pudiendo ocasionar colisiones entre ambos robots o entre los obstáculos. Para este caso en particular, el robot seguidor más que mantener una posición relativa al robot líder, debe mantenerse sobre la trayectoria dictada por el líder, es decir, para evitar que pueda existir una colisión y lograr que el robot seguidor se mantenga en la ruta, se debe mantener una velocidad lineal constante e igual para cada robot, enviando y almacenando los puntos ruta por los que el robot líder pase. Con dichos puntos de ruta, el robot seguidor usa el mismo principio que usa el robot líder para reorientarse.

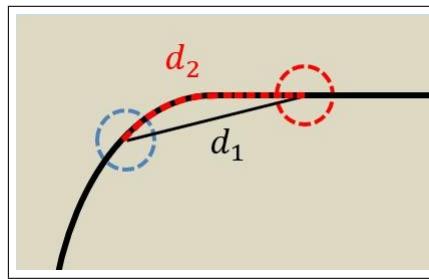


Figura 2.24: Representación de las distancias entre el líder y el seguidor.



En la *Figura 2.24* se tiene al robot líder (color rojo) y al robot seguidor (color azul), la curva de color negro representa un tramo de la trayectoria a seguir, la cual está formada por $P_n(x_n, y_n)$, siendo P_l la coordenada donde se localiza el robot líder y P_s donde se localiza el robot seguidor, definiendo d_1 y d_2 en las *ecuaciones 2.1* y *2.2*, de la siguiente manera.

$$d_1 = \sqrt{(x_l - x_n)^2 + (y_l - y_n)^2} \quad (2.1)$$

$$d_2 = \sum_{P_n=x_n,y_n}^{P_l=x_{l-1},y_{l-1}} = \sqrt{(x_{n+1} - x_n)^2 + (y_{n+1} - y_n)^2} \quad (2.2)$$

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 usado, se procede al diseño del algoritmo que cumpla lo propuesto “*Nodo Seguidor*”.

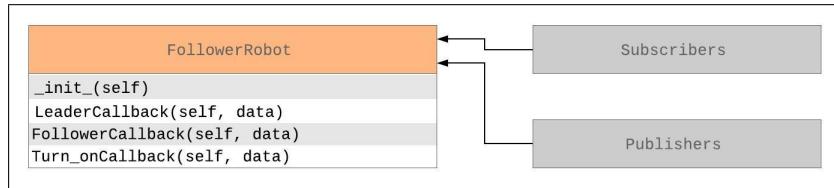


Figura 2.25: Diagrama UML del objeto *FollowerRobot* a implementar en el nodo seguidor.

Dicho nodo se suscribe a las posiciones publicadas por el líder y las posiciones del seguidor en la construcción del objeto “*FollowerRobot*”, como se muestra en la *Figura 2.26*. Cada vez que se recibe una nueva posición del líder, esta es almacenada en una lista de Python llamada *WPL* mostrada en la *Figura 2.27*. En la *Figura 2.27* se muestran los Callbacks del nodo del seguidor, obteniendo la posición del robot líder para despues almacenarla en la lista WPL (LeaderCallback) y en la instrucción de avanzar (Turn_onCallback).

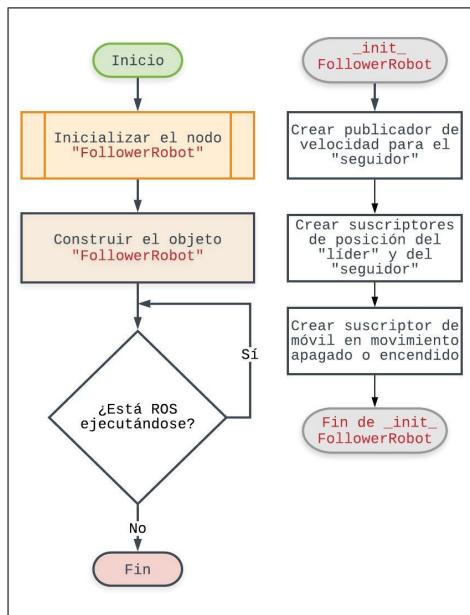


Figura 2.26: Diagrama de flujo del programa principal del nodo seguidor.

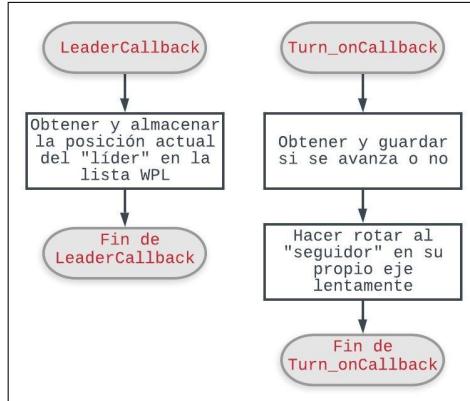


Figura 2.27: Diagrama de flujo de los Callbacks para obtención de datos.

El método *FollowerCallback* mostrado en la *Figura 2.28*, es utilizado por parte del robot seguidor para cubrir los mismos puntos alcanzados por parte del robot líder, reorientando al robot seguidor bajo el mismo lazo de control implementado en el líder, y en función de la posición y la lista con los puntos de referencia almacenados en la lista WPL.

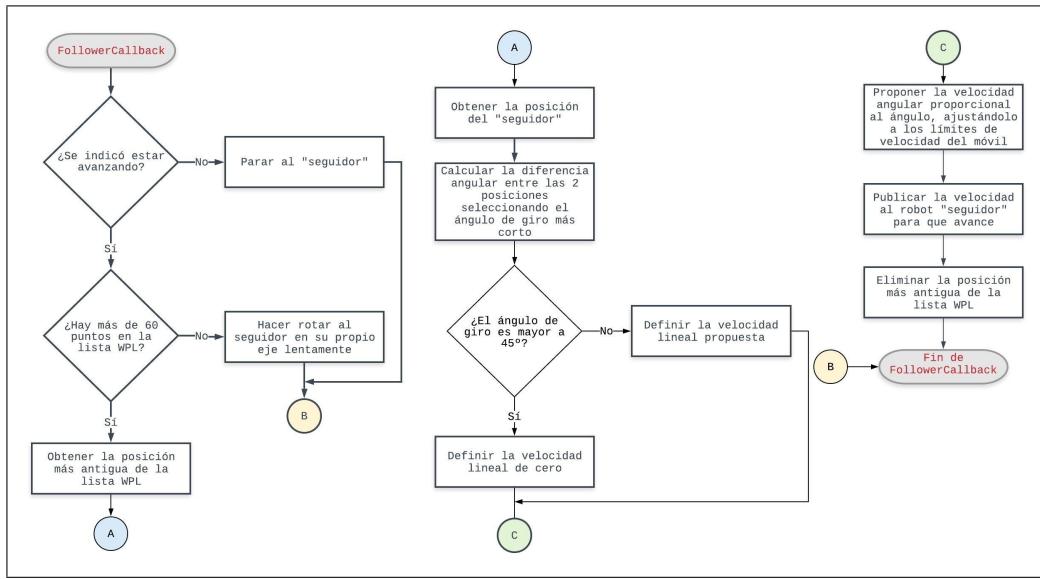


Figura 2.28: Diagrama de flujo del método FollowerCallback.

Evasión de obstáculos Para la evasión de obstáculos es necesario un método que se ajuste con la forma de operar de la Matriz de fuerza, por lo que se cuenta con dos posibilidades; *algoritmo por campos potenciales* y *VFH*. De Ribeiro en Evasión de Obstáculos se tiene un análisis de los pros y contras de los distintos algoritmos para la evasión de obstáculos [25]. Basado en ese estudio se elige el *VFH*, ya que es un algoritmo que requiere un menor costo computacional, no afecta de manera 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 implementar el *VFH* es dividiendo el algoritmo en dos nodos; el primer nodo es el de la *Distancia mínima* mostrado en la Figura 2.29, el cual tiene como único objetivo calcular y enviar la distancia entre el robot líder y cualquier obstáculo que tenga enfrente o al lado. El segundo es el nodo llamado *LaserVFH*, mostrado en la Figura 2.32, tiene como propósito generar un histograma polar binario en el cual se muestren los ángulos de donde hay y no hay obstáculo, dado un cierto umbral que después es mandado al nodo del robot líder. Como se puede observar en



el nodo del líder en la función “*Follow*”, en la primera parte se decide si aplica VFH o seguimiento de trayectoria en razón de la distancia mínima a cualquier obstáculo. En la segunda parte se aplica el VFH para encontrar el ángulo libre más cercano al ángulo dado por la trayectoria, gracias al histograma binario proporcionado por el nodo *LaserVFH*.



Figura 2.29: Diagrama UML para nodo de distancia mínima.

El nodo de distancia mínima en su función principal mostrada en la *Figura 2.30*, inicializa al nodo con su publicador de “distancia mínima” y además configura los límites de error debido a que en ciertas ocasiones el LIDAR entrega valores muy pequeños que no concuerdan con la distancia real entre los robots y los objetos muy cercanos, por lo cual la distancia debe ser mayor al radio de la estructura de los robots, ya que al no considerar este efecto, generaría que la distancia mínima siempre fuera de 0 y nunca se avanzaría. Por último, el constructor ejecuta el método “*Obtain_min_distance*” con el que se logra obtener la distancia mínima entre todos los ángulos frontales y laterales del robot.

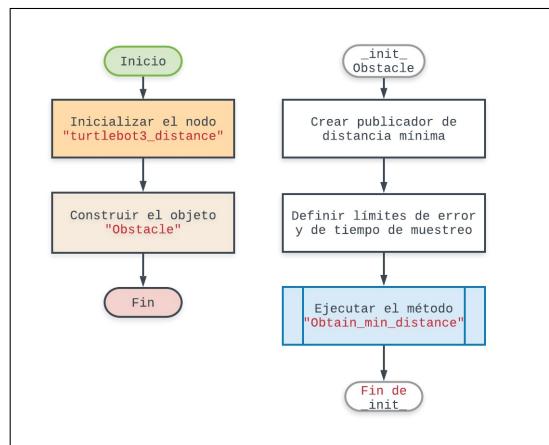


Figura 2.30: Diagrama de flujo del programa principal del nodo de distancia mínima.



Profundizando más en la obtención de la distancia mínima, en la *Figura 2.31* se muestra el diagrama de flujo de como se obtiene la distancia mínima, comenzando por verifica primero si ROS se está ejecutando de manera correcta para después proseguir con el programa, posteriormente se guarda en una lista todos los valores de las distancias de donde el robot líder pudiera chocar con un objeto (considerando que este siempre avanza hacia enfrente), y por último, se calcula el valor mínimo de la lista y este es publicado para que sea recibido en el nodo del robot líder.

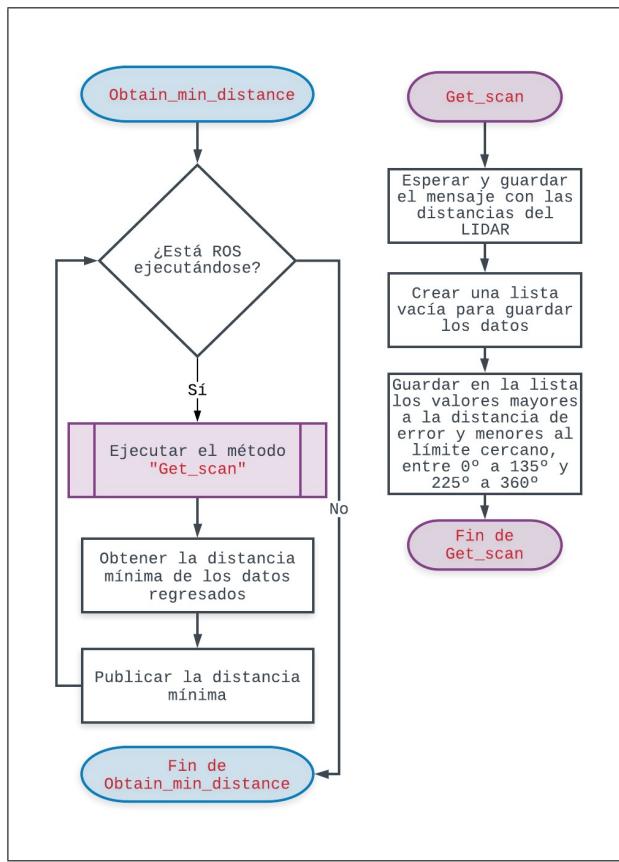


Figura 2.31: Diagrama de flujo método para obtener la distancia mínima al obstáculo.

Centrándose más en el diseño del algoritmo “*Vector Field Histogram*”, el algoritmo se distribuye en 2 partes; la primera parte consiste en la obtención del histograma binario a partir de las distancias dictadas por el LIDAR y su orientación, las cu-



les son mostradas en un diagrama UML y en un programa principal en las *Figuras 2.32 y 2.33*. La segunda parte consiste en usar el histograma binario para evadir el obstáculo, tal y como se explica en el nodo del líder.

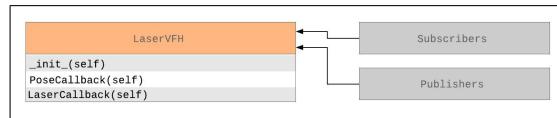


Figura 2.32: Diagrama UML para nodo LaserVFH.

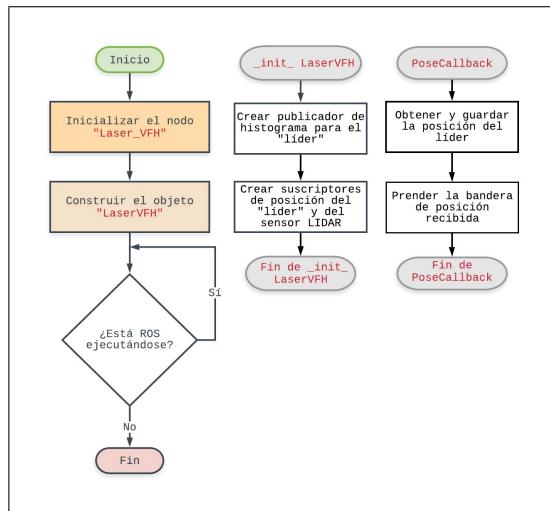


Figura 2.33: Diagrama de flujo del programa principal del nodo LaserVFH.

Para generar el histograma binario se aplica el proceso mostrado en la *Figura 2.34* donde, a partir del histograma de distancias proporcionado por el LIDAR y la orientación del líder, se genera un histograma binario base, acoplado al marco de referencia global sobre el cual está la trayectoria. Considerando así la orientación del líder para reorientar los ángulos del LIDAR que se obtienen conforme al marco de referencia del líder y un umbral de distancia al cual se le considera ángulo libre u ocupado. Posteriormente, con el fin de ensanchar este histograma binario (debido a que el móvil no es una partícula sino un cuerpo rígido), se calculan los puntos del histograma donde existen flancos de subida o flancos de bajada, obteniendo así los



valles y las crestas. A partir de estos ángulos se ensancha el histograma quitando “N” ángulos libres iniciando en cada cresta o valle (“N” se acostumbra desde 20 hasta 35). Por último, se envía el histograma ensanchado al nodo del líder.

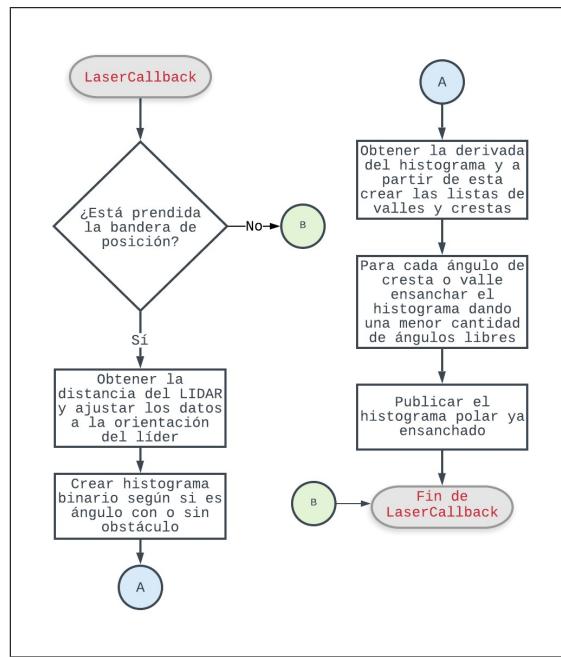


Figura 2.34: Diagrama de flujo para generar histograma binario.

Comunicación La comunicación se realiza por medio de WiFi entre los distintos dispositivos, es decir, el robot líder, el robot seguidor y la computadora de monitoreo, tienen una comunicación continua. Como se explicó en la parte del diseño conceptual; la arquitectura para el desarrollo de este proyecto es de tipo distribuida, ya que cada dispositivo antes mencionado, procesa distintos tipos de información y realiza 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 son 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*
- */tb3₀/Histogram*
- */tb3₀/amcl_pose*
- */tb3₀/cmd_vel*
- */tb3₀/odom*
- */tb3₀/scan*
- */tb3₁/amcl_pose*
- */tb3₁/cmd_vel*
- */tb3₁/odom*
- */tb3₁/scan*

Para poder enlazar estos mensajes de comunicación con los robots, es necesario establecer una comunicación WiFi entre los distintos robots y la computadora de monitoreo. A continuación se muestra como hacerlo.

Se obtiene la IP de la PC remota y se modifica en la computadora del TurtleBot3 Burger.

- Ingresar el siguiente comando en cada Turtlebot 3 burger y PC remota >>
`$nano /.bashrc`
- Modificar 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 2.35: Configuración de la IP para la comunicación vía WiFi [23].



Figura 2.36: Ejemplo de configuración WIFI [31].

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 mantener una velocidad lineal constante en todo momento, como se explica a continuación.

Lo primero es considerar la *velocidad lineal* a la que se desean que vayan los robots (V_k) y la velocidad máxima (V_{max}). Tomando la ecuación de la cinemática inversa, obtenemos las ecuaciones 2.3 y 2.4 para la velocidad en ambos motores de los robots.



$$\phi'_r = \frac{V_k + \theta' b}{r} \quad (2.3)$$

$$\phi'_l = \frac{V_k - \theta' b}{r} \quad (2.4)$$

Debido a la restricción de velocidad máxima de los motores (ϕ'_{max}), el máximo en la velocidad angular que se puede obtener, se representa en la *ecuación 2.5*.

$$\theta' = \frac{\phi'_{max} r - V_k}{b} \quad (2.5)$$

Las *ecuaciones 2.3, 2.4, 2.5*, son las implementadas por la librería que usa ROS para controlar a los robots.

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

Como se puede observar en la *Figura 2.9* de la página 33, la tercera área funcional a analizar es la de *Percepción*, la cual tiene la función de adquirir los datos propioceptivos y exteroceptivos de cada uno de los robots, es decir, la recabación de datos como la distancia, velocidad y posición de los robots, con el fin de poder ubicar a los robots, implementando odometría (*información proprioceptiva*), o llevando a cabo la recabación de la información del ambiente dentro del área de trabajo (*información exteroceptiva*).

Para la recabación de la *información proprioceptiva* es necesario un control preciso de los actuadores, con el objetivo de lograr orientar de manera correcta a los robots, y en conjunto con una buena lectura de los datos (obtenidos de la rotación de las ruedas), llevar a cabo una estimación más precisa de la posición de los robots durante la navegación.

2.2.3.1. Datos propioceptivos

Este tipo de datos son obtenidos a partir de variables internas de los robots, es decir, su medición es únicamente referente a variables internas cuantificables, como lo son; la medición de la batería, la medición de una señal proporcionada por un encoder en un actuador, etc. La principal ventaja que los sensores proprioceptivos tienen es que no dependen de estímulos externos por su característica de medir únicamente los datos o variables referentes a los estados internos del robot, 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*, pudiéndose estimar con un muestreo de los sensores, pero si nunca llegar a tener el valor verdadero.

2.2.3.2. 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 las que estas se mueven. 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, aunque los más comunes son los ópticos.

2.2.3.3. 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, el número de agujeros en el disco determina la precisión del encoder y este es colocado de modo tal que el eje del disco coincida con el eje del motor y el de la rueda (que normalmente son el mismo). 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, 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 entrada de reloj y la otra, 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).

2.2.3.4. Encoder óptico absoluto

Los encoders ópticos absolutos están formados por patrones codificados de zonas opacas y transparentes y suelen ser más apropiados para casos en los que se producen rotaciones más lentas, o poco frecuentes, como por ejemplo la averiguación de la rotación de un volante en un vehículo automatizado. El funcionamiento es similar al de los encoders relativos, pero en lugar de utilizarse un único led y un único foto receptor, se utilizan varios, tal y como se muestra en la *Figura 2.37*. Por medio de estos sensores se obtiene la posición absoluta sin necesidad de ningún cálculo como ocurre con el relativo, además de conocerse siempre su posición inicial, su tiempo de respuesta no es tan alto como en el relativo y su construcción es más compleja y al tenerse más señales.

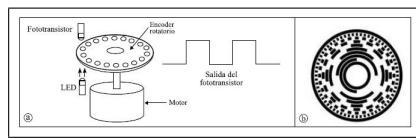


Figura 2.37: Funcionamiento del encoder absoluto.



2.2.3.5. Acelerómetros

Los acelerómetros son dispositivos que detectan cambios en la velocidad. La mayoría no están preparados para medir velocidad constante, sino que únicamente 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, 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 son 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.

2.2.3.6. Datos exteroceptivos

Este tipo de datos nos brindan información del medio exterior al recibir estímulos externos como temperatura, presión, acidez o alcalinidad, luz, etc. Los datos obtenidos por este tipo de sensores normalmente son interpretados por los robots como características del medio, donde a partir de estos, se elaboran mapas del entorno, conociendo los elementos que lo conforman.

2.2.3.7. Sensores láser

La palabra láser responde a las siglas en inglés *Light Amplification by Stimulated Emission of Radiation (LASER)*. Por lo general un sensor láser, es un dispositivo que emite un haz de luz por medio de un elemento transmisor y recibe devuelta ese



mismo haz de luz por medio de un elemento receptor, una vez que este es reflejado o interrumpido por un objeto, como se muestra a continuación.

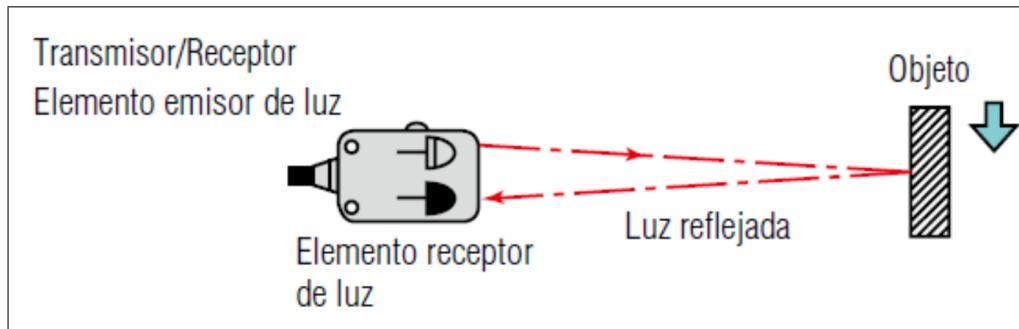


Figura 2.38: Funcionamiento de un sensor láser.

El sensor LIDAR es un tipo de sensor láser, que basa su funcionamiento en la medida del *tiempo de vuelo* como se observa en la *Figura 2.39*. 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 el tiempo de vuelta del pulso (tiempo de vuelo), se calcula fácilmente la distancia al objeto detectado. De este modo es posible medir la distancia en una sola dirección, pero gracias a que el sensor LIDAR dispone internamente de un espejo rotatorio, logra un efecto de barrido de 360°. Cabe mencionar 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.

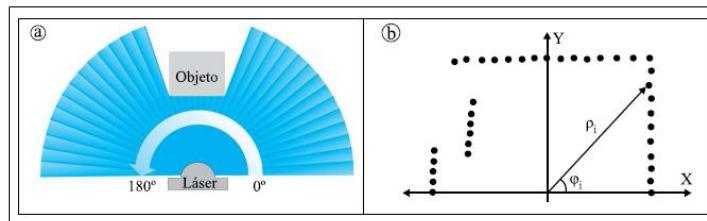


Figura 2.39: Funcionamiento del sensor LIDAR.

2.2.3.8. Sensor LIDAR LDS-01

Es un sensor láser 2D capaz de escanear en 360° las diferentes distancias, las cuales son utilizadas para llevar a cabo el *SLAM* para la localización y *VFH* para la esquivación obstáculos.

Elemento	Características
Voltaje de Operación	5V ± 5 %
Fuente de Luz	Diodo semiconductor de láser ($\lambda=785\text{nm}$)
Seguridad del láser	IEC60825-1 Clase 1
Consumo de corriente	400 mA o menor (Corriente pico 1A)
Detección de distancia	120mm – 3,500mm
Interfaz	<ul style="list-style-type: none"> ■ 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

Tabla 2.12: Características del sensor LIDAR.



2.2.3.9. Dimensiones

En las *Figuras 2.40* y *2.41* se muestran las dimensiones del sensor LIDAR.

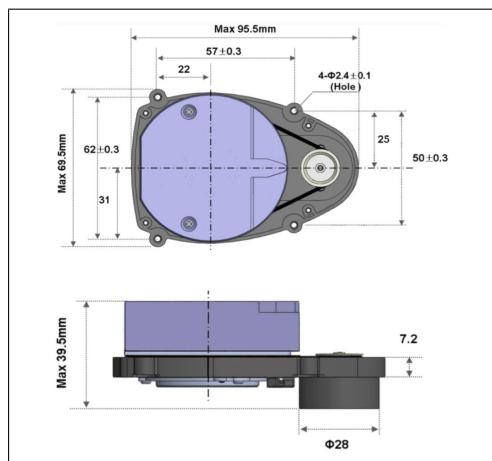


Figura 2.40: Dimensiones del sensor LIDAR. Vista superior y lateral.

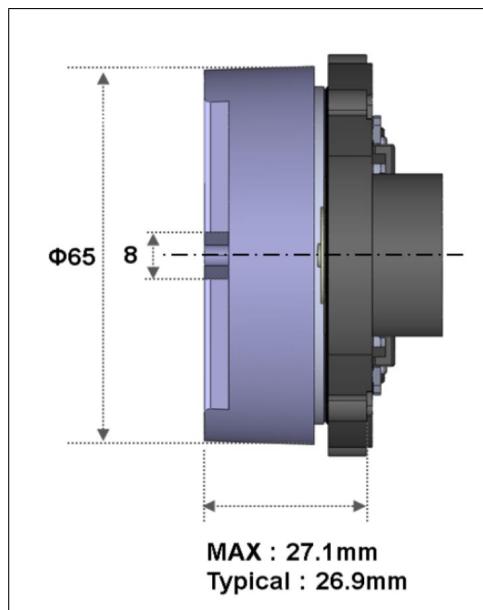


Figura 2.41: Vista Frontal del sensor LIDAR.



2.2.3.10. Acondicionamiento

Para conectar el sensor LIDAR a la tarjeta Raspberry Pi 3, se usa un convertidor de protocolo USB a protocolo UART, como se observa en la *Figura 2.42*, esto debido a que el LIDAR recibe comandos de 40 bytes con los cuales decide cada cuantos grados va a realizar un escaneo, así como también, la intensidad del láser. Al finalizar de recibir los 40 bytes se comprueba mediante una suma si los datos fueron correctos y se prosigue, en caso contrario, se vuelven a enviar los datos.

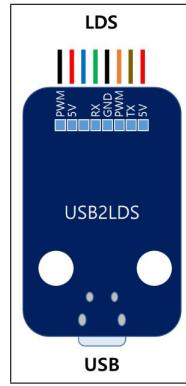


Figura 2.42: Pines del sensor LIDAR.

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 actuador encargado del giro del sensor LIDAR, así como también 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.

Tabla 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.

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

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

Tabla 2.15: Comandos para operación del LIDAR.

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

La cuarta área funcional denominada *Alimentación*, es la encargada de abastecer de energía por medio de una batería (*Figura 2.44*) a tres áreas fundamentales para los robots. Es necesario abastecer de energía a dos tarjetas de procesamiento, en segunda área funcional; Raspberry Pi 3 y la ARM Cortex M7. En la tercera área funcional, es necesario alimentar al sensor LIDAR encargado de la recabación de la información del ambiente, y en la quinta y última área funcional encargada del movimiento, es necesario abastecer de energía a dos actuadores Dynamixel (uno en cada llanta), responsables del movimiento en los robots y de la ejecución de toda la información antes proporcionada por las distintas áreas funcionales.



La responsable de abastecer energía a todos los elementos que conforman a cada una de las áreas funcionales antes mencionadas, es una batería de tipo LiPo (*Polímero de Litio*) ubicada en la primera capa del TurtleBot3, como se muestra en la *Figura 2.43*.

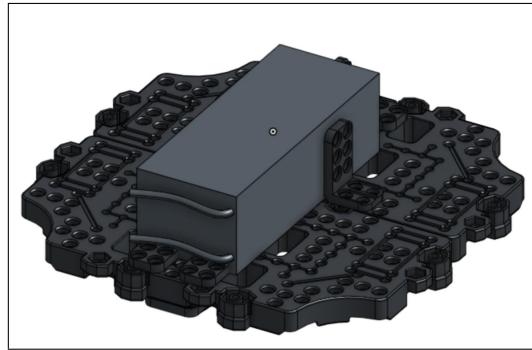


Figura 2.43: Ubicación de la batería de alimentación. Primera capa del Turtlebot3.

La batería tiene una capacidad nominal de 1800 mAh con una sola salida de 11.1V que abastece de energía a todo el robot y una entrada como se muestra en la *Figura 2.44*, encargada de cargar a la batería.



Figura 2.44: Batería LiPo, con su entrada y salida [33].



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)

Tabla 2.16: Especificaciones técnicas de la batería. [12]

2.2.4.2. Ensamble de la batería LiPo con la estructura

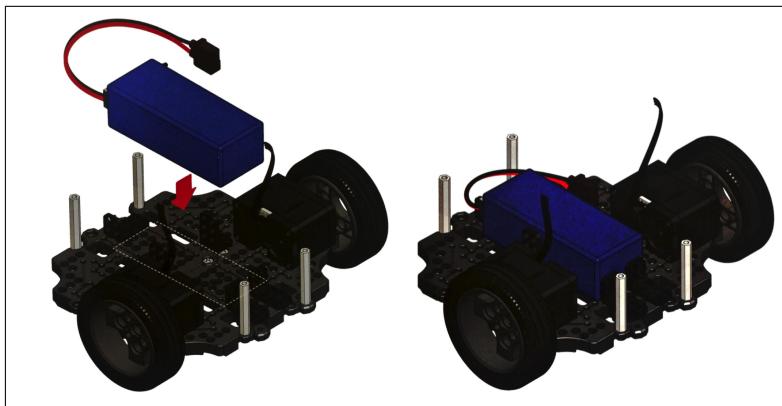


Figura 2.45: Ensamble de la batería LiPo en la primera capa del Turtlebot3.

Como ya se ha mencionado, la batería LiPo va ensamblada en el primer piso de cada uno de los robots y cuenta con un cuatro barras que limitan el movimiento de misma. A los costados se encuentran los actuadores de los robots y en la parte de arriba se encuentran las tarjetas de procesamiento y el sensor LIDAR. En la sección de *Integración del Sistema* se habla más a detalle.

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 el robot ejecute las instrucciones proporcionadas por las 4 funciones previas que engloban el proyecto. En la *Figura 2.46* se muestra el proceso de ensamble para cada llanta.



Figura 2.46: Ensamble del rin y la llanta de los robots.

El sistema de locomoción se encuentra en la primera capa dentro de la estructura de los 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 2.47* y *2.48*.



Figura 2.47: Estructura de la locomoción de los robots.



Figura 2.48: Ensamble de los actuadores con la estructura.

2.2.5.1. Actuadores Dynamixel

Para el control de la posición y velocidad de los robots, se utilizan dos motores Dynamixel XL430-W250-T por cada robot, los cuales cuentan con un controlador interno capaz de ajustarse hasta en 6 modos distintos. La información o instrucciones se transmiten por medio del protocolo UART a la Open CR, encargada de mandar información a los motores [28].



2.2.5.2. Análisis interno del motor

El actuador básicamente está compuesto por un motor de CD, una tarjeta controladora y un encoder, como se muestra en la *Figura 2.49*. La tarjeta controladora se encarga de la recepción y ejecución de las instrucciones, así como también de leer los valores del sensor, mientras que el encoder es el sensor encargado de determinar la posición.

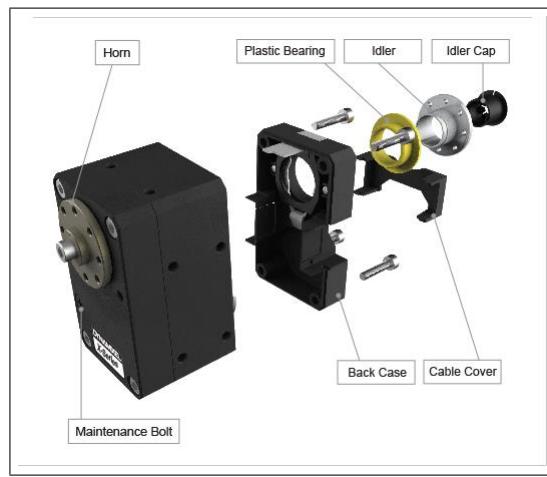


Figura 2.49: Explosión del motor del Dynamixel.

Los tipos de control que puede realizar la tarjeta controladora del Dynamixel 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 2.50: Características del actuador.

Los actuadores Dynamixel son uno de los mejores y más avanzados motores económicos y de alto rendimiento, también llamados actuadores inteligentes. Ofrecen la posibilidad de programar entre 50 y 57 comandos, permitiendo definir el comportamiento del actuador de una mejor y otorgando funciones especiales como las mostradas en la *Figura 2.50*, que en comparación con un servomotor típico, este solo interpreta la orden de “ángulo objetivo” proporcionado por una señal PWM.

En las *Figuras 2.51* y *2.52* se muestran los motores ya ensamblados para formar la configuración diferencial.

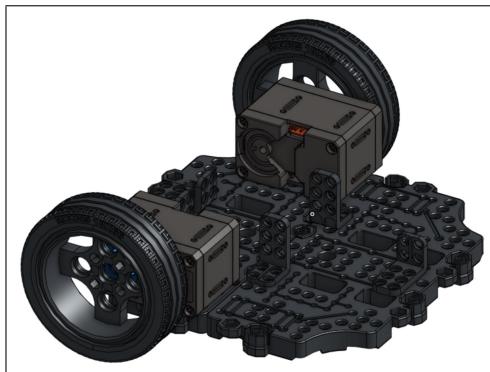


Figura 2.51: Sistema de locomoción en conjunto con los actuadores Dynamixel XL 430 - W250.

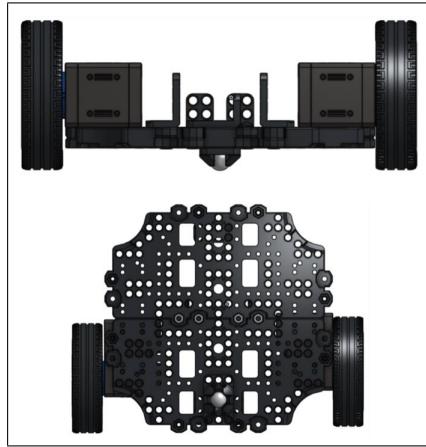


Figura 2.52: Sistema de locomoción en conjunto con los actuadores Dynamixel XL 430 - W250. Vista frontal e inferior.

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, haciéndolo sumamente útil para la odometría. Entre la información que pueden proporcionar los actuadores Dynamixel se encuentra leer la posición actual del motor, la velocidad, la temperatura interna, el par y la tensión de alimentación.

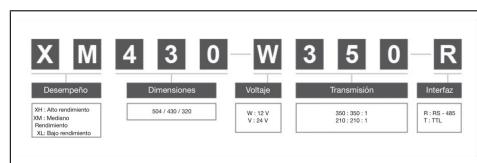


Figura 2.53: Nomenclatura de los actuadores Dynamixel [28].

Los actuadores Dynamixel XL 430 - W250, son los actuadores del más bajo desempeño dentro de las tres posibles clasificaciones de los actuadores (*Figura 2.53*), y cuyas especificaciones y características técnicas se mencionan en la *Tabla 2.17*, en conjunto con sus curvas características mostradas en la *Figura 2.54*.



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]	<ul style="list-style-type: none"> ■ 9.0 ■ 11.1 ■ 12.0
Torque	<ul style="list-style-type: none"> ■ 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)	<ul style="list-style-type: none"> ■ 47 [rev/min] (a 9.0 [V]) ■ 57 [rev/min] (a 11.1 [V]) ■ 61 [rev/min] (a 12.0 [V])

Tabla 2.17: Características técnicas del actuador Dynamixel XL 430 W250 [27].

Elemento	Características
Algoritmo de control	<ul style="list-style-type: none"> ▪ PID
Grados de precisión	<ul style="list-style-type: none"> ▪ 0.088°
MCU	<ul style="list-style-type: none"> ▪ ST CORTEX M3 32 Bits
Sensor de posición	<ul style="list-style-type: none"> ▪ Enconder sin contacto (12Bit, 360) por AMS
Resolución	<ul style="list-style-type: none"> ▪ 0.088 x 4.096 pasos
Rango de operación	<ul style="list-style-type: none"> ▪ Modo control de velocidad: Encendido sin fin. ▪ Modo control de posición: 0° a 360°. ▪ Modo control extendido: 256 revoluciones. ▪ Modo control PWM: Encendido sin fin.
Voltaje de salida [V]	<ul style="list-style-type: none"> ▪ 6.5 a 12.0V (Voltaje recomendado: 11.1 V)

Tabla 2.18: Características técnicas del actuador Dynamixel XL 430 W250 [27].

Elemento	Características
Temperatura de operación	5°C a 72°C
Señales de control	<ul style="list-style-type: none"> ■ Paquete digital
Tipo de protocolo	<ul style="list-style-type: none"> ■ Comunicación serie asíncrona semidúplex (8 bits, sin paridad)
Transmisión de datos	<ul style="list-style-type: none"> ■ 9600 bps a 4.5 Mbps
Retroalimentación	<ul style="list-style-type: none"> ■ Posición ■ Velocidad ■ Carga ■ Trayectoria ■ Temperatura ■ Voltaje de entrada
Material	<ul style="list-style-type: none"> ■ Carcasa: Plástico. ■ Engranes: Plástico.

Tabla 2.19: Características técnicas del actuador Dynamixel XL 430 W250 [27].

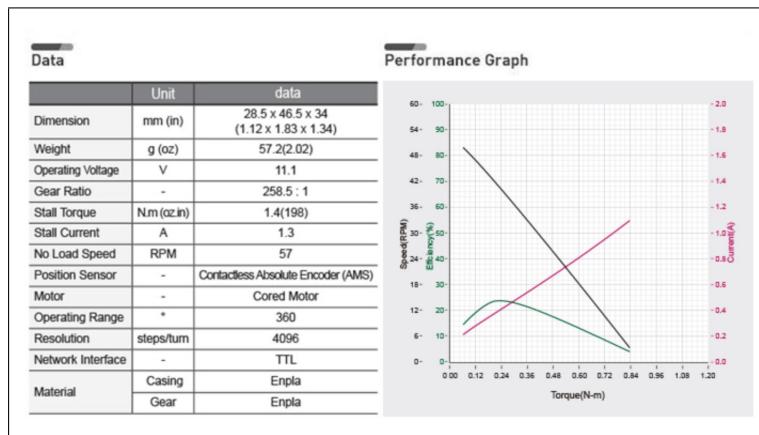


Figura 2.54: Gráfica de funcionamiento del motor DYNAMIXEL [28].

Entre cualidades del motor Dynamixel XL 430 W250 actuador tenemos:

1. Torque mejorado y diseño compacto.
2. 6 modos de funcionamiento.
3. Control perfil para la planificación de movimientos.
4. Eficiencia energética con tiempo mejorado de operación.

En cuanto a la programación de los actuadores Dynamixel, estos pueden ser programados mediante los siguientes lenguajes de programación.

- OpenCM ID.
- C/C++, Labview, Matlab, Visual Basic.
- Software exclusivo [Dynamixel Workbench].

Basándose en este último, el Dynamixel Workbench es un metapquete 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. 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 2.55*.

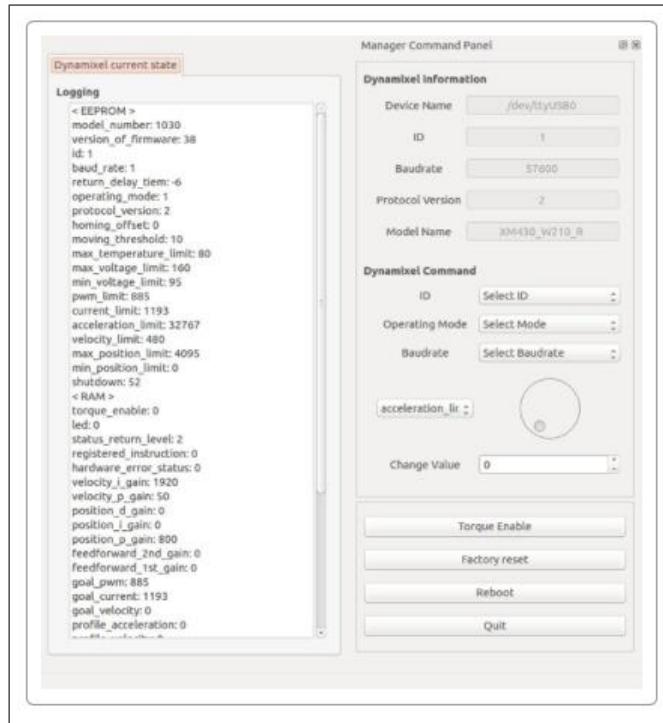


Figura 2.55: Interfaz de programación de los actuadores Dynamixel.



2.2.6. Integración del Sistema

Una vez conociendo todos los elementos que conforman a los robots móviles, se da 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 LiPo que suministra energía a todo el robot como se muestra en la *Figura 2.56*.

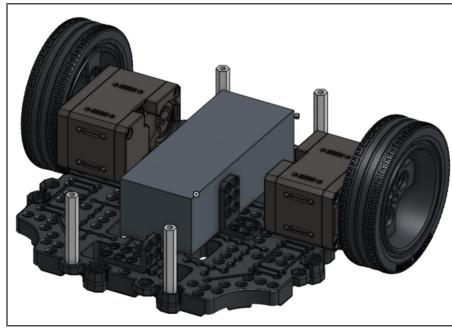


Figura 2.56: Capa 1. Implementación de la capa con los actuadores y la batería LiPo.

La segunda capa como se muestra en la *Figura 2.57* es la destinada para el montaje la tarjeta OpenCR, 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.

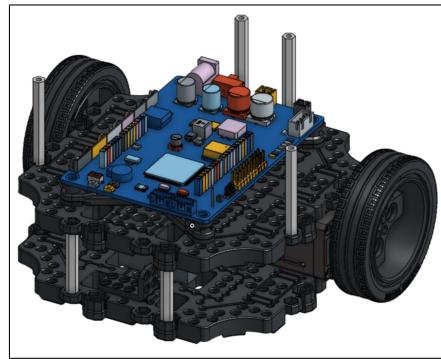


Figura 2.57: Capa 2. Implementación de la capa 1, capa 2 y tarjeta Open CR.



En la tercera capa como se muestra en la *Figura 2.58* 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.

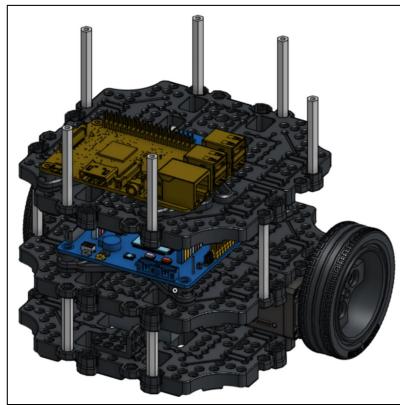


Figura 2.58: Capa 3. Implementación de la capa 1, capa 2, capa 3 y tarjeta Raspberry Pi 3.

Ya explicadas las distintas capas que conforman todo el sistema del robot, se muestra ensamblaje del robot en las *Figuras 2.59* y *2.60* utilizando un total de 193 piezas en la parte estructural y 12 piezas que involucran la parte electrónica.

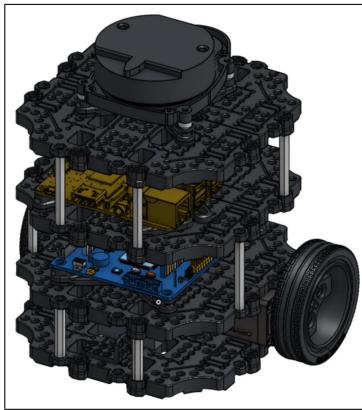


Figura 2.59: Vista isométrica del robot con todas las capas ensambladas.

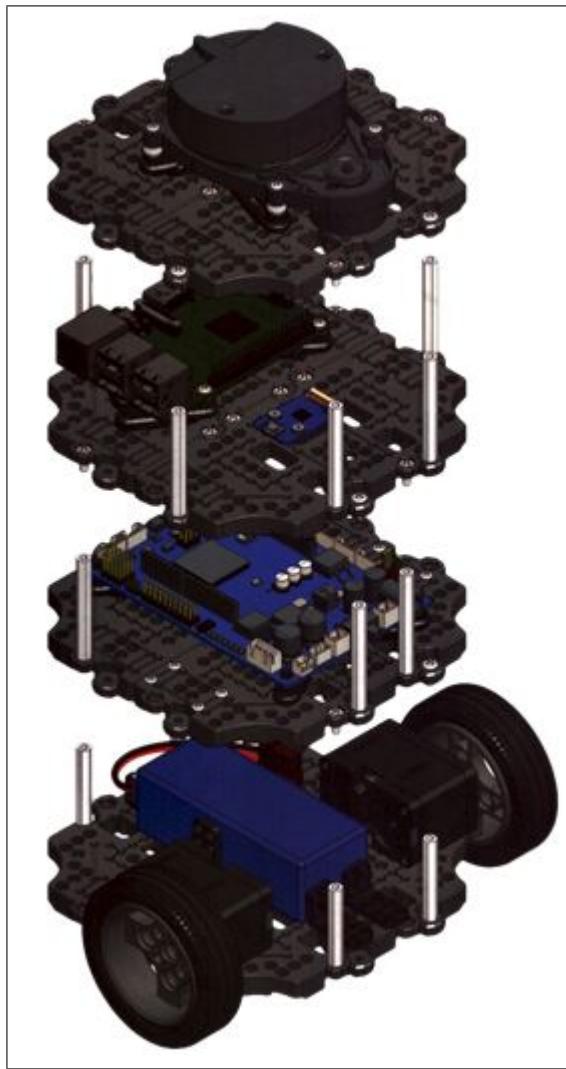


Figura 2.60: Explosión y distribución de los componentes que conforman al robot.

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 cumple una función completa, sino que dicha programa está dividido en distintos nodos. La manera en que los diversos nodos se comunican es mediante el uso de tópicos, los cuales también son enlistados 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 TurtleBot3 burger. Se proponen, para el líder y seguidor la interconexión de los siguientes nodos mostrados en las *Figuras 2.61* y *2.62*.

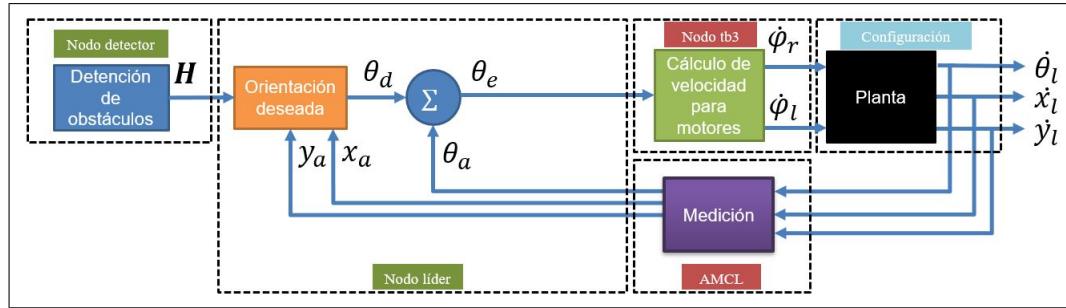


Figura 2.61: Esquema general de funcionamiento del sistema de control para el seguimiento de trayectorias y evasión de obstáculos.

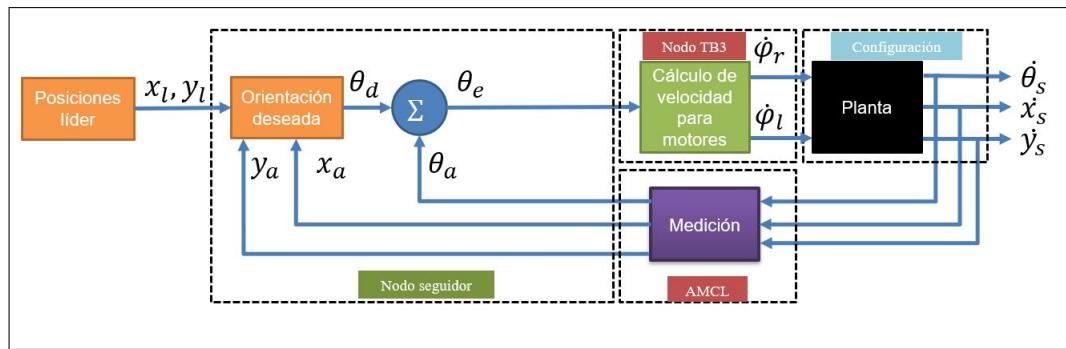


Figura 2.62: Esquema general de funcionamiento del sistema de control para el seguidor.

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 ya procesada, por lo que estos de manera natural pueden y están hechos para trabajar en conjunto.



Para mandar a llamar a todos los nodos, se hace uso de otro tipo de archivo de ROS con extensión .launch, el cual permite mandar a llamar a los nodos de manera automática. Para lograr la integración que se propone en los diagramas de las *Figuras 2.61* y *2.62*, se debe primero mandar a llamar el nodo que inicializa a los motores y el nodo que inicializa el sensor LIDAR. Posteriormente se manda a llamar el mapa, para la localización. Una vez inicializado el mapa, se manda a llamar a los nodos líder y seguidor, en los cuales se ejecuta la parte medular del proyecto para finalmente mandar a llamar por medio de otro launch, al localizador del líder y al localizador del seguidor.

Los diagramas antes presentados muestran la integración de los nodos, sin embargo, dentro de cada nodo existen tópicos (variables compartidas) los cuales también forman parte de la integración de dichos códigos. El diagrama de la *Figura 2.63* muestra como esos tópicos se comunican y van transformando la información, cada vez que pasa por algún nodo.

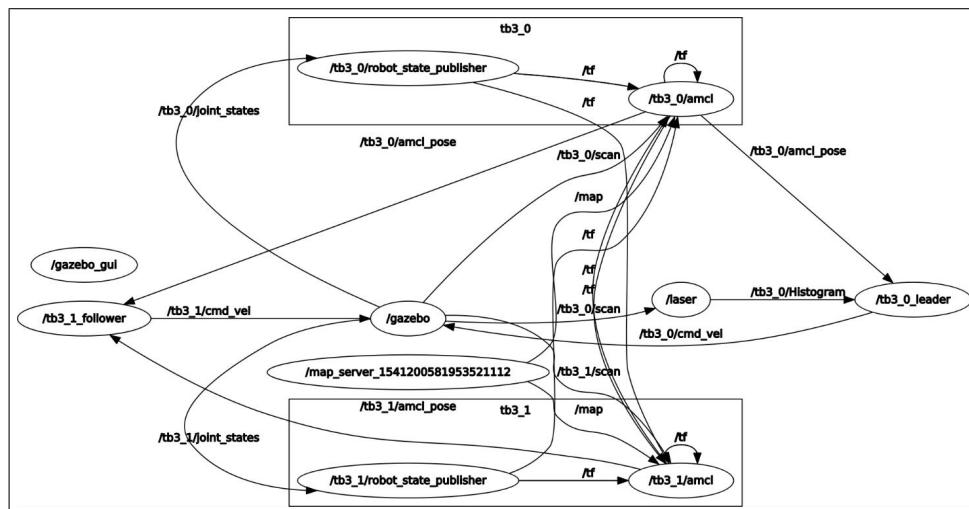


Figura 2.63: Comunicación entre tópicos de los distintos nodos.



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 selecciona y envia datos a través de una computadora.

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 2.64*.

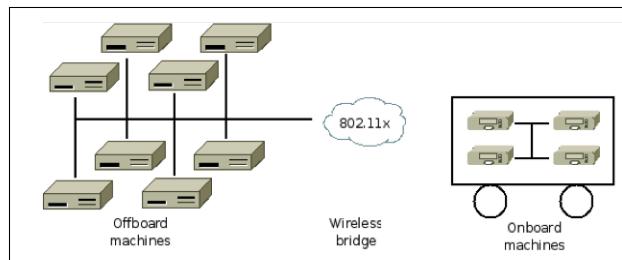


Figura 2.64: Conexión típica de ROS.

Para el desarrollo de la interfaz gráfica es importante entender como trabaja este tipo de comunicación, ya que la interfaz gráfica que se implementa desde una computadora funciona como anfitrión (teniendo un único anfitrión), que envia 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 2.65*.

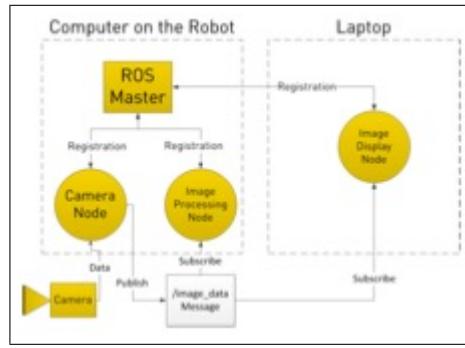


Figura 2.65: Ejemplo de comunicación de ROS con agentes externos.

La *Figura 2.64*, 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 la *Figura 2.65*, la recabación de los datos exteroceptivos de cada robot, es mediante un sensor LIDAR.

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 TurtleBot3 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 TurtleBot3 burger. Una vez hechos los programas, fueron implementados en un ambiente de simulación llamado *Gazebo* incluyendo allí el modelo 3D del TurtleBot3 burger, con sus características físicas y restricciones mecánicas como la 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.

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, mostradas en las *Figuras 2.66 y 2.67*, 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 2.68 y 2.69*, estas figuras también tienen sus ejes coordinados en metros. Como se puede observar de las *gráficas 2.68 y 2.69* los vectores generados siguen las trayectorias propuestas cuando están dentro de la trayectoria y se orientan al punto más cercano cuando están fuera de trayectoria, por lo cual se toma como válido el algoritmo.

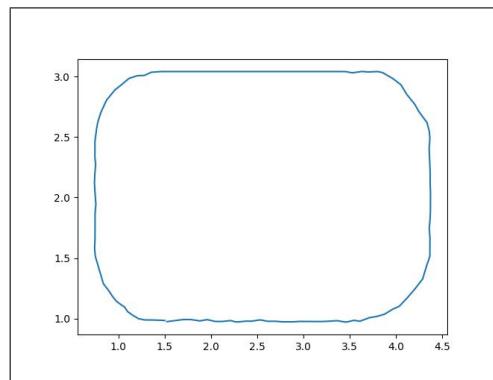


Figura 2.66: Trayectoria propuesta 1 (unidades en metros).

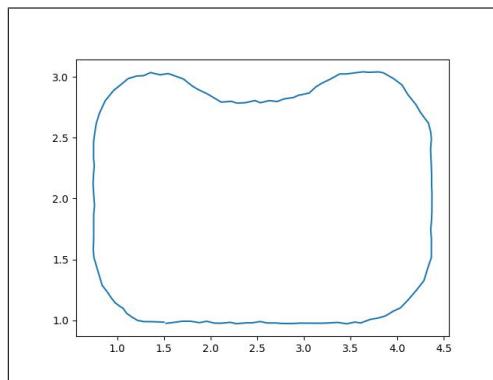


Figura 2.67: Trayectoria propuesta 2 (unidades en metros).

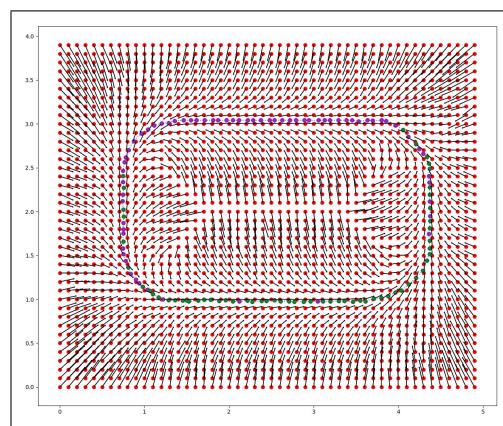


Figura 2.68: Matriz de fuerza para trayectoria propuesta 1 (unidades en metros).

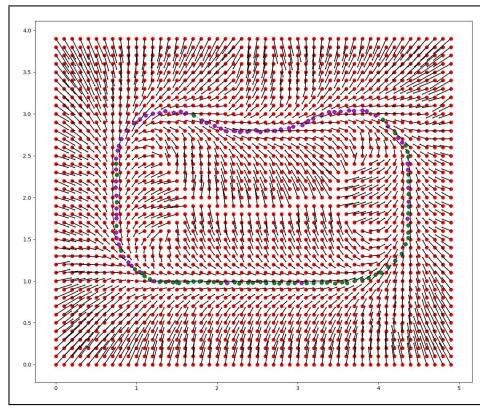


Figura 2.69: Matriz de fuerza para trayectoria propuesta 2 (unidades en metros).

A continuación se valida que el TurtleBot3 burger en el ambiente virtual sea capaz de seguir la trayectoria de la *Figura 2.66*. Recordando que la simulación es realizada en Gazebo y los datos son grabados para posteriormente ser 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 2.71*.



Figura 2.70: Resultados extraídos de la simulación de seguimiento de trayectoria (unidades en metros).

La *Figura 2.71* 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 2.71* y *2.66*, a simple vista se puede observar que los puntos representados en la *Figura 2.71* se asemejan a la trayectoria propuesta en la *Figura 2.66*, 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 2.71* y *2.72*, 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.

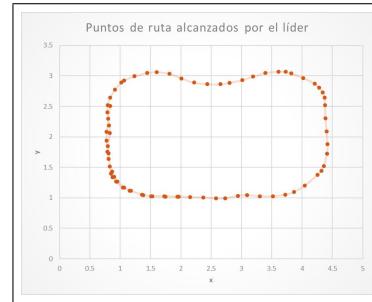


Figura 2.71: Resultados extraídos de la simulación del líder (unidades en metros).



Figura 2.72: 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 2.73*, 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 2.74*.

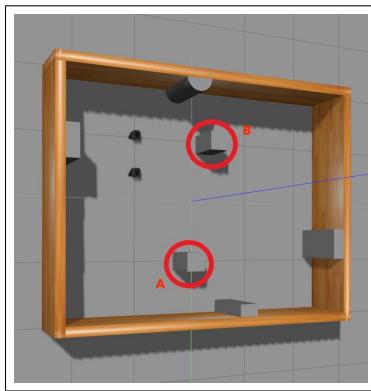


Figura 2.73: Obstáculos para la validación del algoritmo propuesto.

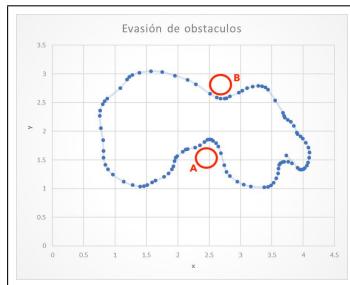


Figura 2.74: Puntos de ruta alcanzados por el líder durante la evasión de obstáculos (unidades en metros).

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 esta área funcional estuvo relacionada con el armado del robot, que como ya se había mencionado anteriormente en capítulos pasados, está área funcional tenía como requerimiento albergar a todas las demás funcionales, lo cual no presentó mayor dificultad porque cada robot ya viene diseñado cumpliendo con el objetivo de centrarse en armar y aplicar el esquema líder-seguidor, sin realizar un diseño exhaustivo del sistema mecánico.

3.1.1. Recomendaciones a considerar

Para el armado de estructura y de las partes electrónicas se siguieron las instrucciones tal y como se indican en el manual de ensamble del Turtlebot 3 burger [30], con algunos consejos que a continuación se mencionan.

- Antes de armar el Turtlebot 3 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 (al menos) 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.

3.1.2. Armado de los robots móviles



Figura 3.1: Caja con los diferentes componentes del robot.



Los componentes del Turtlebot 3 burger vienen en 4 cajas enumeradas y mostradas en la Figura 3.1, cada caja cuenta 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 batería 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 Tarjeta 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

3.1 Implementación del Área Funcional Estructura

898



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 batería 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.

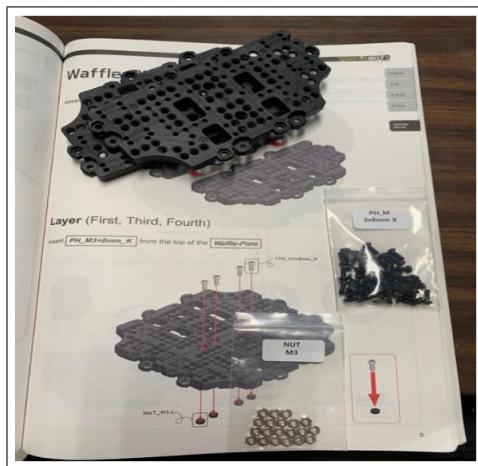


Figura 3.7: Unión de los pisos del robot.

3.1 Implementación del Área Funcional Estructura

89%



Figura 3.8: Piezas Ensambladas.

Los problemas que se presentaron en esta parte 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.

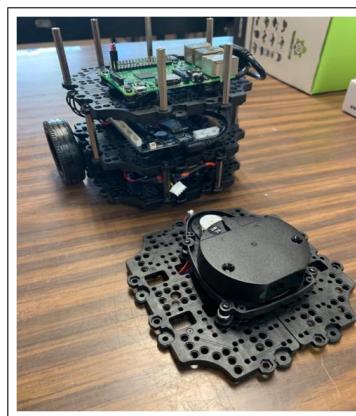


Figura 3.9: Ensamble del robot.



3.1.3. 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 de los pisos del Turtlebot 3 burger así como el orden en el que fueron ensamblados.

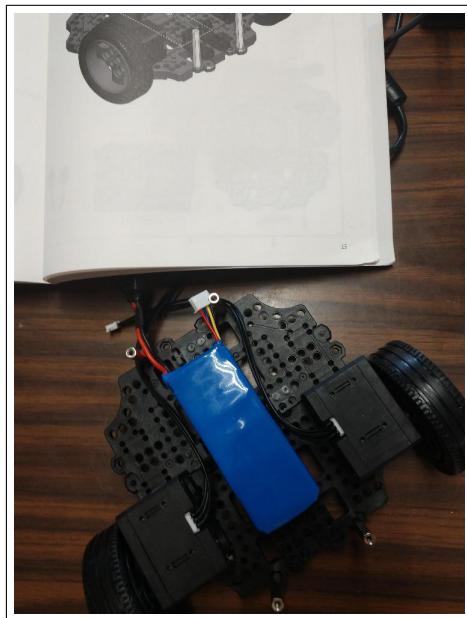


Figura 3.10: Ensamble del primer piso.

3.1 Implementación del Área Funcional Estructura

89%



Figura 3.11: Ensamble del segundo piso.



Figura 3.12: Ensamble del tercer piso.

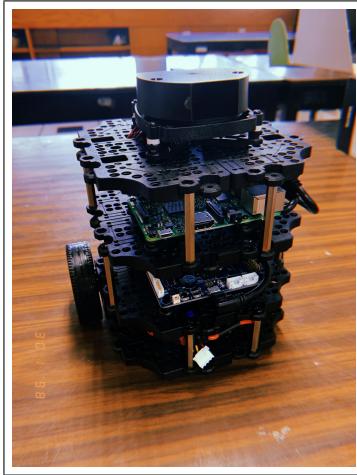


Figura 3.13: Ensamble final del robot.

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 bien encajados y posicionados dificultándose porque algunas piezas eran de ensamble múltiple por lo cual si una se movía se tenía que reiniciar el proceso y como eran muy pequeñas eran difíciles de insertar y se perdían fácilmente. Fue por estas razones que la tarea de ensamble duró aproximadamente 5 horas de trabajo continuo para cada Turtlebot 3 burger. Como se muestra en la Figura 3.14 el sensor LIDAR tuvo que ser bien posicionado antes de terminar de montarlo debido a su conexión con el piso inferior inmediato.

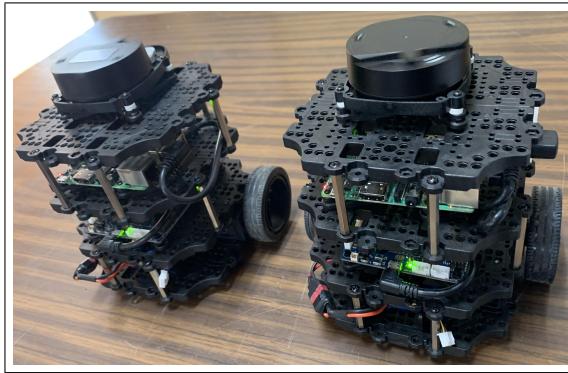


Figura 3.14: Robot líder y robot seguidor.

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 la Raspberry y la 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 del sistema operativo Raspbian a la Raspberry Pi 3 también nos proporcionó herramientas de validación para saber el estado de los elementos conectados a los móviles.

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 controlador PID encargado de mantener estable las velocidades de los robots no fue necesario realizar 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.



En la Figura 3.15 se muestra el ensamblaje mínimo para poder verificar el área funcional de movimiento usando los botones de la tarjeta OpenCR para controlar el movimiento del móvil.



Figura 3.15: Implementación del área funcional movimiento.

3.3. Implementación del Área Funcional Percepción

En la implementación de la tercera área funcional; percepción se utilizan 2 sensores el sensor LIDAR y el 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), calculando así las distancias mínimas de 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 el 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 Turtlebot 3 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 está trabajando correctamente.

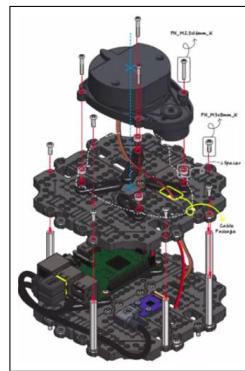


Figura 3.16: Montaje del sensor LIDAR en la estructura.

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 considerando que el sensor LIDAR tenga su origen en 0 grados en la parte frontal del móvil, si se no respetara esta 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 está apuntando el del sensor LIDAR cuando se inicia el escaneo.



Figura 3.17: Sensor LIDAR del Turtlebot 3 burger.

Para comprobar que el sensor está 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 Turtlebot 3 burger y la PC principal, y estando en la misma red con los mismos parámetros se obtuvieron los resultados mostrados en la Figura 3.18, donde los puntos en amarillo son las distancias escaneadas por el LIDAR.

3.3 Implementación del Área Funcional Percepción

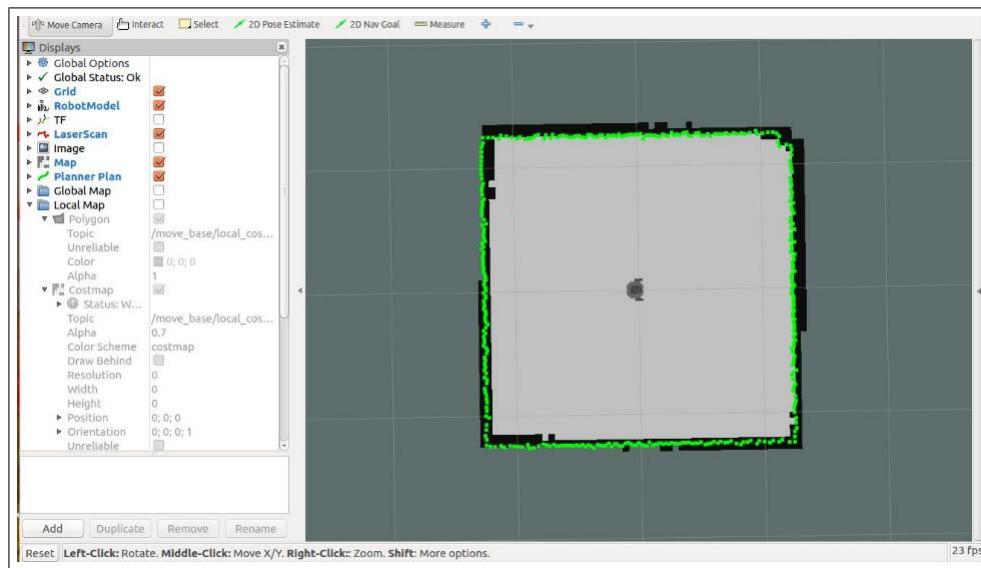


Figura 3.18: Distancias del LIDAR en RVIZ

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 la ejecución de los comandos ha sido incorrecta.

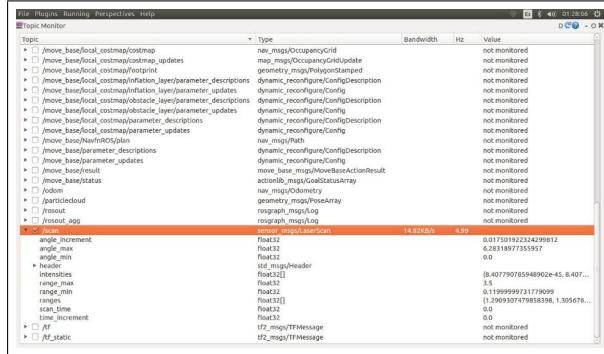


Figura 3.19: Visualización en rqt de los valores del sensor LIDAR.

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 encoder lleva a cabo la tarea en conjunto con el microcontrolador del motor DYNAMIXEL, de estimar el 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.

3.3 Implementación del Área Funcional Percepción

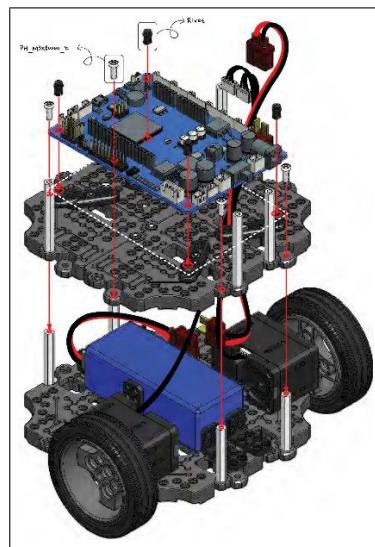


Figura 3.21: Ensamble de los motores, OpenCR y batería LiPo.

Una vez ensamblada esa capa se procedió a verificar el funcionamiento de los motores así como el funcionamiento de los encoders 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. En la Figura ?? se muestran los botones que se usan para probar el movimiento del Turtlebot 3 burger usando la OpenCR.

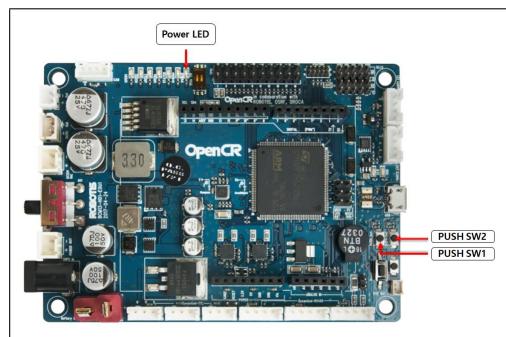


Figura 3.22: Localización de los botones de prueba en la tarjeta OpenCR.



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 de 11.1V 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 Turtlebot 3 burger se encuentran en su manual online, a continuación, se presentan algunas recomendaciones 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 [18]. La mejor forma de instalar ROS sobre la computadora remota es con el método que propone el manual online de los Turtlebot 3 burger, todo el proceso es automático, ya que a diferencia de los métodos convencionales donde es necesario ejecutar una serie



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 [24], además en el mismo manual online, 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 la tarjeta OpenCR, igual existen dos formas de configurarla, la primera por medio de la computadora remota y sirve cuando se está ensamblando por primera vez el Turtlebot 3 burger y la segunda directo de la Raspberry Pi 3, la segunda opción es la más sencilla cuando ya se tiene ensamblado el Turtlebot 3 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 serían 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 medidas 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 del Turtlebot 3 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 del 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 están presentes en el manual y se ejecutan sobre la terminal de la computadora remota son:

- rosrun turtlebot3_slam turtlebot3_slam.launch slam_methods:=gmapping
- rosrun 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.

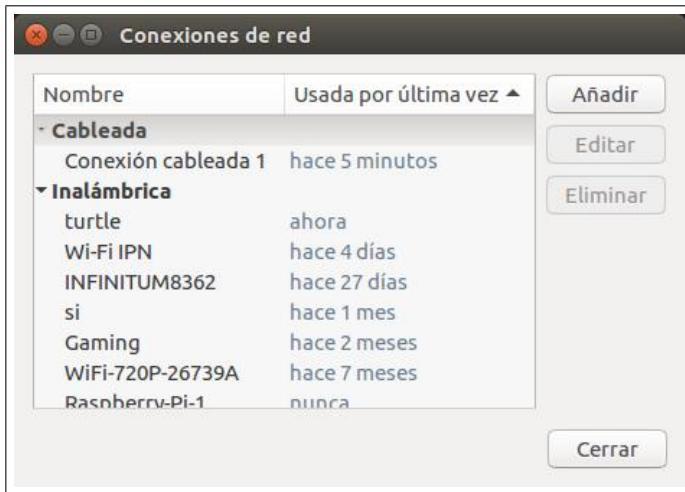


Figura 3.24: Creación de una nueva red en Ubuntu.

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



Figura 3.25: Elegir tipo de conexión.

3.5 Implementación del Área Funcional Procesamiento



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

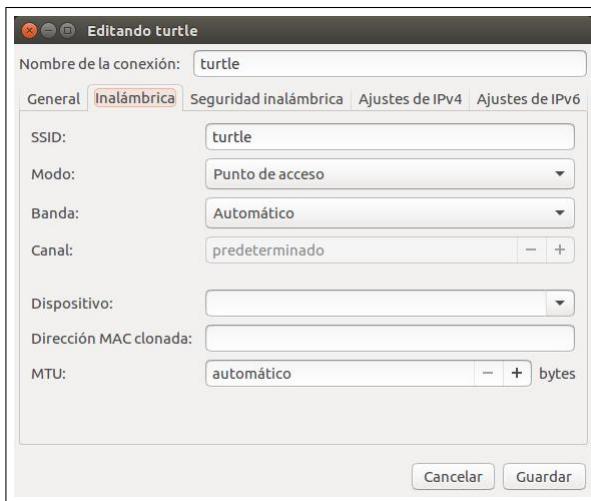


Figura 3.26: Configuración del Modo y Nombre.

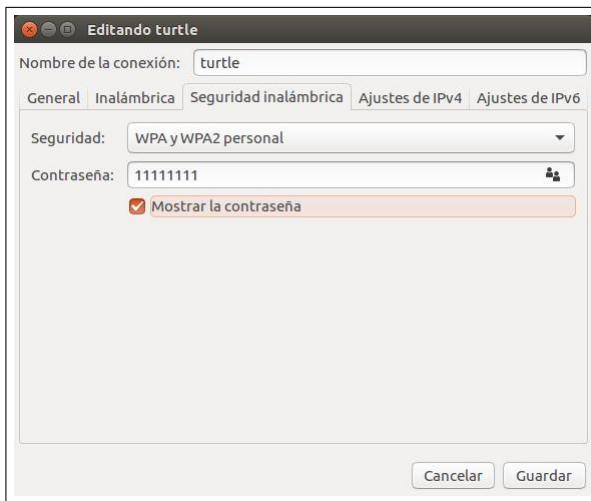


Figura 3.27: Configuración de la seguridad.

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 Turtlebot 3 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 Turtlebot 3 burger.

Además con apoyo de RQt, se puede visualizar el valor actual de algunos tópicos, en este caso se eligió el estado de la batería como se muestra en la Figura 3.29.

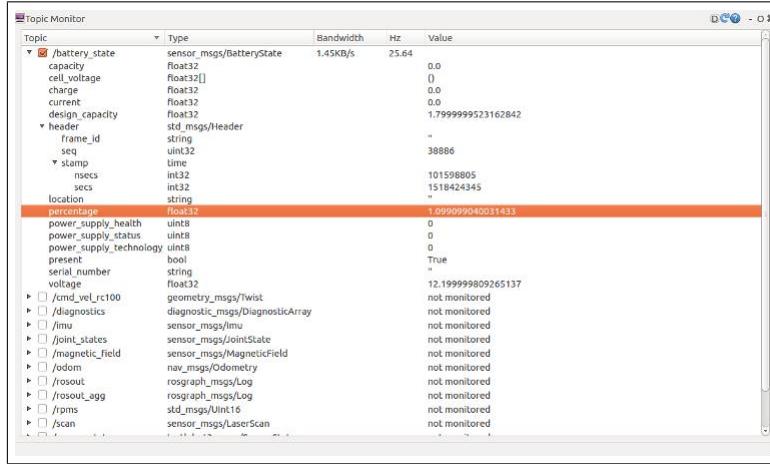


Figura 3.29: RQt mostrando los datos del tópico battery.

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 esquema_lider_seguidor, 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 Turtlebot 3 burger, debido a que en un principio cada Turtlebot 3 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 envía 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 fuera hecha correctamente, esta 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. Para este algoritmo se toman los puntos y se grafican a la dirección que indican ir, el conjunto de todos las direcciones graficadas generan un aproximado de la trayectoria original. 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.

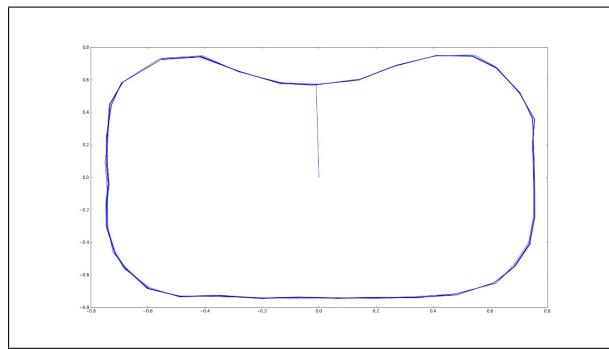


Figura 3.30: Reconstrucción de la trayectoria A.

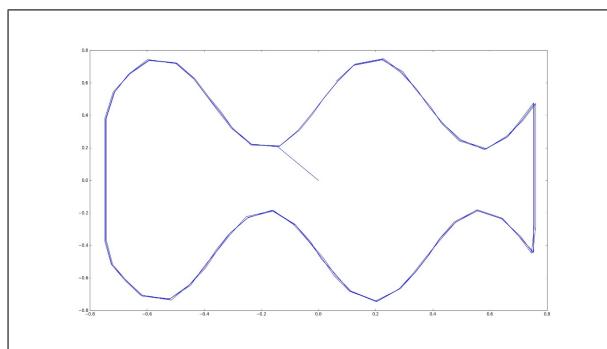


Figura 3.31: Reconstrucción de la trayectoria B.

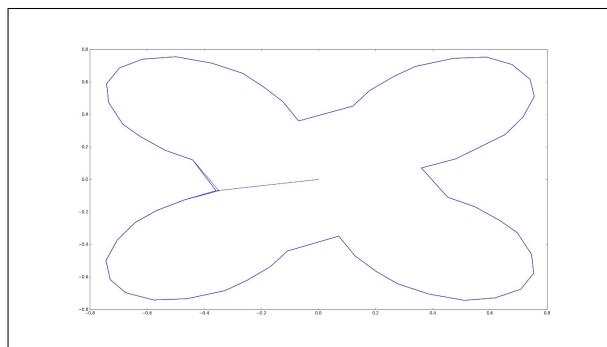


Figura 3.32: Reconstrucción de la trayectoria C.

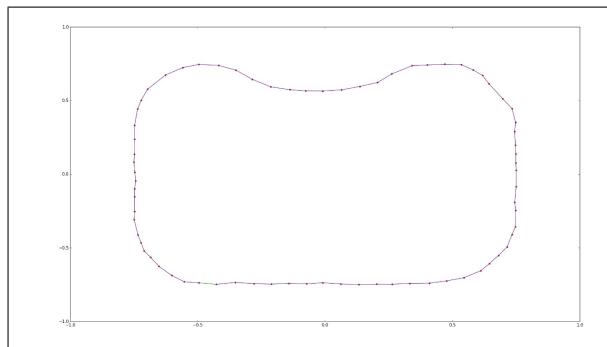


Figura 3.33: Puntos de trayectoria A.

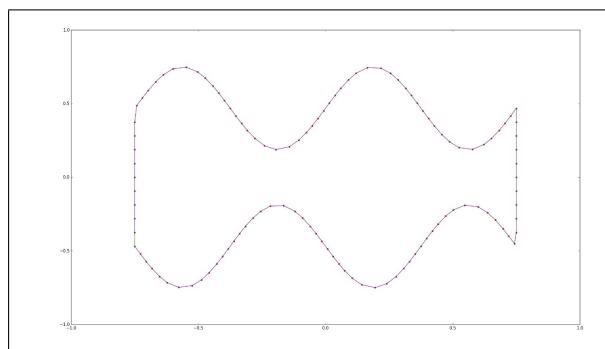


Figura 3.34: Puntos de trayectoria B.

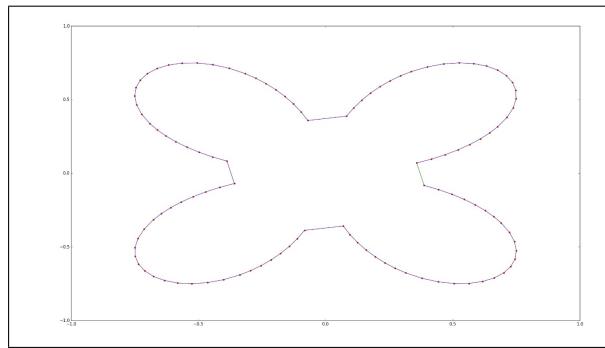


Figura 3.35: Puntos de trayectoria C.

3.5.3.1. Análisis de resultados

De las Figuras 3.30, 3.31, 3.32, 3.33, 3.34 y 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 uso del ambiente de simulación hecho en Gazebo para saber si funcionaba correctamente, en cuanto a la simulación se hizo la confirmación del seguimiento de trayectoria, 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 esquema_lider_seguidor lider.launch`” por medio de la computadora remota iniciando la recabación de las posiciones con el comando, “`rosrecord tb3_0/amcl_pose tb3_1/amcl_pose`”, los datos que se recabaron fueron 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 3.36, 3.37 y 3.38.

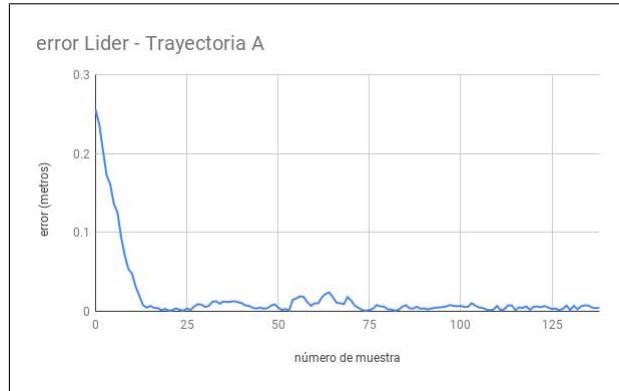


Figura 3.36: Error del líder en seguimiento de la trayectoria A.

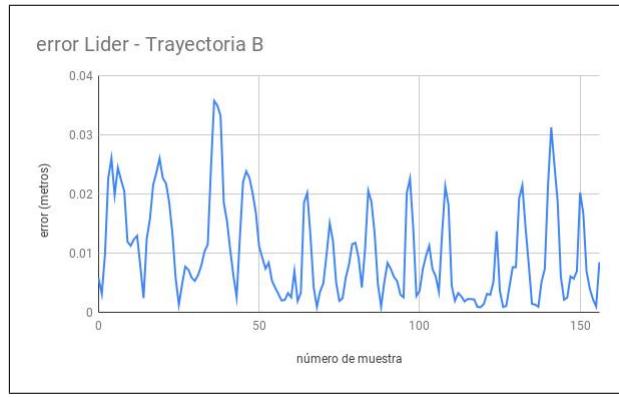


Figura 3.37: Error del líder en seguimiento de la trayectoria B.

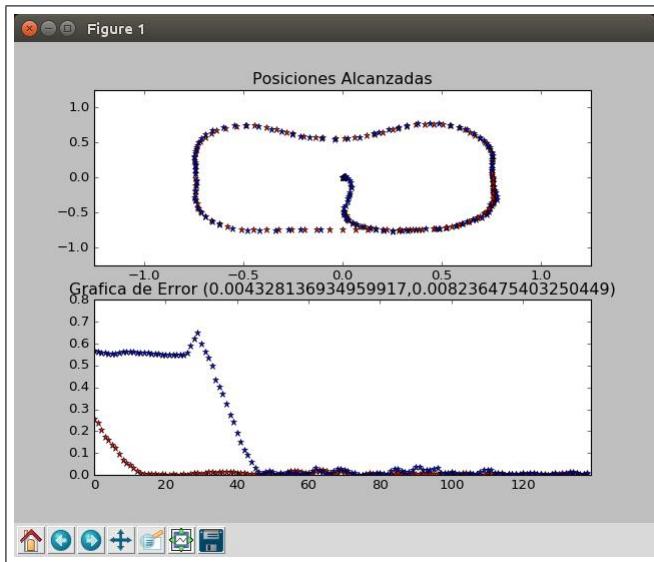


Figura 3.38: Gráfica de las posiciones alcanzadas y el error.

3.5.4.1. Análisis de resultados

Se puede observar de las Figuras 3.36, 3.37 y 3.38 como el error disminuye y posteriormente se mantiene 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 permitir cambiar la dirección del móvil una vez actualizada la posición del seguidor, esta condición se aplicó debido a que el algoritmo utilizado en su primera versión 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 que llega el seguidor.

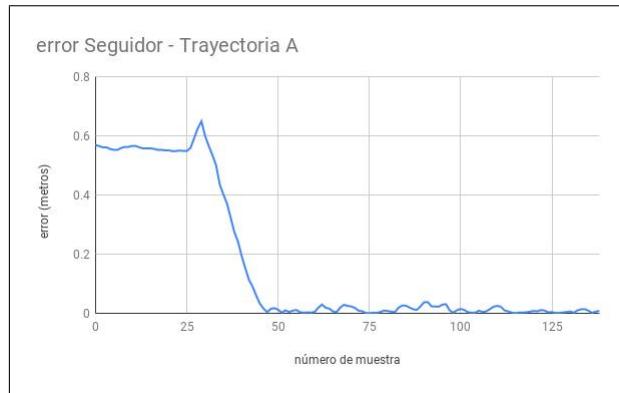


Figura 3.39: Error del seguidor en la trayectoria A.



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 como se muestra en las Figuras 3.39 y 3.40, obteniendo un error respecto a la trayectoria, dicho error, después de llegar a la trayectoria se mantiene en cierto umbral, validando así 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.

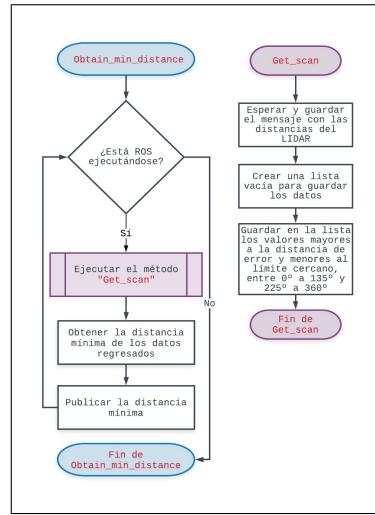


Figura 3.41: Método para calcular la distancia mínima y publicarla.

En el programa donde se implementó el VFH fue necesario validarla 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.

Integración del Sistema

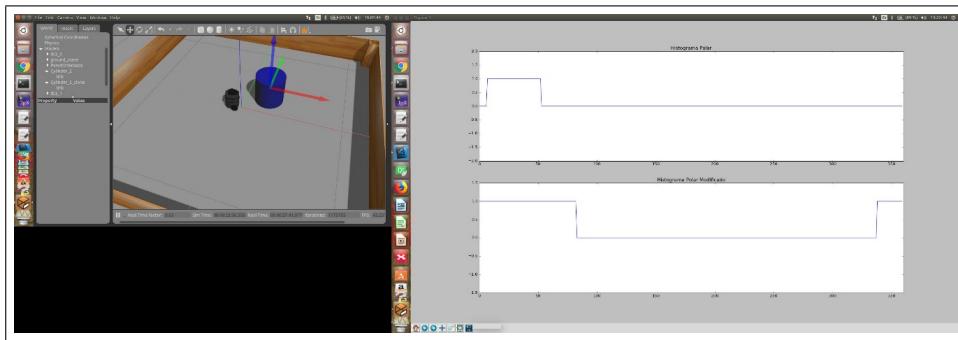


Figura 3.42: Detección del obstáculo cercano colocado en el primer cuadrante.

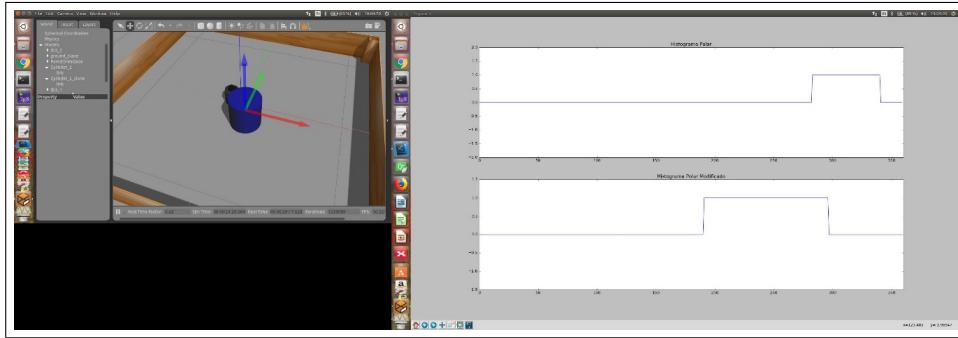


Figura 3.43: Detección del obstáculo cercano colocado en el cuarto cuadrante.

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 un método llamado “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 dimensiones 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.

3.5 Implementación del Área Funcional Procesamiento

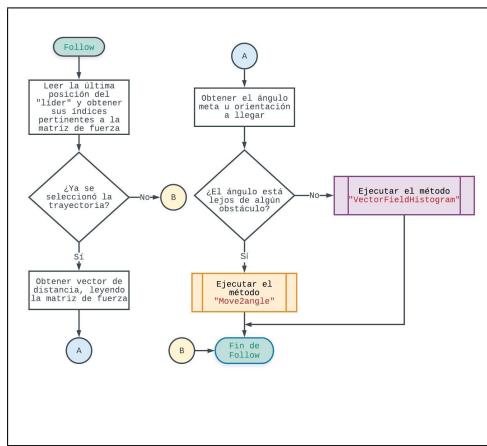


Figura 3.44: Función Follow para ejecutar VFH o seguimiento de trayectoria.

Para incluir la evasión de obstáculos se propuso tener una distancia mínima mostrada en la Figura 3.41, con la cual se decide si 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 alinear con el ángulo deseado con 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 garantizando la rotación en su propio eje para cambios angulares grandes. Estas velocidades son publicadas para que el robot tome estos valores de velocidad, posteriormente se le da un retardo a la ejecución del código para tener tiempo suficiente de recopilar los datos.

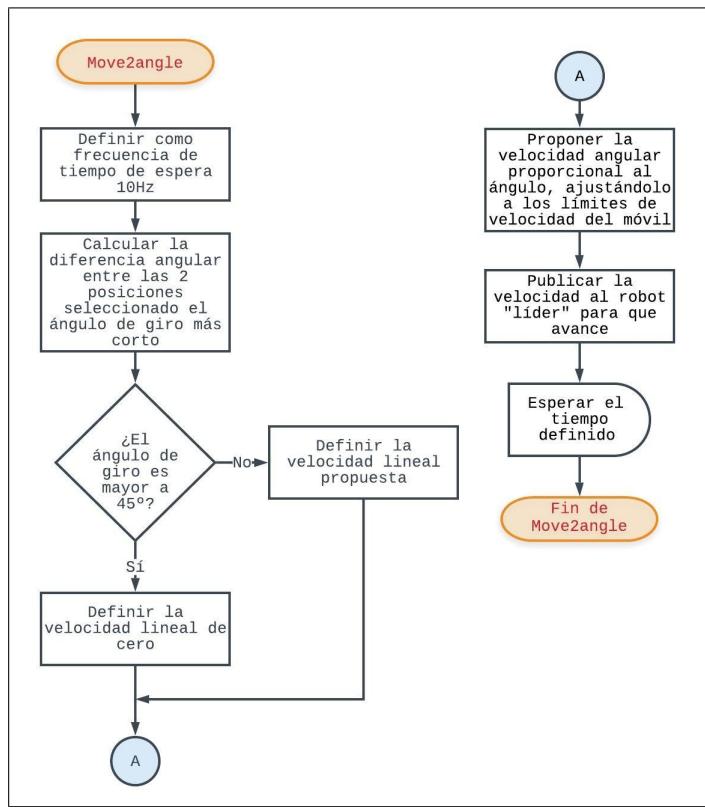


Figura 3.45: Función de seguimiento de trayectoria por dirección.

En el caso en que la distancia es menor a la propuesta como se muestra en la Figura 3.46 se verifica si no hay un obstáculo en la dirección por la que el robot debe ir. De no haber obstáculo sobre el ángulo deseado se ejecuta la rutina anterior con el mismo ángulo indicado por la trayectoria, 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 si está muy cerca del obstáculo y ahora alinea la orientación del robot con el nuevo angulo libre, finalmente la nueva velocidad es publicada. Esto se ejecuta controlado por la interfaz que indica el avance o paro del robot.

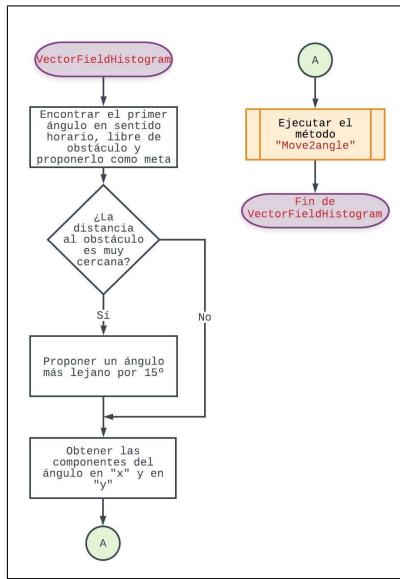


Figura 3.46: Algoritmo para aplicar VFH.

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



3.5.6.2. Análisis de resultados

Como se observó en las Figuras 3.42 y 3.43 los algoritmos de evasión funcionan correctamente, ya que el histograma muestra datos correctos, para los diferentes casos en que se presenta el obstáculo.

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

El objetivo de esta prueba fue validar el funcionamiento del sistema en conjunto, líder-seguidor 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 recorridos para cada trayectoria, si únicamente colisionaba una vez por trayectoria se consideró válido, adicionalmente mostramos las gráficas de error en las Figuras 3.47 y 3.48.

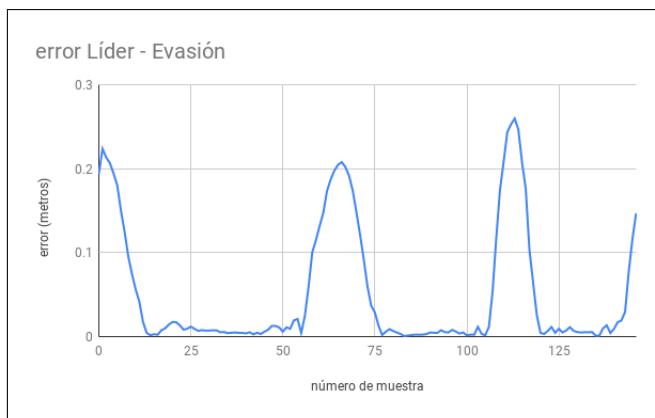


Figura 3.47: Error del Líder durante la evasión.

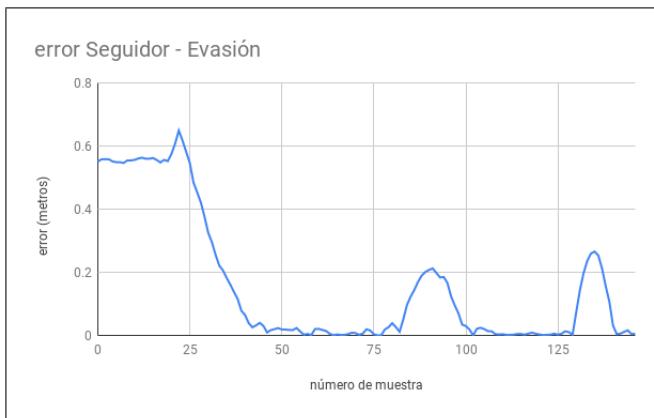


Figura 3.48: Error del Seguidor durante la evasión.

3.5.7.1. Análisis de resultados

Como se puede observar en las Figuras 3.47 y 3.48, 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.

3.5.8. Realización de la HMI

Para lograr que el sistema tenga una mejor interacción con el usuario se optó por integrarle 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 esto por medio de la interfaz RQt realizada en Qt, en dicha interfaz se pueden 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 Turtlebot 3 burger y graficar la información de los nodos a través del tiempo entre muchas otras funciones como las que en las Figuras 3.49 y 3.50 se visualizan.

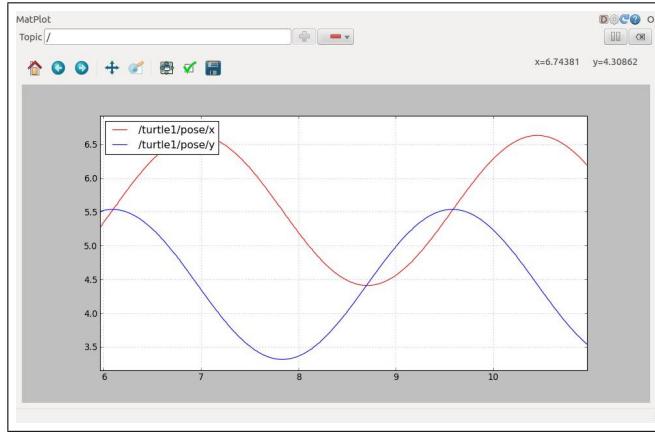


Figura 3.49: RQt para graficar distintos tópicos con respecto al tiempo.

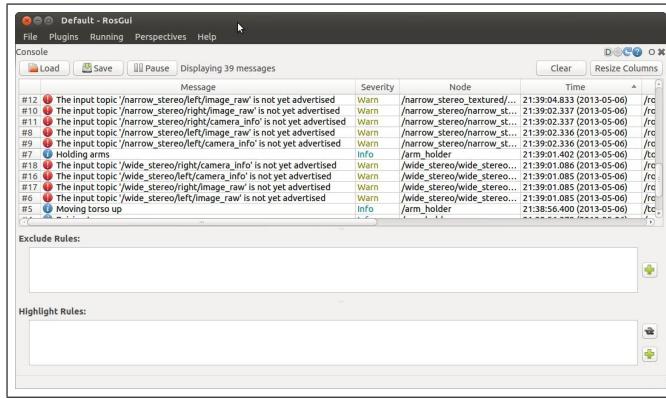


Figura 3.50: RQt mostrando y filtrando los distintos tipos de mensajes.

Con el objetivo de usar Qt, dada su versatilidad con ROS, 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 en una computadora con Ubuntu 16.04 se usaron las siguientes instrucciones en la terminal de comandos de Ubuntu:

- 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 [36], como detalle importante cabe mencionar que para generar el paquete de ROS relacionado con la interfaz se usó la instrucción en la terminal en la carpeta catkin_ws de la siguiente forma:

- catkin_create_qt_pkg gui_lider_seguidor

Donde “catkin_create_qt_pkg”, es la instrucción para generar el paquete y “gui_lider_seguidor”, 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 realiza la compilación correctamente entonces 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 3.57.

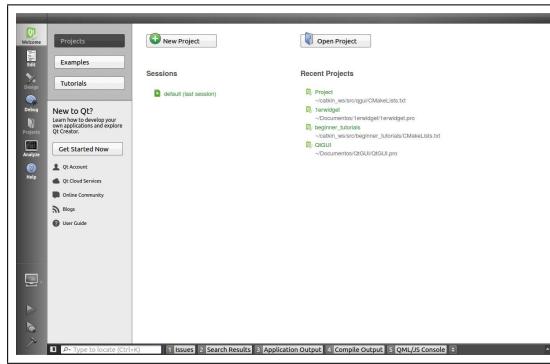


Figura 3.51: Pantalla de bienvenida de QtCreator.



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 “gui_lider_seguidor”, como se muestra en la Figura 3.52.

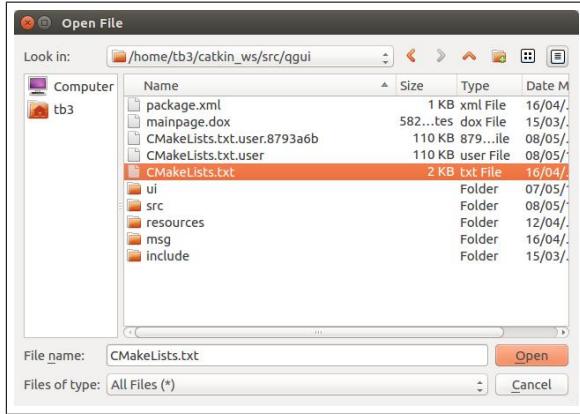


Figura 3.52: Ventana para compilar paquete en QtCreator.

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 3.53.

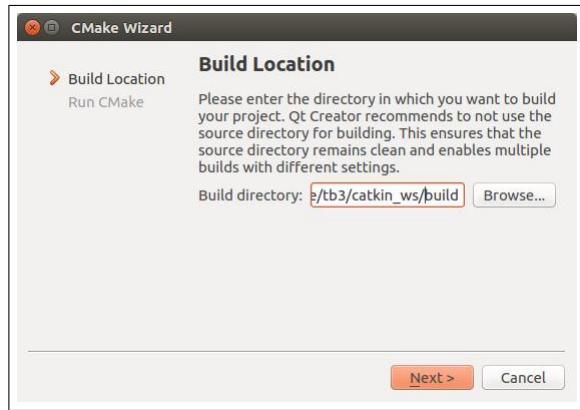


Figura 3.53: Ingreso de la carpeta de compilación.

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 más a detalle en la sección “catkin_make” de la wiki de ROS [35].

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 con el proyecto para poderse editar finalmente como se muestra en la Figura 3.54.

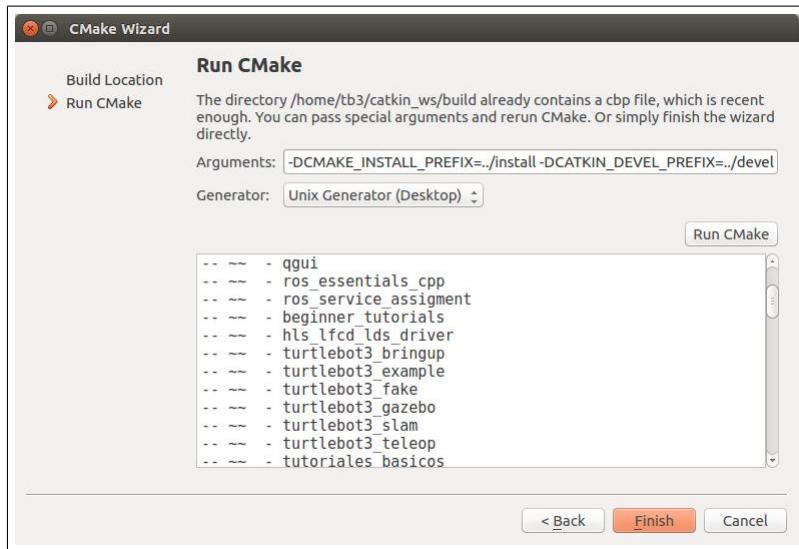


Figura 3.54: Ingreso y compilación con el comando CMake.

Una vez realizado esto exitosamente se muestra una ventana como la de la Figura 3.55, cabe mencionar que este procedimiento se puede aplicar para cualquier paquete del la carpeta “catkin_ws”.

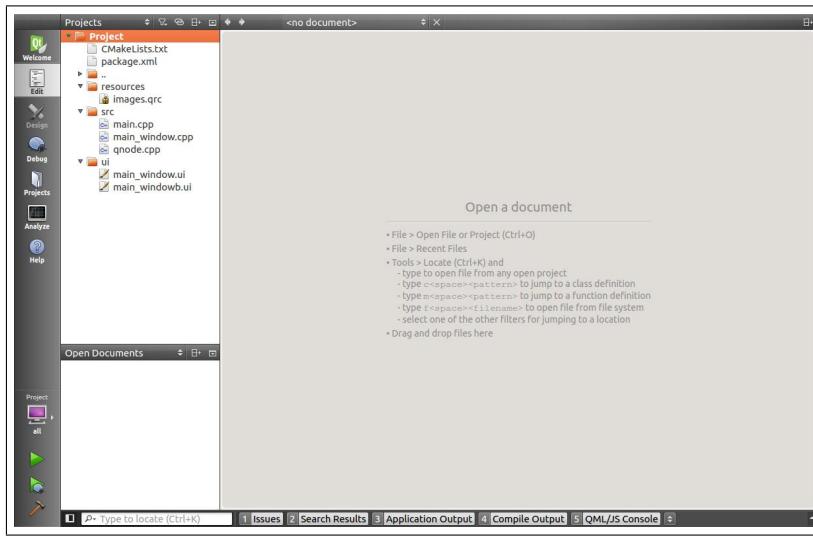


Figura 3.55: Ventana en Qt con el paquete qgui.

La edición del paquete se describe a detalle igualmente en la wiki de ROS en la sección de “qt_build” [36], pero a grandes rasgos este proyecto se divide en 4 partes:

- El programa principal del proyecto.
- El programa de la funcionalidad del código de 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.

3.5.8.1. 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 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 envíá mensajes al ecosistema de ROS en los tiempos adecuados. Este programa en general contiene todo porque de aquí llaman a los demás programas anteriormente mencionados.

3.5.8.2. 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.

3.5.8.3. 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 dando funcionalidad y manejan definiciones de la clase respectivamente. En cuanto a lo que realiza este programa es inicializar la comunicación con ROS cuando le es pedido desde la interfaz gráfica, lo 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 programa de la ventana gráfica y por defecto contiene el sistema de llamado de mensajes de consola de ROS.

3.5.8.4. Framework de Trabajo

El framework de trabajo es una herramienta muy útil, ya que se puede editar fácilmente y de manera interactiva 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 3.56.

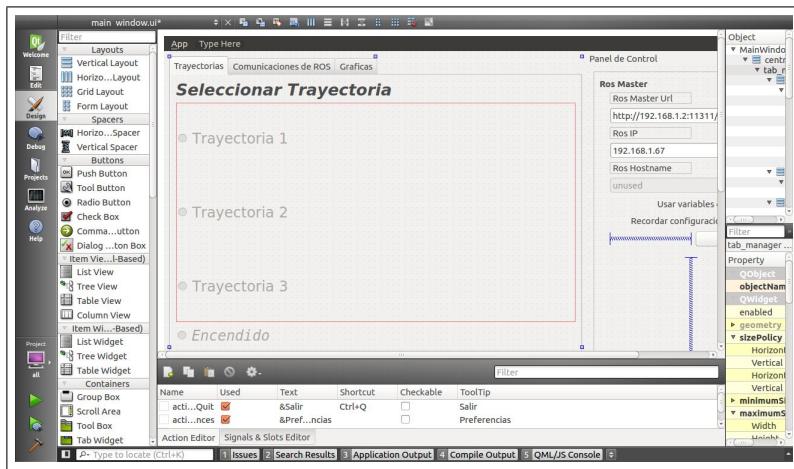


Figura 3.56: Framework de Qt para GUI.



3.5.8.5. 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 al generar código erróneo, posteriormente compilarlo y después con este procedimiento ya aparecerán los nuevos componentes reconocidos automáticamente por el sistema para así 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`”.

3.5.8.6. 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 editar lo correctamente sin presentar errores.

3.5.8.7. 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 3.57 y 3.58 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 predeterminada es la 0 en la cual nunca se avanza.

3.5 Implementación del Área Funcional Procesamiento

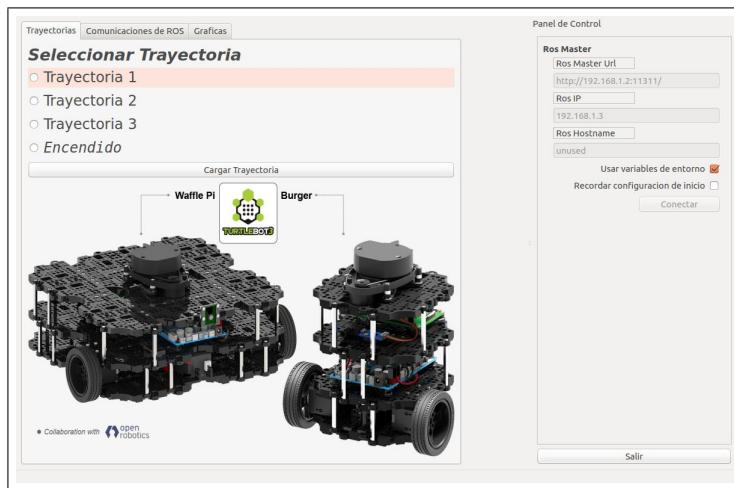


Figura 3.57: Ventana inicial de la interfaz.

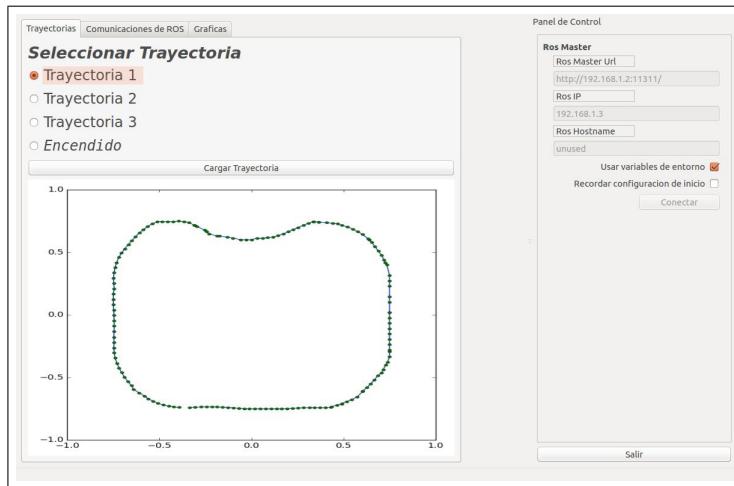


Figura 3.58: Configuración de la interfaz con la trayectoria 1 seleccionada y apagada.

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 3.59 y 3.60, para apagarlo basta con volver a presionar el botón.

Integración del Sistema

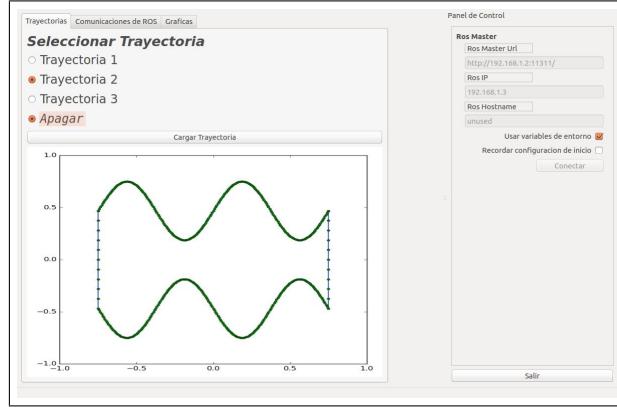


Figura 3.59: Configuración de la interfaz con la trayectoria 2 seleccionada y encendida.

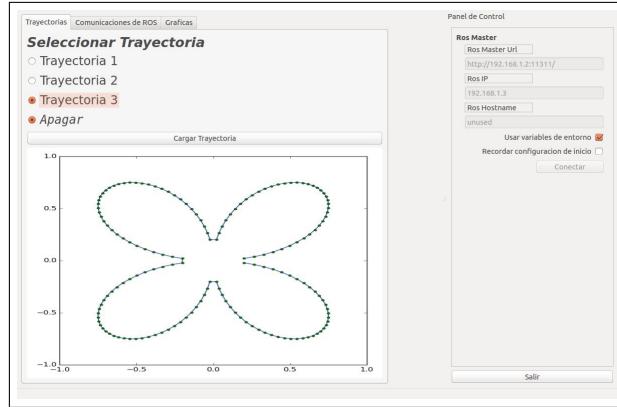


Figura 3.60: Configuración de la interfaz con la trayectoria 3 seleccionada y encendida.

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 3.61 sin nunca haber cargado ninguna trayectoria y en la Figura 3.62 con un historial de trayectorias cargadas (donde además se puede consultar cual es la trayectoria a seguir actualmente).

3.5 Implementación del Área Funcional Procesamiento

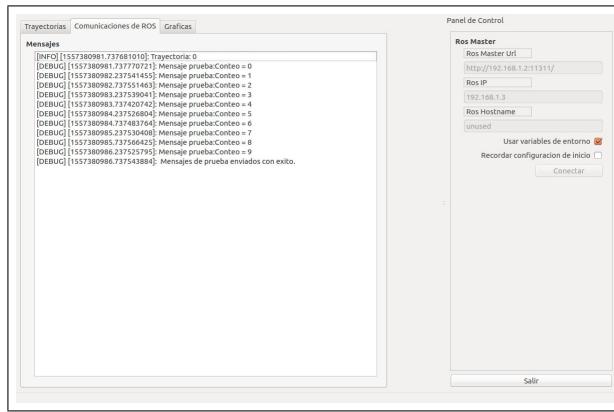


Figura 3.61: Interfaz con los mensajes de prueba enviados.

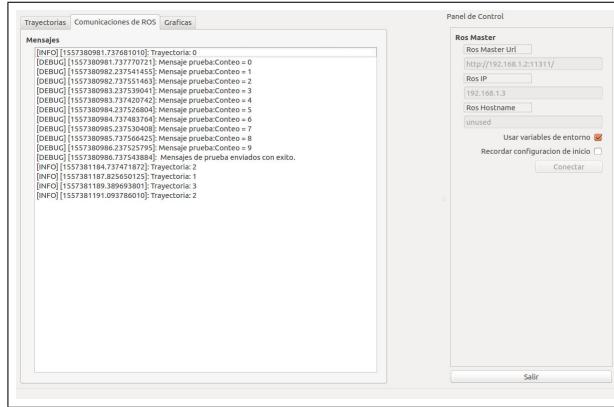


Figura 3.62: Interfaz con el historial de trayectorias seleccionadas.

En la tercera ventana como se observa en la Figura 3.63 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 .

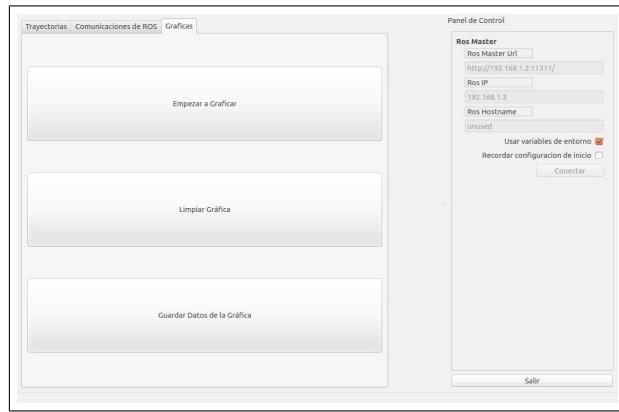


Figura 3.63: Interfaz mostrando las opciones de graficación.

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 baterías LiPo, la velocidad de los motores, la resolución de los sensores (LIDAR y encoders), 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 Alimentación

Cada batería 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

En la Figura 3.64 se muestran la red de nodos implementada para el proyecto.

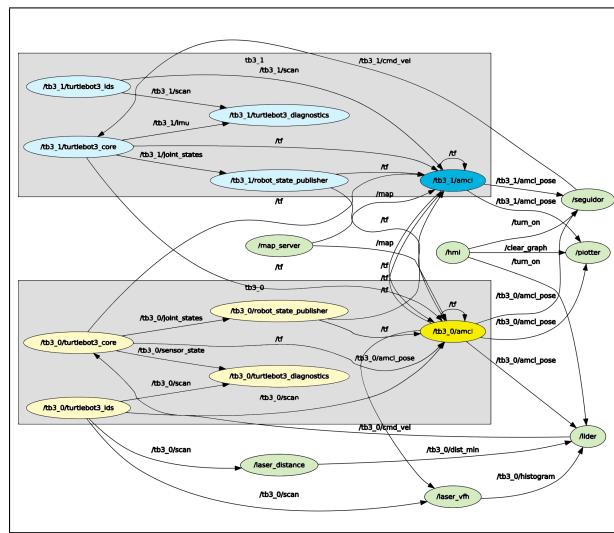


Figura 3.64: Nodos de Rosgraph.

Análisis de Costo

4.1. Costos variables

Los costos variables corresponden a los costos directos involucrados en la producción y venta de un artículo. Alineándonos a nuestro proyecto, no cubrimos gastos por ventas, pero si por producción, en donde, se derivan dos categorías:

- Costos por materias primas.
- 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
Turtlebot 3 burger	2	590 USD*	1180 USD*
Teclado y mouse Logitech	2	343.97 MXN	687.80 MXN
Cargador 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

Tabla 4.1: Tabla de costos variables.

*La compra de los Turtlebot 3 burger se hizo el 14 de septiembre de 2018, ese día por el tipo de cambio se compraba un dólar estadounidense a 19.24 pesos mexicanos.

4.2. Costos fijos

Los costos fijos se presentan a continuación considerando las horas trabajadas.

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

Tabla 4.2: Días trabajados en TT1.



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

Tabla 4.3: Días trabajados TT2.

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

Tabla 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 de estancia 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

Tabla 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.

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

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 ejecutar correctamente todos los sensores para la correcta recabación de datos, así mismo diversos algoritmos fueron satisfactoriamente implementados para el correcto comportamiento de los robots, todo bajo el control de una interfaz gráfica, 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 algoritmo 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 aumentó la velocidad de respuesta del sistema y dio paso a tener movimientos fluidos en ambos robots sin complicar la programación por la adición de otras acciones a ejecutar.



Al realizar un sistema de robots múltiples se dejó como contribución y al alcance de la comunidad politécnica dos algoritmos totalmente funcionales para ser usados en diversas aplicaciones relacionadas con la robótica cooperativa. Entre ellos está el VFH implementado en ROS y en Python, haciendo uso únicamente de un sensor LIDAR, lo que permite su aplicación con facilidad a otros robots o sistemas y el algoritmo líder-seguidor que puede ser aplicado para más de dos robots o con distintas configuraciones motrices.

Los resultados de este proyecto dejan en la Unidad Profesional las bases para poder desarrollar aplicaciones prácticas, entre las cuales está el transporte del material con robots móviles autónomos, un sistema eficiente de flujo de tráfico aplicando el principio del esquema líder-seguidor y la evasión de obstáculos con drones por medio de sensores LIDAR en combinación con 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 considerando de este análisis entonces se eligió al Turtlebot 3 burger, la segunda parte del objetivo se cumplió al ensamblar los 2 Turtlebot 3 burger como lo indicaban los manuales y ver su desempeño en la superficie de prueba. 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 en 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 FIFO, que cumple con la función de orientar de forma correcta al seguidor, para así imitar la trayectoria realizada por el líder. La implementación fue hecha con el lenguaje de programación Python, ROS. Los detalles de implementación se mencionan en el área funcional de procesamiento, específicamente en la prueba C.

Establecer e implementar un método de comunicación apropiado para la configuración líder seguidor. El método de comunicación que se eligió fue por medio de ROS que permite la fácil 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 (Vector Field Histogram), el cual fue 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 en 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 seleccionado 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 del objetivo en cuestión. Los detalles de implementación se profundizan en la sección de Implementación titulada Realización de la HMI.

Referencias

- [1] Allevato, A. (2019). Names. <http://wiki.ros.org/Names>.
- [2] Alvaro, G. (2013). Ros: Robot operating system. *Universidad Politécnica de Cartagena*.
- [3] AutoModelCar (2016). *Robotis e-Manual*. <http://emanual.robotis.com/docs/en/platform/turtlebot/> specifications.
- [4] Berlin, F. U. (2018). *Automodel Car*. Dahlem Center for Machine Learning and Robotics, Freie Universität Berlin. <https://github.com/AutoModelCar/AutoModelCarWiki/wiki/Navigation>.
- [5] Bishop, R. H. (2008). *The mechatronics handbook*. London.
- [6] 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.
- [7] 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*.

REFERENCIAS



- [8] Darpa (2007). Route network definition file (rndf) and mission data file (mdf) formats. *Grandchallenge*, 1:4 to 9. https://www.grandchallenge.org/grandchallenge/docs/RNDF_MDF_Formats_031407.pdf.
- [9] García Tovar, V. E., Guzmán Martínez, S., Pérez Nicolás, P., and Terán Chapul, R. E. (2017). Robot repartidor autonómico.
- [10] 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.
- [11] Gerkey, B. P., Lu, D. V., Ferguson, M., and Hoy, A. (2018). Amcl. <http://wiki.ros.org/amcl>.
- [12] 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>.
- [13] Gutiérrez, J. (2014). Qué es un framework web. *ros.org*.
- [14] Harashima, F., Tomizuka, M., and Fukuda, T. (1996). *Mechatronics, What is it, why and how?* IEEE/ASME Transactions on Mechatronics.
- [15] Jet (2017). Tx1 jet robot kit. <https://www.servocity.com/tx1-jet-robot-kit>.
- [16] Juárez Palafox, J., Muro Maldonado, D., and Franco González, A. (2005). Construcción y control de un robot móvil.
- [17] Juliá Cristóbal, M. (2005). Seguimiento de robots mediante visión artificial. aplicación al mantenimiento de formaciones basada en comportamientos.
- [18] Kumar, P. (2016). *Install Ubuntu*. <https://www.linuxtechi.com/install-ubuntu-16-04-with-screenshots/>.



- [19] Lagunes Fortiz, M. A. (2014). Sistema de navegación evasor de obstáculos en un robot móvil.
- [20] Madhevan, B. and Sreekumar, M. (2013). Tracking algorithm using leader follower approach for multi robots. *Procedia Engineering*, 64:1426 to 1435.
- [21] Molina Villa, M. A. and Rodríguez Vásquez, E. L. (2014). Flotilla de robots para trabajos en robotica cooperativa.
- [22] 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.
- [23] Rangel, Cortina, A., Park, J. Y., and Lloyd (2016). Turtlebot 3 burger [us].
- [24] Raspberrypi.org (2013). *SSH (Secure Shell)*.
<https://www.raspberrypi.org/documentation/remote-access/ssh/>.
- [25] Ribeiro, M. I. (2005). Obstacle avoidance.
<http://users.isr.ist.utl.pt/mir/pub/ObstacleAvoidance.pdf>.
- [26] Riveros Guevara, A. and Solaque Guzmán, L. E. (2013). Formación de robots móviles mediante el uso de controladores. *USBMed*, page 63 to 64.
- [27] Ro-Botica (2016). *DYNAMIXEL XL430-W250*. Accensit.
<http://emanual.robotis.com/docs/en/dxl/x/xl430-w250/control-table-of-eeprom-area>.
- [28] Ro-Botica (2018). *Motores DYNAMIXEL*. Accensit. <http://robotica.com/tienda/ROBOTIS-DYNAMIXEL>.
- [29] Robotis (2016a). *e-Manual Features*. <http://emanual.robotis.com/docs/en/platform/turtlebot3/features>
- [30] Robotis (2016b). *e-Manual specifications*.
<http://emanual.robotis.com/docs/en/platform/turtlebot3/specifications/>.

REFERENCIAS



- [31] 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>.
- [32] Robotis (2016d). Raspberry pi 3 model b. <https://www.raspberrypi.org/products/raspberry-pi-3-model-b/>.
- [33] Robotis (2018a). *LIPo Battery 11.1V.* Purdue University. <http://www.robotis.us/lipo-battery-11-1v-1800mah-lb-012/>.
- [34] Robotis (2018b). Opencr. <http://emanual.robotis.com/docs/en/parts/controller/opencr10/>.
- [35] ROS (2014a). catkin make. http://wiki.ros.org/catkin/commands/catkin_make.
- [36] ROS (2014b). qt-build. http://wiki.ros.org/qt_ros.
- [37] RullyFoote (2019). Topics. <http://wiki.ros.org/topics>.
- [38] Siegwart, R., Nourbakhsh, I. R., and Scaramuzza, D. (2011). *Introduction to autonomous mobile robots.* MIT.
- [39] Store, H. (2017). Rosbot 2.0. <https://store.husarion.com/collections/dev-kits/products/rosbot?variant=8799150276662>.
- [40] Thrun, S., Burgard, W., and Fox, D. (2010). *Probabilistic robotics.* MIT Press.
- [41] 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>.
- [42] Vidal, R., Shakernia, O., and Sastry, S. (2003). Formation control of nonholonomic mobile robots with omnidirectional visual servoing and motion segmentation. *ieeexplore*.

- [43] Xiaohai, L., Jizong, X., and Zijun, C. (2005). Backstepping based multiple mobile robots formation control. *ieeexplore*.
- [44] 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

Apéndice 1



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 odometry 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 distribution 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: $\pi/6.0$ 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

Apéndice 1



```
~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-amcl-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-amcl-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		

Tabla 1: Tabla de selección para Percepción

Anexo 1. Criterios de selección en Áreas funcionales



	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
Ttb3	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
Ttb3	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
Ttb3	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		

Tabla 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
Ttb3	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
Ttb3	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
Ttb3	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		

Tabla 3: Tabla de selección para Procesamiento

Anexo 1. Criterios de selección en Áreas funcionales

898

	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
Ttb3	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
Ttb3	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
Ttb3	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		

Tabla 4: Tabla de selección para Motores