



# Proyecto Integrador Final



## Hermes III

Autores

Coutinho Juan Agustín – Malatini Hernán

Director

PhD. Orlando Micolini

Codirector:

Ing. Luis Ventre

## CONTACTOS

Coutinho, Juan Agustín

---

Tel: 0351-157895495

E-mail: cou647@gmail.com

Malatini, Hernán

---

Tel: 0358-154293197

E-mail: nani93@gmail.com

---

# ÍNDICE GENERAL

## CONTENIDO

<b>1</b>	<b>INTRODUCCIÓN.....</b>	<b>15</b>
1.1	ESTADO DEL ARTE .....	15
1.2	MOTIVACIÓN .....	19
1.3	OBJETIVO .....	19
1.3.1	<i>Objetivos principales.....</i>	20
1.3.2	<i>Objetivos secundarios .....</i>	20
1.4	REQUERIMIENTOS.....	21
1.4.1	<i>Requerimientos funcionales.....</i>	21
1.4.1.1	Pruebas de Requerimientos Funcionales .....	22
1.4.2	<i>Requerimientos no funcionales.....</i>	24
1.5	ANÁLISIS DE RIESGOS.....	25
1.5.1	<i>Listado de riesgos.....</i>	26
1.5.2	<i>Estimación de probabilidad .....</i>	28
1.5.3	<i>Estimación de impacto.....</i>	29
1.5.4	<i>Exposición al riesgo.....</i>	30
1.5.5	<i>Mitigación, Monitorización y Gestión de Riesgos (RMMM) .....</i>	31
1.6	ARQUITECTURA PRELIMINAR DE ALTO NIVEL DEL SISTEMA.....	32
<b>2</b>	<b>MARCO TEÓRICO.....</b>	<b>33</b>
2.1	METODOLOGÍA DE DESARROLLO .....	33
2.1.1	<i>Elección del modelo .....</i>	35
2.2	ROBOTS .....	36
2.2.1	<i>Sistemas mecánicos de locomoción .....</i>	36
2.2.1.1	Locomoción diferencial .....	36
2.2.1.2	Locomoción síncrona .....	36
2.2.1.3	Triciclo.....	37
2.2.1.4	Locomoción de Ackerman.....	38
2.2.1.5	Locomoción omnidireccional .....	38
2.2.2	<i>Restricciones holonómicas y no-holonómicas.....</i>	39
2.3	COMPONENTES .....	40
2.3.1	<i>Mapeo.....</i>	40
2.3.1.1	Kinect .....	42
2.3.2	<i>Posicionamiento.....</i>	44
2.3.2.1	Sensores optoacopladores .....	46
2.3.3	<i>Fuente de alimentación eléctrica.....</i>	48
2.3.3.1	Baterías de ion litio .....	49
2.3.3.2	Baterías de polímero de litio .....	49
2.3.3.3	Comparativa y elección .....	50
2.4	SISTEMA DE CONTROL.....	51
2.4.1	<i>Análisis físico .....</i>	51
2.5	SISTEMA OPERATIVO .....	53
2.5.1	<i>ROS (Robot Operating System) .....</i>	54
2.6	SISTEMA DE COMUNICACIÓN.....	56
2.6.1	<i>Transmission Control Protocol .....</i>	56
2.6.2	<i>Bluetooth .....</i>	58

2.6.3	<i>Wi-Fi</i> .....	58
<b>3</b>	<b>ITERACIÓN 0</b> .....	<b>60</b>
3.1	MATRIZ DE TRAZABILIDAD DE ITERACIONES - REQUERIMIENTOS/PRUEBAS .....	61
<b>4</b>	<b>ITERACIÓN 1: PUESTA A PUNTO DE ROS</b> .....	<b>62</b>
4.1	INTRODUCCIÓN .....	62
4.2	REQUERIMIENTOS.....	62
4.3	DESARROLLO .....	62
4.3.1	<i>Selección de la placa principal de desarrollo</i> .....	62
4.3.1.1	Nvidia Jetson Tegra K1 .....	64
4.3.2	<i>Selección de versión del Sistema Operativo</i> .....	67
4.3.2.1	Criterio general de ponderación .....	67
4.3.3	<i>Instalación de ROS "Indigo Igloo"</i> .....	69
4.3.3.1	Pasos a seguir sobre Ubuntu 14.04 .....	69
4.3.4	<i>Interconexión entre los distintos nodos de ROS</i> .....	71
4.4	PRUEBA .....	72
4.5	RESULTADOS.....	73
4.6	RIESGOS MITIGADOS.....	73
4.7	CONCLUSIONES .....	73
<b>5</b>	<b>ITERACIÓN 2: MODELADO Y SIMULACIÓN DEL ROBOT</b> .....	<b>75</b>
5.1	INTRODUCCIÓN .....	75
5.2	REQUERIMIENTOS.....	75
5.3	DESARROLLO .....	76
5.3.1.1	URDF .....	76
5.3.2	<i>Modelo URDF del Hermes</i> .....	79
5.3.2.1	Creación de archivos STL (Standard Triangle Language) .....	79
5.3.2.2	Creación de archivos COLLADA (COLLAborative Design Activity).....	82
5.3.2.3	Visualización del modelo en software de monitoreo.....	82
5.3.3	<i>Simulación</i> .....	83
5.3.3.1	Gazebo .....	83
5.3.3.2	Simulando el Robot Hermes.....	85
5.4	PRUEBAS.....	88
5.5	RESULTADOS.....	90
5.6	RIESGOS SUPERADOS.....	90
5.7	CONCLUSIONES .....	91
<b>6</b>	<b>ITERACIÓN 3: IMPLEMENTACIÓN DE SENSORES Y ACTUADORES</b> .....	<b>92</b>
6.1	INTRODUCCIÓN .....	92
6.2	REQUERIMIENTOS.....	92
6.3	DESARROLLO .....	92
6.3.1	<i>Actuadores</i> .....	93
6.3.1.1	Motores de corriente continua .....	93
6.3.2	<i>Sensor de Imágenes y Profundidad</i> .....	97
6.3.2.1	Kinect .....	97
6.3.2.2	Autosuspendido del USB .....	98
6.3.4	<i>Sensor de Velocidad</i> .....	100
6.3.4.1	1. Obtención de datos de un fenómeno físico .....	100
6.3.4.2	2. Procesamiento, transformación y disponibilización de los datos .....	105
6.3.4.3	Tópicos ROS.....	108

6.3.4.4	Diagrama de secuencia .....	110
6.3.5	<i>Diagrama esquemático</i> .....	111
6.4	PRUEBAS.....	112
6.5	RESULTADOS.....	113
6.6	RIESGOS SUPERADOS.....	114
6.7	CONCLUSIONES .....	115
<b>7</b>	<b>ITERACIÓN 4: SLAM (SYSTEM LOCALIZATION AND MAPPING) .....</b>	<b>116</b>
7.1	INTRODUCCIÓN .....	116
7.2	REQUERIMIENTOS.....	116
7.3	DESARROLLO .....	116
7.3.1	<i>Real Time Appearance Based Mapping</i> .....	116
7.3.1.1	Instalación .....	117
7.3.2	<i>Configurando RTAB Map en Hermes III</i> .....	119
7.3.3	<i>Ajuste de parámetros de mapeo</i> .....	120
7.3.3.1	Parámetros configurados .....	120
7.3.4	<i>Precisión del algoritmo SLAM</i> .....	122
7.3.4.1	Información de profundidad simple.....	123
7.3.4.2	Información de profundidad compleja .....	124
7.4	PRUEBAS.....	127
7.5	RESULTADOS.....	129
7.6	RIESGOS SUPERADOS.....	129
7.7	CONCLUSIONES .....	130
<b>8</b>	<b>ITERACIÓN 5: SISTEMA DE CONTROL.....</b>	<b>131</b>
8.1	INTRODUCCIÓN .....	131
8.2	REQUERIMIENTOS.....	131
8.3	DESARROLLO .....	131
8.3.1	<i>PID Ros</i> .....	131
8.3.1.1	Configurando Hermes .....	132
8.3.1.2	Error de Inmovilización .....	134
8.3.1.3	Medición de velocidad de las ruedas deficiente .....	137
8.3.1.4	Precisión del sistema de control .....	137
8.4	PRUEBAS.....	138
8.5	RESULTADOS.....	139
8.6	RIESGOS MITIGADOS .....	139
8.7	CONCLUSIONES .....	140
<b>9</b>	<b>ITERACIÓN 6: SISTEMAS DE MANDO .....</b>	<b>142</b>
9.1	INTRODUCCIÓN .....	142
9.2	REQUERIMIENTOS.....	142
9.3	DESARROLLO .....	142
9.3.1	<i>Mando a través de teclado de PC</i> .....	142
9.3.2	<i>Mando a través de Joystick Inalámbrico</i> .....	143
9.3.3	<i>Mando a través de Smartphone</i> .....	144
9.4	PRUEBAS.....	145
9.5	RESULTADOS.....	146
9.6	RIESGOS SUPERADOS.....	146
9.7	CONCLUSIONES .....	146
<b>10</b>	<b>SISTEMA FINAL.....</b>	<b>148</b>

10.1	INTRODUCCIÓN .....	148
10.2	INTEGRACIÓN DE COMPONENTES .....	148
10.2.1	<i>Diagramas de despliegue</i> .....	148
10.2.1.1	Hermes 3 .....	148
10.2.1.2	PC de Monitoreo.....	149
10.3	PUESTA EN FUNCIONAMIENTO.....	150
10.3.1	<i>Configuracion previa</i> .....	150
10.3.2	<i>Encender el robot</i> .....	150
10.3.3	<i>Verificación de conectividad</i> .....	150
10.3.4	<i>Configurar IP's Maestro-Esclavo de ROS</i> .....	150
10.3.4.1	NVIDIA Jetson .....	150
10.3.4.2	PC de Monitoreo.....	151
10.3.5	<i>Ejecución de launchfiles</i> .....	152
10.3.5.1	NVIDIA Jetson .....	152
10.3.5.2	PC de Monitoreo.....	152
10.4	COMANDOS DE CONTROL .....	154
10.5	PRUEBAS DE INTEGRACIÓN.....	154
10.6	ESTADO FINAL DE REQUERIMIENTOS .....	157
11	CONCLUSIONES .....	159
12	BIBLIOGRAFÍA.....	160
13	ANEXOS.....	163
13.1	A.1 MODELO_HERMESII.URDF.....	163
13.2	A.2 DISPLAY.LAUNCH.....	165
13.3	A.3 CONFIG.RVIZ .....	165
13.4	A.4 GAZEBO.LAUNCH .....	170
13.5	A.5 MODELO_HERMESIII.URDF.....	170
13.6	A.6 DISPLAY_HERMESIII.LAUNCH.....	171
13.7	B.1 ATMEGA128CAPTUREWITHI2C.INO .....	172
13.8	B.2 ARDUINO_CONTROLLER_HERMES3_V2.5_WITH_ATMEGA128A.INO .....	174
13.9	C.1 RTABMAP_ODOMETRY.LAUNCH .....	185
13.10	C.2 SLAM_RTABMAP.LAUNCH .....	186
13.11	C.3 REMOTE_VISUALIZATION.LAUNCH .....	186
13.12	C.4 REMOTE_VISUALIZATION.RVIZ .....	186
13.13	D.1 ARDUINO_ROS.SH.....	187
13.14	D.2 PID.LAUNCH.....	187
13.15	D.3 SEND_VELOCITY.PY .....	188
13.16	D.4 PID_GENERAL.PY .....	189
13.17	D.5 FIX_INTEGRALERROR.PY .....	190
13.18	E.1 XBOX360.PY .....	192
13.19	E.2 XBOX360_TELEOP.LAUNCH.....	193
13.20	E.3 KEYBOARD_TELEOP.LAUNCH.....	193
13.21	F.1 GENERAL.LAUNCH.....	194
13.22	F.2 IMAGEN DE COMPONENTES.....	194



# ÍNDICE DE FIGURAS

Figura 1-1: Mars 3 .....	15
Figura 1-2: Opportunity/Spirit .....	16
Figura 1-3: Curiosity .....	16
Figura 1-4: ExoMars Rover .....	16
Figura 1-5: Roomba 980 .....	17
Figura 1-6: STAIR: STanford Artificial Intelligence Robot.....	18
Figura 1-7: Capacidades de un robot móvil .....	19
Figura 1-8: RMMM .....	25
Figura 1-9:Arquitectura del sistema .....	32
Figura 2-1: Desarrollo en cascada .....	33
Figura 2-2: Desarrollo en espiral .....	34
Figura 2-3: Desarrollo basado en componentes .....	34
Figura 2-4: Desarrollo incremental.....	35
Figura 2-5: Movimientos de locomoción diferencial.....	36
Figura 2-6: Movimientos de locomoción sincrona .....	37
Figura 2-7: Movimientos de triciclo .....	37
Figura 2-8: Movimientos de locomoción de Ackerman .....	38
Figura 2-9: Movimientos de locomoción omnidireccional .....	38
Figura 2-10: Rueda omnidireccional .....	39
Figura 2-11: Componentes de Microsoft Kinect .....	42
Figura 2-12: Luz estructurada .....	42
Figura 2-13: Cálculo de distancia en luz estructurada.....	43
Figura 2-14: Profundidad de enfoque .....	43
Figura 2-15: Sensado de Microsoft Kinect dos instantes de tiempo diferentes.....	44
Figura 2-16: Trayectoria de robot en un plano 2D .....	44
Figura 2-17: Mapa elaborado con distancia a objetos .....	45
Figura 2-18: Movimientos sobre seis grados de libertad .....	45
Figura 2-19: Grados de libertad para un robot terrestre .....	46

Figura 2-20: Sensor octocoplador .....	46
Figura 2-21: Sensado de un sensor octocoplador .....	47
Figura 2-22: Encoder .....	47
Figura 2-23: Distancia en base al radio de una rueda .....	48
Figura 2-24: Batería recargable .....	49
Figura 2-25: Batería de Ion Litio .....	49
Figura 2-26: Batería de Polímero de Litio .....	50
Figura 2-27: Sistema de control PID .....	51
Figura 2-28: Fuerzas de un sistema onmidireccional .....	51
Figura 2-29: Intercambio de paquetes TCP .....	57
Figura 2-30: Reenvío de paquetes perdidos .....	57
Figura 2-31: Modelo Maestro-Esclavo .....	58
Figura 4-1: Distancia a ciegas vs FPS .....	63
Figura 4-2: SoC Nvidia Tegra K1 .....	64
Figura 4-3: Procesador “NVIDIA 4-Plus-1™ Quad-Core ARM Cortex-A15” .....	65
Figura 4-4: Dual ISP .....	66
Figura 4-5: Dual Next Gen ISP .....	66
Figura 4-6: GPU Nvidia con CUDA Cores .....	67
Figura 4-7: Repositorios de paquetes Ubuntu .....	69
Figura 4-8: Conexión roscore maestro-esclavo .....	71
Figura 4-9: Roscore corriendo sobre Ubuntu 14.04 .....	72
Figura 5-1: "Link" de una pieza .....	76
Figura 5-2: "Joint" de dos piezas .....	77
Figura 5-3: Modelo ejemplo de unión de varias piezas .....	78
Figura 5-4: Rueda onmidireccional .....	80
Figura 5-5: Diseño 3D de rueda onmidireccional .....	80
Figura 5-6: Rodillo de rueda onmidireccional .....	81
Figura 5-7: Chasis de Hermes II .....	81
Figura 5-8: Diseño 3D de chasis de Hermes II .....	82
Figura 5-9: Movimiento y visualización de cuerpos rígidos .....	83

Figura 5-10: Simulación de turtlebot .....	84
Figura 5-11: Visualización de sensores sobre simulación de turtlebot.....	85
Figura 5-12: Simulación de Hermes II .....	86
Figura 5-13: Visualización de Hermes III .....	87
Figura 6-1: Motores de C.C.....	93
Figura 6-2: Componentes de un motor .....	93
Figura 6-3: Parámetros de cálculo .....	94
Figura 6-4: Torque del motor con eficiencia del 60% .....	94
Figura 6-5: Torque del motor con eficiencia del 80% .....	94
Figura 6-6: Cálculo de fuerzas .....	95
Figura 6-7: Controlador de motores .....	96
Figura 6-8: Diagrama esquematico del controlador .....	96
Figura 6-9: I/O del controlador .....	97
Figura 6-10: Imágenes obtenidas del sensor de profundidad y del sensor IR por parte de Kinect.....	99
Figura 6-11: Vista inferior del prototipo refuncionalizado .....	100
Figura 6-12:Vista del sensor acoplado al motor .....	100
Figura 6-13: ATmega128 M128 AVR Minimum Development Board .....	101
Figura 6-14: Configuración de FUSES del microcontrolador .....	102
Figura 6-15: Programador ISP genérico .....	102
Figura 6-16: Generación de binarios mediante IDE Arduino .....	103
Figura 6-17: Diagrama de flujo del programa que corre sobre el microcontrolador ATmega128 .....	104
Figura 6-18: Flujo de datos entre ATmega128 y Arduino .....	105
Figura 6-19: Diagrama de flujo del programa que corre sobre Arduino .....	105
Figura 6-20: Códigos de ejemplo de ROS sobre Arduino .....	109
Figura 6-21: Comunicación de Arduino con roscore .....	110
Figura 6-22: Diagrama de secuencia entre los distintos microcontroladores .....	110
Figura 6-23: Diagrama esquematico de conexión entre Arduino y ATmega128.....	111
Figura 7-1: Algoritmo de RTAB-Map.....	117
Figura 7-2: Configuración de RTAB-Map según la cantidad de sensores .....	119
Figura 7-3: Desplazamiento de Hermes III .....	120

Figura 7-4: Distancia mínima 46.5 centímetros .....	122
Figura 7-5: Posición inicial sin referencia (vista superior) .....	123
Figura 7-6: Desplazamiento de medio metro sin referencia (vista inferior) .....	123
Figura 7-7: Regreso al punto de partida sin referencia (vista inferior) .....	124
Figura 7-8: Posición inicial con referencia (vista superior) .....	125
Figura 7-9: Desplazamiento de medio metro con referencia (vista inferior) .....	125
Figura 7-10: Regreso al punto de partida con referencia (vista inferior) .....	126
Figura 7-11: Mapa 3D del Laboratorio de Arquitectura de Computadoras.....	129
Figura 8-1: Cálculo de velocidades de cada rueda .....	133
Figura 8-2: Inestabilidad a baja velocidad .....	134
Figura 8-3: Hermes III en movimiento.....	136
Figura 8-4: Hermes III detenido .....	136
Figura 8-5: Prueba de trayectoria en línea recta del robot Hermes III .....	137
Figura 8-6: Estado transitorio y estado estable [34] .....	138
Figura 9-1: Comando a través de teclado.....	143
Figura 9-2: Referencia de botones de Joystick Xbox 360 .....	143
Figura 9-3: Aplicación Ros Control para Android .....	144
Figura 10-1: Diagrama de despliegue Hermes 3 .....	148
Figura 10-2: Diagrama de despliegue PC de Monitoreo .....	149
Figura 10-3: Pines panel frontal NVIDIA Jetson .....	150
Figura 10-4: Conexión por SSH al Hermes III .....	151
Figura 10-5: Ejecución de general.launch por SSH 1.....	152
Figura 10-6: Ejecución de arduino_ros.sh por SSH 2 .....	152
Figura 10-7: Ejecución de remote_visualization.launch .....	152
Figura 10-8: RVIZ en PC de monitoreo .....	153
Figura 10-9: Comando del Hermes III por joystick.....	154

# ÍNDICE DE TABLAS

Tabla 1-1: Prioridad de requerimientos .....	21
Tabla 1-2: Requerimientos funcionales .....	21
Tabla 1-3: Pruebas de requerimientos funcionales .....	23
Tabla 1-4: Requerimientos no funcionales .....	24
Tabla 1-5: RI-01 Sistema Operativo inestable .....	26
Tabla 1-6: RI-02 Incompatibilidad o avería de componentes .....	26
Tabla 1-7: RI-03 Intercomunicación de componentes ineficiente o ineficaz .....	26
Tabla 1-8: RI-04 Prestaciones insuficientes de componentes .....	26
Tabla 1-9: RI-05 Modificación de los requerimientos del proyecto .....	27
Tabla 1-10: RI-06 Dificultad en conseguir determinados componentes .....	27
Tabla 1-11: RI-07 Excesivo tiempo para cumplir los objetivos del proyecto .....	27
Tabla 1-12: RI-08 Reducción de la fuerza de trabajo .....	27
Tabla 1-13: Cuantificación de la probabilidad .....	28
Tabla 1-14: Probabilidad de ocurrencia de riesgos .....	28
Tabla 1-15: Cuantificación del impacto de los riesgos .....	29
Tabla 1-16: Impacto en base a los diferentes riesgos .....	29
Tabla 1-17: Exposición al riesgo .....	30
Tabla 1-18: Plan de mitigación, monitorización, y gestión de los riesgos .....	31
Tabla 2-1: Lasers Range-Finder .....	40
Tabla 2-2: Sensores de profundidad 3D .....	41
Tabla 2-3: Comparación de baterías Li-Ion vs LiPo .....	50
Tabla 2-4: Comparación entre sistemas operativos .....	54
Tabla 2-5: Capas físicas del estándar 802.11 .....	59
Tabla 2-6: Comparación entre diferentes estándares .....	59
Tabla 3-1: Plan de acción frente a cada iteración .....	60
Tabla 3-2: Matriz de trazabilidad de iteraciones - requerimientos/pruebas .....	61
Tabla 4-1: Características Nvidia Jetson Tegra K1 .....	65
Tabla 4-2: Referencia de ponderación .....	68

Tabla 4-3: Comparación frente a diferentes versiones de ROS .....	68
Tabla 4-4: Riesgos mitigados en iteración 1 .....	73
Tabla 5-1: Requerimientos funcionales de iteración 2.....	75
Tabla 5-2: Prueba nro. 1 de iteración 2 .....	88
Tabla 5-3: Prueba nro. 2 de iteración 2 .....	89
Tabla 5-4: Prueba nro. 3 de iteración 2 .....	89
Tabla 5-5: Riesgos mitigados en iteración 2 .....	90
Tabla 6-1: Requerimientos funcionales de iteración 3.....	92
Tabla 6-2: Prueba nro. 1 de iteración 3 .....	112
Tabla 6-3: Prueba nro. 2 de iteración 3 .....	113
Tabla 6-4: Riesgos mitigados en iteración 3.....	114
Tabla 7-1: Requerimientos funcionales iteración 4.....	116
Tabla 7-2: Prueba nro. 1 iteración 4 .....	127
Tabla 7-3: Prueba nro. 2 Iteración 4 .....	128
Tabla 7-4: Prueba nro. 3 iteración 4 .....	128
Tabla 7-5: Riesgos mitigados en iteración 4.....	130
Tabla 8-1: Requerimientos funcionales iteración 5.....	131
Tabla 8-2: Prueba nro. 1 de iteración 5 .....	138
Tabla 8-3: Prueba nro. 2 de iteración 5 .....	139
Tabla 8-4: Riesgos mitigados en iteración 5.....	140
Tabla 9-1: Requerimientos funcionales iteración 6 .....	142
Tabla 9-2: Prueba nro. 1 de iteración 6 .....	145
Tabla 9-3: Riesgos mitigados en iteración 6 .....	146
Tabla 10-1: Prueba nro. 1 Sistema Final .....	155
Tabla 10-2: Prueba nro. 2 Sistema Final .....	155
Tabla 10-3: Prueba nro.3 Sistema Final .....	156
Tabla 10-4: Estado final de requerimientos funcionales.....	157
Tabla 10-5: Estado final de requerimientos no funcionales .....	158

# RESUMEN

En el presente proyecto se detalla el proceso de rediseño del sistema de sensores y de control del robot móvil omnidireccional Hermes II, el cual fué desarrollado como proyecto integrador de la carrera de grado Ingeniería en Computación de ésta misma Facultad a cargo de los ingenieros Ana Belén García Cabral y Sergio Javier Sagripanti.

Esta nueva versión, denominada Hermes III, cuenta con el mismo sistema de locomoción que su versión predecesora e incorpora mejoras sustanciales en cuanto al sistema de control que comanda el robot móvil, sentando las bases que a futuro servirán para seguir incorporando nuevas capacidades. En otras palabras, se incrementa la escalabilidad del sistema mediante el uso de lo denominado Sistema Operativo Robótico, comúnmente conocido por sus siglas en inglés **ROS** (Robot Operating System) que es un framework para el desarrollo de software para robots. Hoy, a diez años de su creación y dada su naturaleza Open Source, es una herramienta avalada, mantenida y utilizada por gran parte de la comunidad de desarrolladores de robots móviles.

Éste nuevo sistema operativo estará implementado sobre una nueva placa de desarrollo, diferente a su antecesora, denominada comercialmente NVIDIA Jetson TK1, la cual por sus especificaciones tecnológicas es capaz de acompañar e impulsar el crecimiento del proyecto.

Se expondrá entonces en éste informe, el proceso de rediseño e implementación de un robot móvil omnidireccional con las herramientas anteriormente mencionadas.

# CAPÍTULO 1

## 1 INTRODUCCIÓN

### 1.1 ESTADO DEL ARTE

La tecnología robótica se encuentra en plena fase de crecimiento y, por tanto, es previsible una presencia cada vez mayor de los robots, tanto en el sector industrial y de servicios, como en el sector doméstico. Como ha ocurrido en la informática, el desarrollo de robots se hace cada vez más accesible y por consiguiente sus capacidades tienden a ser cada vez mayores gracias al aporte de una comunidad en auge. Éste aumento en el interés de las personas sobre ésta área, plantea ciertos desafíos, no sólo en los aspectos tecnológicos, sino también desde el punto de vista cultural y ético.

En la última mitad del siglo pasado el foco estaba puesto sobre todo en el campo de la robótica industrial para impulsar el desarrollo de la producción a gran escala. Hoy se ha logrado gracias a ello un nivel tal de tecnología que el estudio y la realización de robots flexibles capaces de adaptarse a diferentes propósitos y a diferentes entornos de trabajo, ya no es una utopía. Desde el 2000, la investigación en el campo robótico también se dirige hacia el diseño de robots dedicados a la exploración del medio ambiente y de los llamados "robots de servicio".

Si hacemos foco en lo que respecta a los **robots exploradores móviles**, los mayores avances tecnológicos se dan en la exploración de otros planetas. Actualmente los científicos están enfocados en el planeta Marte, así como lo fue en su momento la superficie Lunar. Los primeros desarrollos de éste tipo de vehículos no tripulados surgen principalmente durante la carrera espacial entre Estados Unidos y la URSS en la década del 60/70, donde los más conocidos, exceptuando los exploradores lunares han sido los siguientes:

En la Figura 1-1 puede observarse al explorador Mars3 enviado en 1971 por la URSS el cual funcionó sobre la superficie marciana durante unos escasos 20 segundos.

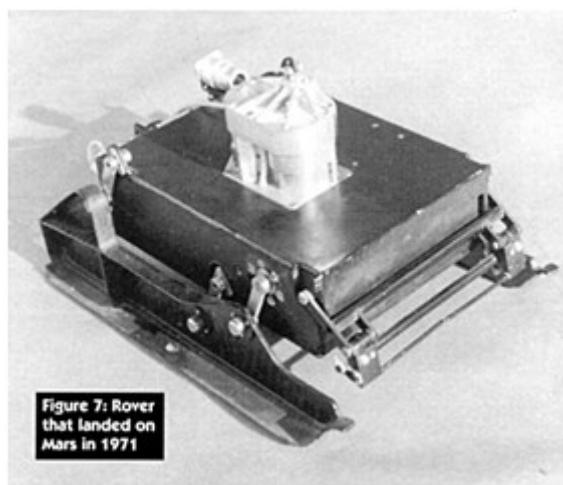


Figura 1-1: Mars 3

En 2003, la NASA envió dos vehículos idénticos de 174 kilogramos con seis ruedas y paneles solares para recorrer parte de la superficie marciana, el Spirit y el Opportunity (Figura 1-2). El primero transmitió datos que hacen pensar que en el pasado existió agua líquida en Marte, hasta que se atascó y dejó de funcionar. El Opportunity todavía está operativo y ostenta el récord de distancia recorrida en otro planeta, con más de 42,6 kilómetros hasta el momento.



Figura 1-2: Opportunity/Spirit

En 2011 la NASA envió a Marte el Curiosity (Figura 1-3), el vehículo más pesado (899 kilogramos) que nunca ha llegado a ese planeta. Aún está operativo y ha aportado las primeras evidencias de moléculas orgánicas en el planeta rojo.



Figura 1-3: Curiosity

El 14 de marzo de 2016 la Agencia Espacial Europea (ESA) y la rusa Roscosmos envian a Marte la misión ExoMars 2016 (Figura 1-4), la primera de dos misiones que buscan analizar la atmósfera marciana y colocar en su superficie dos vehículos, el segundo (2018) capaz de excavar dos metros por debajo de la superficie. [1]



Figura 1-4: ExoMars Rover

Por otra parte, tenemos a la **robótica de servicio** que es útil en el campo de la robótica médica, para la asistencia en rehabilitación y cirugía, así como en el campo doméstico para realizar tareas de limpieza y vigilancia.

La robótica móvil para estos sectores se ha convertido en una "herramienta de trabajo" importante y cada vez adquiere mayor protagonismo.

En los últimos años, mientras que la investigación de punta se enfoca en el diseño de robots para la exploración de Marte, en virtud de las tecnologías desarrolladas para tal fin, se hicieron los primeros robots para uso civil. Algunos de ellos, gracias a su pequeño tamaño y bajo costo de producción, gozan de cierto éxito comercial. Podemos mencionar de ejemplo al robot Roomba, hecho para la limpieza y lavado de pisos, una tarea que podría considerarse sencilla pero que implica un intenso trabajo computacional.



Figura 1-5: Roomba 980

El producto de la Figura 1-5 ha sido concebido gracias a ROS (Robot Operating System).

ROS se desarrolló originalmente en 2007 bajo el nombre de Switchyard por el Laboratorio de Inteligencia Artificial de Stanford para dar soporte al proyecto del Robot con Inteligencia Artificial de Stanford (STAIR: Figura 1-6). Desde entonces se han publicado numerosos Papers relacionados con el tema y existe una gran comunidad que avala y respalda el crecimiento de ésta herramienta dada su naturaleza Open Source.



Sea cual sea el ámbito de aplicación, el diseño de robots móviles de pequeño tamaño y bajo costo de producción, pero equipados con lógica de control avanzada, ha demostrado ser de gran utilidad para el hombre.

El proyecto Hermes III ilustra la concepción y construcción de un pequeño robot móvil. Se detallarán todas las etapas de desarrollo del mismo, partiendo de los componentes básicos hasta el producto terminado.

Todo proyecto robótico tiene por objeto crear un sistema complejo, cuyo desarrollo requiere conocimientos interdisciplinarios. El robot se coloca, de hecho, en la frontera entre la mecánica, la electrónica, el automóvil y la informática. [2]

**Figura 1-6: STAIR: STanford Artificial Intelligence Robot**

## 1.2 MOTIVACIÓN

A medida que fueron avanzando las investigaciones en robótica, se estableció como regla general que para que un robot móvil pueda auto navegar de forma exitosa debe ser capaz de responder a tres preguntas: ¿Dónde estoy?, ¿Hacia dónde voy? y ¿Cómo puedo llegar allí? [3]

Actualmente las investigaciones se enfocan en diversos campos que confluyen en la búsqueda de un objetivo común: el desarrollo de un sistema autónomo. En la Figura 1-7 se puede observar de una manera gráfica la estrecha interrelación que existe entre los principales campos y hasta qué punto el desarrollo de todos y cada uno de ellos es necesario para dotar de autonomía a un robot. [4]



Figura 1-7: Capacidades de un robot móvil

- Navegación: Capacidad de movimiento.  
Exploración: El robot puede desplazarse generando un mapa de su entorno.
- Mapeado: Capacidad de detectar profundidad de campo.  
SLAM (Simultaneous Localization and Mapping): El robot, teniendo conocimiento de la magnitud de sus movimientos traslacionales y contando con la capacidad de sentir profundidad de campo, puede construir un mapa de su entorno y ubicarse dentro del mismo.
- Localización: Capacidad de sentir sus movimientos y medirlos.  
Localización Activa: El robot puede moverse y conocer su posición en relación a un punto de partida.

## 1.3 OBJETIVO

El objetivo principal del proyecto es construir un robot, el cual posea un sistema de posicionamiento y sea capaz de realizar dinámicamente un mapa de la habitación donde se encuentre.

En primera instancia se atacarán los tres grandes campos que han sido mencionados en la sección anterior en la Figura 1-7: Capacidades de un robot móvil. Cada uno de estos campos (navegación, mapeado y localización) denotan capacidades específicas que se incorporarán al robot.

Una vez concretado lo anterior, se hará foco especialmente en la interrelación entre el mapeado y localización, tecnología que se denomina SLAM (Simultaneous Localization and Mapping). Para lograr ésto, es conveniente contar con un framework de desarrollo o un sistema operativo que nos otorgue herramientas específicas para tal fin. A la par de lo anterior, se deberá elegir una placa de desarrollo la cual soporte este sistema operativo y pueda correr las aplicaciones que serán desarrolladas.

---

### 1.3.1 OBJETIVOS PRINCIPALES

- Implementar un sistema operativo que nos brinde de herramientas específicas para el desarrollo de un robot móvil.
- Implementar el mapeo de una habitación en 2D.
- Implementar un Sistema de Posicionamiento Local (LPS).
- Implementar un Sistema de Control de trayectoria.

---

### 1.3.2 OBJETIVOS SECUNDARIOS

- Utilizar una placa de desarrollo adecuada para los requerimientos del proyecto.
- Reutilizar lo que fuese factible del proyecto Hermes II.
- Seleccionar las soluciones que se adapten a las necesidades del proyecto, realizando un análisis comparativo e indagatorio de las opciones que se tengan, en lo que respecta a:
  - Sistema operativo.
  - Sistema de sensado de profundidad de campo.
  - Sistema de sensado de movimiento.
  - Sistema de control de movimiento.
  - Fuente de alimentación eléctrica.

## 1.4 REQUERIMIENTOS

Los requerimientos funcionales son aquellas declaraciones de los servicios que debe proporcionar el sistema, cómo debe reaccionar ante determinados eventos y cómo debe comportarse en situaciones particulares.

Los requerimientos no funcionales son aquellos que no se refieren directamente a la funcionalidad del sistema sino a aquellas propiedades que emergen de ella como la fiabilidad, la capacidad de almacenamiento, disponibilidad, etc. También definen las restricciones del sistema como los dispositivos de entrada/salida y las interfaces de comunicación del sistema. [5]

También cabe destacar que los requerimientos cuentan con atributos, entre los que se destaca su prioridad. La misma será expresada en base a los siguientes términos:

Palabras que denotan la prioridad	Semántica
Debe	Requisitos obligatorios que son fundamentales para el sistema.
Debería	Requisitos importantes, pero que se pueden omitir.
Podría	Requisitos que son opcionales (se realizarán si se cuenta con el tiempo suficiente)
Quiere	Requisitos que pueden esperar para versiones posteriores del sistema final.

Tabla 1-1: Prioridad de requerimientos

### 1.4.1 REQUERIMIENTOS FUNCIONALES

Estos requerimientos han sido obtenidos por encuesta a Orlando Micolini (director del proyecto) y en base a investigaciones realizadas sobre documentos de robots con características similares a las del Hermes III.

ID	Descripción
RF1	Debe tener un sistema de locomoción omnidireccional
RF2	Debe poder auto localizarse
RF3	Debe poder realizar un mapa
RF4	Debe tener un sistema de control de movimiento eficaz
RF5	Debe poder controlarse inalámbricamente
RF6	Debería realizarse un modelo de simulación previo a la implementación
RF7	Debería tenerse un sistema de monitorización remoto de los sensores del robot
RF8	Podría realizarse un modelo tridimensional del robot
RF9	Quiere tener la capacidad de autonavegar

Tabla 1-2: Requerimientos funcionales

#### 1.4.1.1 PRUEBAS DE REQUERIMIENTOS FUNCIONALES

En la tabla siguiente se proponen las pruebas para verificar el cumplimiento de los requerimientos funcionales.

Código	Referencia	Prueba y Resultado Esperado
RF1	Omnidireccionalidad	Enviar un comando de movimiento en una determinada dirección. <u>Resultado esperado:</u> Movimiento rectilíneo en la dirección deseada e indicada en el comando.
RF2	Localización	<p>Etapa 1: 1) Conectar el sensor de profundidad y que el sistema embebido principal procese los datos de la misma. <u>Resultado esperado:</u> Se puede visualizar los datos del sensor de profundidad mediante algún software de monitorización.</p> <p>Etapa 2: 2) Mover el robot una distancia de 0.5 metros en línea recta en cualquier dirección, marcando en el suelo el punto de partida y de llegada que refleje dicha distancia. <u>Resultado esperado:</u> Se debe visualizar en el software de monitorización del robot una relación directamente proporcional entre el desplazamiento real y el virtual.</p> <p>3) Girar el robot 180° sobre su propio eje vertical. <u>Resultado esperado:</u> Se debe reflejar el movimiento de giro del robot de 180° en el software de monitorización.</p>
RF3	Mapeo	Poner en funcionamiento el robot en un ambiente cerrado y que recorra la mayor superficie posible del lugar. <u>Resultado esperado:</u> Debe obtenerse un mapa que refleje las proporciones reales del lugar.
RF4	Navegación	<p>Etapa 1: 1) Conectar los motores con sus respectivos <i>encoders</i> y que los datos que produzcan puedan ser procesados por el microcontrolador elegido. <u>Resultado esperado:</u> Se debe poder obtener la velocidad de una rueda.</p> <p>Etapa 2: 2) Poner en funcionamiento el robot sobre una línea recta señalada en el suelo enviándole un comando de movimiento en la misma dirección. <u>Resultado esperado:</u> El robot debe mantener un desplazamiento rectilíneo por encima de la línea marcada.</p>
RF5	Control Inalámbrico	Enviar comandos de movimiento al robot desde diferentes controles remotos, ya sea un joystick, un teclado de un sistema remoto, o un smartphone. <u>Resultado esperado:</u> El robot debe moverse en las direcciones y sentidos indicados por el comando de control.
RF6	Simulación	Modelar las propiedades físicas del robot en un software especializado, de tal manera de poder probar su dinámica de

		<p>movimiento.</p> <p><u>Resultado esperado:</u> El robot simulado debe tener un comportamiento dinámico y cinemático parametrizable.</p>
RF7	Monitorización	<p>1) Visualizar el modelo 3D realizado del Hermes 3 en el software de monitorización.</p> <p><u>Resultado esperado:</u> Se debe ver en el software de monitorización el modelo 3D en la ubicación actual del robot acorde a las coordenadas de ubicación suministradas por el mismo.</p> <p>2) Contrastar el comportamiento real del robot en funcionamiento con el comportamiento en el software de monitorización.</p> <p><u>Resultado esperado:</u> Los datos recolectados y visualizados deben ir a la par del movimiento del robot, siendo consistentes y coherentes con la realidad.</p>
RF8	Modelado	<p>1) Visualizar el diseño 3D creado en cualquier software CAD.</p> <p><u>Resultado esperado:</u> Se debe visualizar el modelo realizado en cualquier software CAD.</p> <p>2) Comparar el modelo 3D realizado con el software CAD frente al modelo que se construyó.</p> <p><u>Resultado esperado:</u> El modelo 3D y el modelo real deben tener similar aspecto y proporciones.</p>
RF9	Autonavegación	<p>Colocar el robot en una habitación desconocida y dejarlo funcionar autónomamente (sin recibir órdenes), hasta que se logre obtener el mapa completo de la habitación.</p> <p><u>Resultado esperado:</u> El mapa resultante debe ser coherente y consistente con la realidad. Además, durante la monitorización del robot, no deben detectarse choques bruscos contra obstáculos.</p>

Tabla 1-3: Pruebas de requerimientos funcionales

#### 1.4.2 REQUERIMIENTOS NO FUNCIONALES

Código	Descripción
Genéricos	
RNF1	Debe poderse interactuar con el robot desde múltiples plataformas (PC, Joystick, Smartphones)
RNF2	Debe contar con documentación de código y de proyecto
RNF3	Debe contar con pruebas y tests de funcionalidades
RNF4	Debería tener una carcasa con un diseño estructural robusto
RNF5	Debería tener tiempos de respuesta aceptables para un buen funcionamiento del sistema de control
Software	
RNF6	Debe utilizarse un Sistema Operativo específico para robótica
RNF7	Debería poderse programar en C++, Java o Python
RNF8	Debería utilizarse un Sistema Operativo y Framework OpenSource
Hardware	
RNF9	Debe tener suficientes entradas para sensores
RNF10	Debe tener suficientes salidas para actuadores
RNF11	Debe permitir el desarrollo modular
RNF12	Debería utilizarse componentes que cuenten con documentación

Tabla 1-4: Requerimientos no funcionales

## 1.5 ANÁLISIS DE RIESGOS

Para el análisis de los riesgos del proyecto, se ha decidido utilizar una estrategia de Mitigación, Monitorización y Gestión de Riesgos, conocido por sus siglas en inglés como RMMM (*Risk Mitigation, Monitoring and Management*). Para llevar a cabo cada una de las tareas anteriormente mencionadas, se ideará un plan de prevención, un plan de monitorización y un plan de contingencia respectivamente.

Se consideran riesgos aquellas situaciones que, en caso de suceder, afectan negativamente determinados aspectos del proyecto. Por ésta razón es importante identificarlos con anticipación para mitigarlos.



Figura 1-8: RMMM

El **plan de prevención** intenta ser un escudo para mantener la cantidad de riesgos latentes al mínimo y que no logren materializarse, por lo que su objetivo es entorpecer lo máximo posible el flujo a través del circuito de la Figura 1-8: RMMM.

El **plan de monitorización** busca detectar tempranamente las situaciones que pueden desencadenar la materialización de un riesgo en particular, para poder evitar, en caso de ser posible, que se concrete el riesgo.

El **plan de contingencia** hace a la gestión del riesgo, es decir, que una vez que se concreta un evento desafortunado asociado a un riesgo, se procede inmediatamente a solucionarlo y luego poder continuar con el proyecto. Las soluciones en ésta etapa no deben ser tomadas apresuradamente porque pueden incrementar el nivel de riesgos de todo el proyecto y hacerlo inestable. [6]

La ponderación de riesgos será calculada en base a la siguiente ecuación:

$$\text{Estimación de la probabilidad} \times \text{Estimación del impacto} = \text{Exposición al riesgo}$$

Ecuación 1.1: Exposición al riesgo

### 1.5.1 LISTADO DE RIESGOS

RI-01 Sistema Operativo inestable
<p><u>Condición:</u> Elección incorrecta del sistema embebido a usar y/o versión prematura de Sistema Operativo.</p>
<p><u>Consecuencias:</u> Interrupciones en el funcionamiento del robot, comportamiento errático, pérdida de enlaces de comunicación, reinicios inesperados, entre otros.</p>
<p><u>Efecto:</u> No se logrará cumplir con las expectativas mínimas de robustez y estabilidad que se espera de un robot autónomo.</p>

Tabla 1-5: RI-01 Sistema Operativo inestable

RI-02 Incompatibilidad o avería de componentes
<p><u>Condición:</u> Utilización de componentes electrónicos defectuosos o que al funcionar afectan negativamente a los demás componentes. También puede darse incompatibilidad de tecnologías, haciendo que directamente no pueda ponerse en funcionamiento un determinado componente.</p>
<p><u>Consecuencias:</u> Componentes inservibles, mediciones erróneas de sensores, daños parciales o totales de componentes.</p>
<p><u>Efecto:</u> Incremento del costo económico de fabricación del robot por tener que reemplazar los componentes incompatibles o dañados.</p>

Tabla 1-6: RI-02 Incompatibilidad o avería de componentes

RI-03 Intercomunicación de componentes inefficiente o ineficaz
<p><u>Condición:</u> Comunicación entre componentes lenta, con interferencias y/o con ruido.</p>
<p><u>Consecuencias:</u> Sistema robótico con tiempos de respuesta grandes que imposibilitan un eficaz funcionamiento de un sistema de control.</p>
<p><u>Efecto:</u> El robot no podrá reaccionar a tiempo para evitar obstáculos que detecte en su camino y podrá chocar contra ellos. Las respuestas del robot a los comandos de movimiento estarán muy desfasadas en el tiempo, lo que dificultará su manejo tanto manual como autónomo. A tiempos de respuesta mayores, menor será la velocidad máxima permitida de desplazamiento del robot.</p>

Tabla 1-7: RI-03 Intercomunicación de componentes inefficiente o ineficaz

RI-04 Prestaciones insuficientes de componentes
<p><u>Condición:</u> Sobreestimar las especificaciones técnicas de componentes. También puede ocurrir de subestimar las necesidades tecnológicas de la solución. Falta de documentación fiable de componentes.</p>
<p><u>Consecuencias:</u> Componentes que no son capaces de cumplir correctamente con su propósito y por ende pasan a ser inservibles para los fines del proyecto.</p>
<p><u>Efecto:</u> Incremento del costo económico del proyecto por el descarte y reemplazo de los componentes por otros de mayores prestaciones.</p>

Tabla 1-8: RI-04 Prestaciones insuficientes de componentes

#### RI-05 Modificación de los requerimientos del proyecto

Condición: Definición de nuevos requisitos o modificación de los existentes durante el desarrollo del proyecto.

Consecuencias: Replanificación de las tareas a realizar. Desarrollos en etapas avanzadas en los que se invirtió tiempo y esfuerzo pueden interrumpirse y quedar inservibles para los fines del proyecto.

Efecto: Retrasos en la finalización del producto final del proyecto.

Tabla 1-9: RI-05 Modificación de los requerimientos del proyecto

#### RI-06 Dificultad en conseguir determinados componentes

Condición: Componentes electrónicos que deben ser importados y provenientes de empresas que no trabajan con envíos al exterior. Altos costos de importación y/o envío. Grandes tiempos de demora en los envíos desde el exterior.

Consecuencias: Análisis de componentes alternativos que puedan ser utilizados para el mismo fin que el componente que no puede conseguirse. En última instancia podrían adaptarse los requerimientos del proyecto.

Efecto: Retrasos en la finalización del producto final del proyecto. Degradación de las capacidades del robot.

Tabla 1-10: RI-06 Dificultad en conseguir determinados componentes

#### RI-07 Excesivo tiempo para cumplir los objetivos del proyecto

Condición: Cualquier dificultad técnica que retrase significativamente el progreso del proyecto. Escasez de tiempo disponible de los integrantes del equipo de desarrollo.

Consecuencias: Robot con capacidades reducidas, menores a las esperadas.

Efecto: Directores del proyecto podrán readaptar los objetivos a alcanzar dando una extensión de tiempo más allá de lo planificado desde un comienzo.

Tabla 1-11: RI-07 Excesivo tiempo para cumplir los objetivos del proyecto

#### RI-08 Reducción de la fuerza de trabajo

Condición: Miembros del equipo que abandonen el proyecto.

Consecuencias: Las tareas a realizar llevarán más tiempo concretarlas.

Efecto: Retraso en la entrega final del proyecto.

Tabla 1-12: RI-08 Reducción de la fuerza de trabajo

### 1.5.2 ESTIMACIÓN DE PROBABILIDAD

En la siguiente tabla se exponen los criterios en base a los cuales se cuantificará la probabilidad de ocurrencia de los riesgos anteriormente mencionados.

Rango de Probabilidad	Promedio para el Cálculo	Expresión en Lenguaje Natural	Valor Numérico	Código de Color
1% a 20%	10%	Muy Baja Probabilidad	1	Blue
21% a 40%	30%	Baja Probabilidad	2	Green
41% a 60%	50%	Mediana Probabilidad	3	Yellow
61% a 80%	70%	Alta Probabilidad	4	Orange
81% a 99%	90%	Muy Alta Probabilidad	5	Red

Tabla 1-13: Cuantificación de la probabilidad

A continuación, se expresan los riesgos identificados para el proyecto con las probabilidades estimadas subjetivamente para cada uno de ellos.

ID	Riesgo	Probabilidad
RI-01	Sistema Operativo inestable	Baja
RI-02	Incompatibilidad o avería de componentes	Mediana
RI-03	Intercomunicación de componentes ineficiente o ineficaz	Alta
RI-04	Prestaciones insuficientes de componentes	Mediana
RI-05	Modificación de los requerimientos del proyecto	Baja
RI-06	Dificultad en conseguir determinados componentes	Muy Alta
RI-07	Excesivo tiempo para cumplir los objetivos del proyecto	Alta
RI-08	Reducción de la fuerza de trabajo	Muy Baja

Tabla 1-14: Probabilidad de ocurrencia de riesgos

### 1.5.3 ESTIMACIÓN DE IMPACTO

En la siguiente tabla se exponen los criterios en base a los cuales se cuantificará el impacto de los riesgos.

Criterio	Retraso en la Planificación	Valor Numérico	Código de Color
Insignificante	1 semana	1	Azul
Moderado	2 a 3 semanas	2	Verde
Medio	4 a 5 semanas	3	Ambar
Crítico	6 a 8 semanas	4	Naranja
Catastrófico	Más de 8 semanas	5	Rojo

Tabla 1-15: Cuantificación del impacto de los riesgos

A continuación, se expresa el impacto estimado en cuanto al máximo de tiempo que puede provocar la efectivización de un riesgo en el peor de los casos.

ID	Riesgo	Impacto
RI-01	Sistema Operativo inestable	Moderado
RI-02	Incompatibilidad o avería de componentes	Moderado
RI-03	Intercomunicación de componentes inefficiente o ineficaz	Medio
RI-04	Prestaciones insuficientes de componentes	Medio
RI-05	Modificación de los requerimientos del proyecto	Crítico
RI-06	Dificultad en conseguir determinados componentes	Crítico
RI-07	Excesivo tiempo para cumplir los objetivos del proyecto	Catastrófico
RI-08	Reducción de la fuerza de trabajo	Crítico

Tabla 1-16: Impacto en base a los diferentes riesgos

#### 1.5.4 EXPOSICIÓN AL RIESGO

ID	Riesgo	Probabilidad	Impacto	Exposición
RI-01	Sistema Operativo inestable	30%	2	0,6
RI-02	Incompatibilidad o avería de componentes	50%	2	1
RI-03	Intercomunicación de componentes ineficiente o ineficaz	70%	3	2,1
RI-04	Prestaciones insuficientes de componentes	50%	3	1,5
RI-05	Modificación de los requerimientos del proyecto	30%	4	1,2
RI-06	Dificultad en conseguir determinados componentes	90%	4	3,6
RI-07	Excesivo tiempo para cumplir los objetivos del proyecto	70%	5	3,5
RI-08	Reducción de la fuerza de trabajo	10%	4	0,4

Tabla 1-17: Exposición al riesgo

Magnitud de exposición al riesgo [7]:

- Aprox. 1 = bajo riesgo
- Aprox. 2 = riesgo medio
- Aprox. 3 = alto riesgo

### 1.5.5 MITIGACIÓN, MONITORIZACIÓN Y GESTIÓN DE RIESGOS (RMMM)

ID	Mitigación	Monitorización	Gestión
RI-01 	Investigar antecedentes de funcionamiento del sistema operativo en el sistema embebido elegido.	Pruebas periódicas de estrés del S.O en la medida que se agregan nuevas funcionalidades al robot.	Utilizar otra versión más estable o en su defecto utilizar otra distribución de S.O.
RI-02 	Investigar las aplicaciones y los resultados obtenidos de los componentes a utilizar, preferentemente del mismo fabricante para evaluar su calidad.	Pruebas de integración constantes del funcionamiento en la medida que se incorporan nuevas prestaciones.	Reemplazar los componentes averiados o incompatibles por otros de similares prestaciones.
RI-03 	Investigar protocolos de comunicación robustos y eficientes que puedan utilizarse.	Pruebas de estrés en las comunicaciones para calcular las velocidades máximas de intercambio de mensajes.	Refactorizar en lo posible los algoritmos de procesamiento de mensajes para hacerlos lo más eficientes posibles. En su defecto utilizar otros protocolos.
RI-04 	Estudiar la documentación de los componentes a utilizar, especialmente sus especificaciones técnicas y prestaciones límites.	Pruebas del robot en condiciones exigentes que podría afrontar en su normal funcionamiento.	Reemplazar los componentes por otros de mayores prestaciones.
RI-05 	Acordar y negociar correctamente los requerimientos del proyecto.	Monitorizar y cuantificar constantemente el nivel de avance sobre los requerimientos sin perder de vista los objetivos del proyecto.	Readaptar el plan de acción para resolver lo antes posible los nuevos requerimientos o las modificaciones de los mismos.
RI-06 	Estudiar previamente la situación del mercado para conseguir todos los componentes necesarios. Pueden utilizarse componentes alternativos de menores prestaciones temporalmente hasta conseguir aquel que se ajusta a las necesidades el proyecto.	Consultar periódicamente los stocks disponibles de los proveedores que se necesitan.	Comunicarse directamente con los proveedores. En su defecto buscar componentes alternativos de otros fabricantes
RI-07 	Estudiar en profundidad los riesgos y dificultades que deberán afrontarse acorde a los objetivos y requerimientos definidos del proyecto.	Monitorizar si los tiempos que llevan concretar las tareas se ajustan a lo que se ha previsto al comienzo del proyecto.	Renegociar los objetivos y requerimientos del proyecto.
RI-08 	Tener una comunicación fluida entre los miembros del equipo.	Charlar constantemente entre los miembros del equipo de las dificultades personales y profesionales que se puedan tener para llevar a cabo las tareas.	Negociar extensión en el plazo de entrega del proyecto terminado o en su defecto buscar nuevo compañero de equipo.

Tabla 1-18: Plan de mitigación, monitorización, y gestión de los riesgos

## 1.6 ARQUITECTURA PRELIMINAR DE ALTO NIVEL DEL SISTEMA

En base a los requerimientos funcionales y no funcionales planteados anteriormente, se bosqueja el siguiente diagrama de arquitectura preliminar, indicando al pie de cada componente el requerimiento principal que satisface.

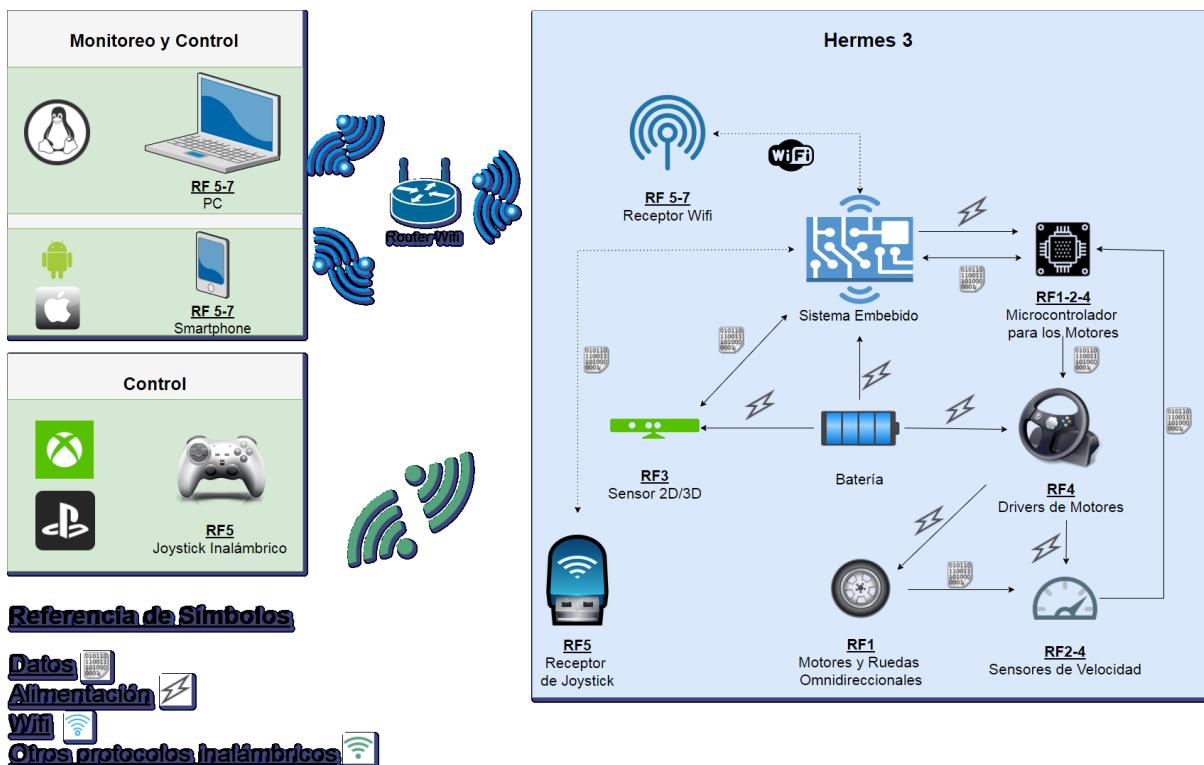


Figura 1-9:Arquitectura del sistema

En la Figura 1-9 pueden diferenciarse dos grandes grupos, el robot Hermes III a la derecha en el recuadro azul y los dispositivos de monitoreo y control a la izquierda en los recuadros verdes.

Lo que se desea mostrar es la interrelación entre todos los componentes que formarán parte necesariamente del sistema final para poder cumplir con todos los objetivos y requerimientos planteados anteriormente.

En ningún caso se especifican marcas o modelos exactos de los elementos a utilizar en el sistema, pero implícitamente todos deben ser compatibles entre sí. Por lo tanto, se planea definir en primera instancia el Sistema Embebido a utilizar que será el centro de interconexión del robot y luego se irán determinando los demás componentes para que se acoplen correctamente al mismo. A lo largo del informe se irá dilucidando este orden y las funcionalidades específicas que se buscan de cada componente.

# CAPÍTULO 2

## 2 MARCO TEÓRICO

### 2.1 METODOLOGÍA DE DESARROLLO

Una buena elección de la metodología de desarrollo ahorra tiempo y mejora la calidad de los sistemas que se producen. Por ende, es importante conocer las metodologías que existen y seleccionar aquella que mejor se ajuste a los requerimientos del proyecto. Podemos distinguir dos grandes grupos de técnicas, las tradicionales y las ágiles. [5]

- Modelos Tradicionales

- Cascada: Es el enfoque metodológico que ordena rigurosamente las etapas del proceso para el desarrollo, de tal forma que el inicio de cada etapa debe esperar a la finalización de la etapa anterior (Ver Figura 2-1).

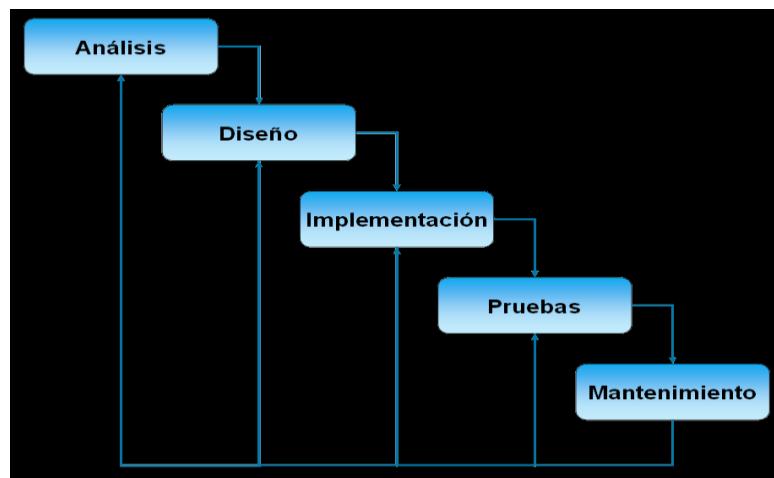


Figura 2-1: Desarrollo en cascada

- Espiral: Esta metodología ideada por Boehm debe sus orígenes a la necesidad de flexibilidad a la hora de realizar un proyecto, característica de la que carece el modelo en cascada. Consiste en ir desarrollando versiones incrementales, partiendo de la planeación de una iteración, luego definiendo pequeños objetivos, consiguientemente analizando riesgos y por último desarrollando y probando. Al finalizar, el ciclo comienza nuevamente, pudiendo agregar o modificar funcionalidades en la medida que lo requiera el destinatario del producto. En las primeras etapas, el cambio de requerimientos es económico en términos de esfuerzo

por el avance progresivo, pero en la medida que se avanza, las modificaciones del rumbo de desarrollo son cada vez más costosas (Ver Figura 2-2).

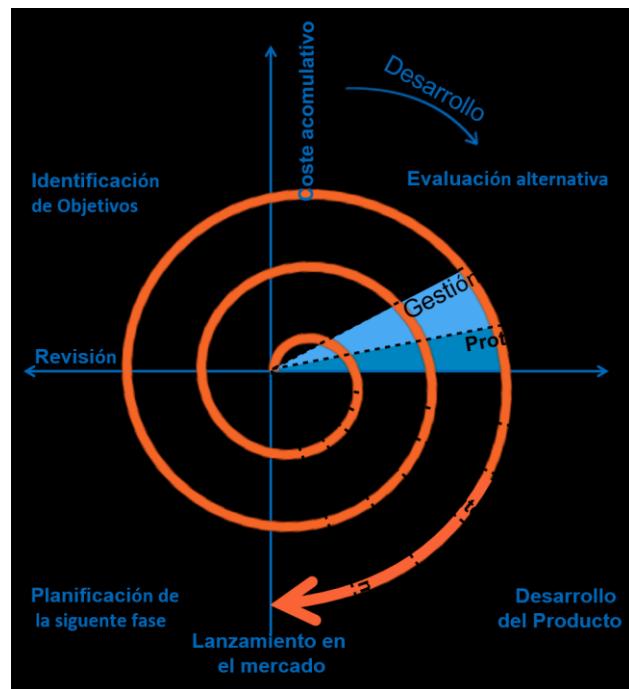


Figura 2-2: Desarrollo en espiral

- **Basada en componentes:** Este modelo se basa en la reutilización, y tiene la ventaja de reducir la cantidad de software que se debe desarrollar y por ende reduce los costos y los riesgos. Aplicar éste método requiere tener una visión clara a futuro de lo que desea obtenerse y desarrollar software flexible y reutilizable de calidad para hacer de éste método eficiente y económico en cada iteración. Adoptar malos patrones de diseño puede ser contraproducente (Ver Figura 2-3).

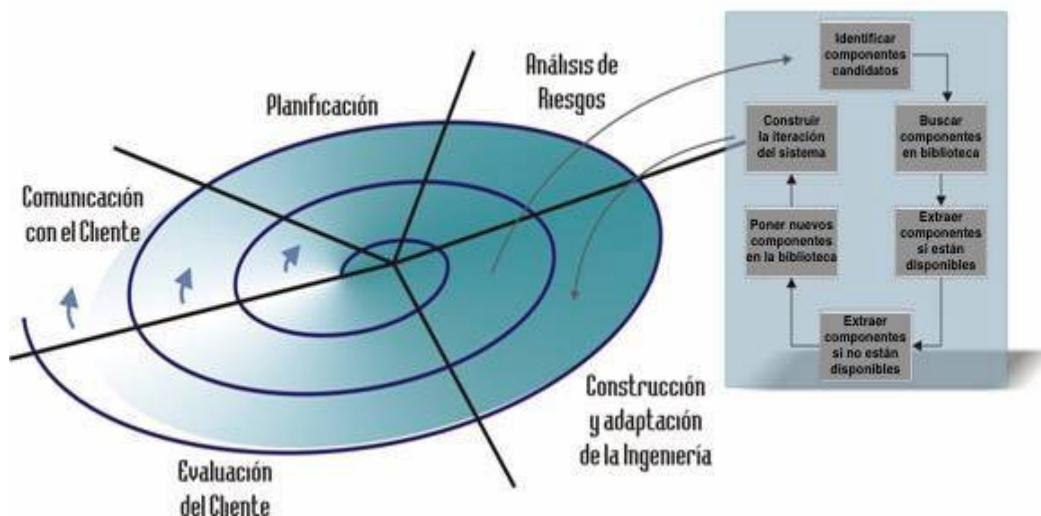


Figura 2-3: Desarrollo basado en componentes

- Modelos Ágiles

- **Incremental:** Consiste en dividir el trabajo en iteraciones bien definidas. En cada una se implementan nuevas funcionalidades o se mejoran las existentes, con el objeto de que el usuario final pueda ir obteniendo beneficios del proyecto de manera creciente.

Para llegar a lograr esto, cada requerimiento debe tener un completo desarrollo en una única iteración que debe incluir pruebas y una documentación para sentar las bases para la siguiente iteración.

Este modelo se divide en 4 etapas por iteración, las cuales son Análisis, Diseño, Programación, y Pruebas. La última iteración se consigue cuando ya se cumplen los objetivos y requerimientos del proyecto (Ver Figura 2-4). [8]



Figura 2-4: Desarrollo incremental

#### 2.1.1 ELECCIÓN DEL MODELO

Como factor determinante en la decisión de la metodología a utilizar en el presente proyecto, se ha considerado lo siguiente:

- En las metodologías tradicionales, importa mucho el **cómo** se lograrán los objetivos, haciendo mucho hincapié en la aplicación de patrones de diseño predefinidos. Esta estructuración permite tener un equipo de desarrollo de muchos integrantes, incluso de manera distribuida y en muchos casos ésto es útil para proyectos de gran envergadura poco modulares.
- En las metodologías ágiles, el cómo lograr los objetivos no es tan importante como sí lo es obtener **resultados**. Ésto hace que el desarrollo no siga precisamente patrones predefinidos de diseño, sino más bien dependa de las decisiones de los miembros más experimentados del equipo. Por ésta razón éstos métodos son recomendados para pequeños equipos, con muy buena comunicación. Por lo general permite afrontar proyectos de poca envergadura, aunque cabe aclarar, que también pueden servir para grandes proyectos divididos correctamente en módulos donde cada pequeño equipo se encarga de uno de ellos aplicando metodologías ágiles.

Debido al tiempo medio que comúnmente conlleva la realización de un proyecto integrador de carrera en ingeniería y dada la envergadura del presente trabajo, se considera que la mejor opción es la de aplicar un **modelo ágil incremental**. Ésta decisión se basa en los siguientes fundamentos:

- El equipo de desarrollo es pequeño y contará con muy buena comunicación.

- Al finalizar cada una de las etapas iterativas, podrá obtenerse un resultado funcional que será expuesto ante los directores del proyecto, que evaluarán los avances y brindarán retroalimentación que enriquecerá las consiguientes iteraciones.
- Al no contar con un patrón de diseño riguroso a aplicar dentro de cada iteración, se tendrá la libertad de implementar los procedimientos de desarrollo que mejor se adapten a las capacidades técnicas y tecnológicas del equipo.

## 2.2 ROBOTS

### 2.2.1 SISTEMAS MECÁNICOS DE LOCOMOCIÓN

Existen una gran variedad de sistemas de locomoción, pero a continuación sólo se exponen aquellos que utilizan ruedas. Entre ellos podemos encontrar:

- Diferencial
- Síncrona
- Triciclo
- Ackerman
- Omnidireccional

#### 2.2.1.1 LOCOMOCIÓN DIFERENCIAL

Este tipo de locomoción se destaca por no contar con ruedas directrices, por lo que, el cambio de dirección se realiza modificando la velocidad relativa de las ruedas izquierda y derecha.

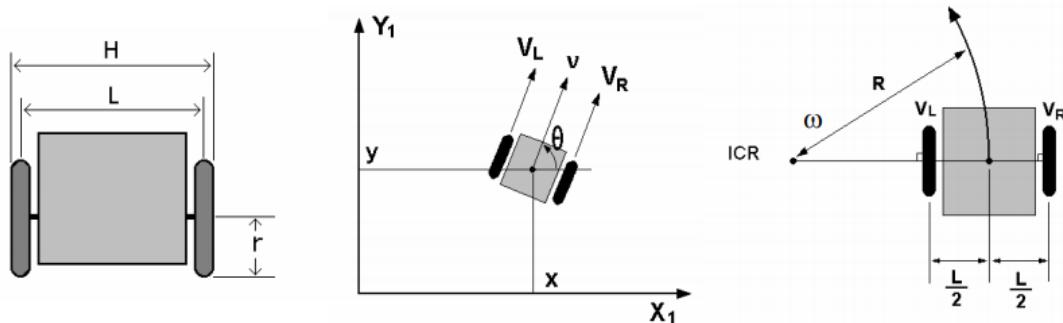
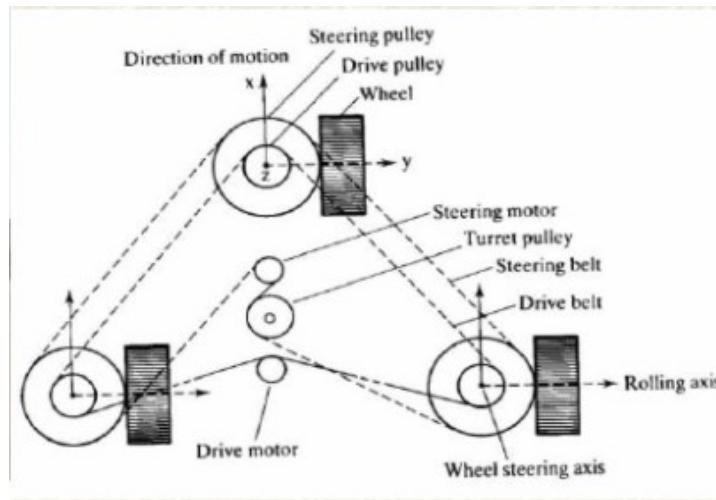


Figura 2-5: Movimientos de locomoción diferencial

Las principales ventajas de este sistema es que tiene un diseño simple, barato, y fácil de implementar. Pero se puede tener como contra que es difícil de controlar y requiere control de precisión para líneas rectas (Ver Figura 2-5).

#### 2.2.1.2 LOCOMOCIÓN SÍNCRONA

Este sistema de locomoción cuenta con tres o más ruedas que se direccionan coordinadamente. La coordinación puede ser mecánica o electrónica, y los motores separados para translación y rotación simplifican el control, además de que el control en línea recta está garantizado.

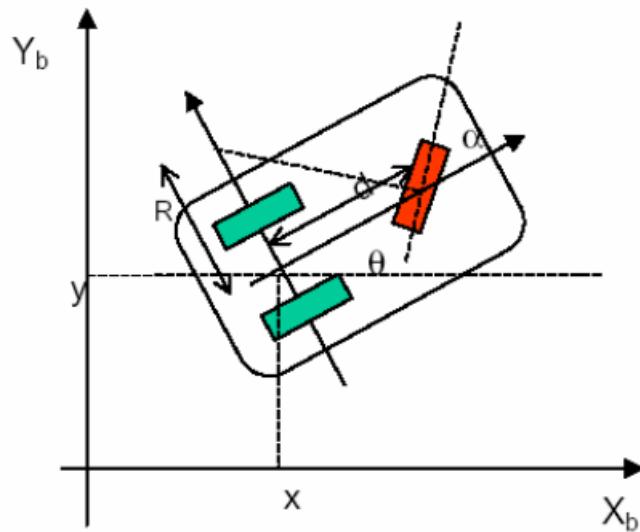


**Figura 2-6: Movimientos de locomoción sincrona**

Este sistema de locomoción no cuenta con restricciones holonómicas, pero su implementación y diseño es muy complejo (Ver Figura 2-6).

#### 2.2.1.3 TRICICLO

En este sistema de locomoción se cuenta con una tercera rueda para la dirección, y la tracción se puede dar o en la rueda de dirección o en las dos ruedas auxiliares, por lo que la tracción puede ser o no diferencial.



**Figura 2-7: Movimientos de triciclo**

Este sistema es no holonómico, y además tiende a ser inestable (Ver Figura 2-7).

#### 2.2.1.4 LOCOMOCIÓN DE ACKERMAN

La locomoción de Ackerman es la más conocida y más difundida por ser la implementada por los automóviles. Consiste en dos ruedas traseras y dos ruedas delanteras las cuales proveen la dirección del robot. La tracción puede estar dada por las ruedas delanteras, traseras o ambas.

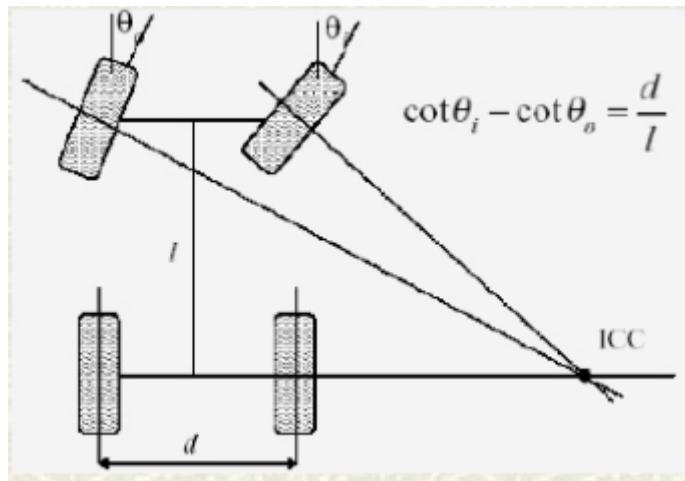


Figura 2-8: Movimientos de locomoción de Ackerman

Este sistema tiene la ventaja de su fácil implementación, con un sistema simple para el control de dirección. Pero como desventaja podemos mencionar que se trata de un sistema no holonómico (Ver Figura 2-8).

#### 2.2.1.5 LOCOMOCIÓN OMNIDIRECCIONAL

Este sistema es una extensión del caso diferencial, en donde se tienen más de dos ruedas, las cuales poseen la particularidad de poder desplazarse en cualquier dirección. Con este tipo de locomoción, todas las ruedas traccionan, pudiendo representar matemáticamente ésta fuerza en vectores, donde la resultante termina siendo la dirección y el sentido de movimiento del robot móvil. Esto es una ventaja ya que su complejo diseño permite mayor libertad de movimientos que los sistemas de ruedas clásicos. A éste tipo de vehículos se los conoce como holonómicos.

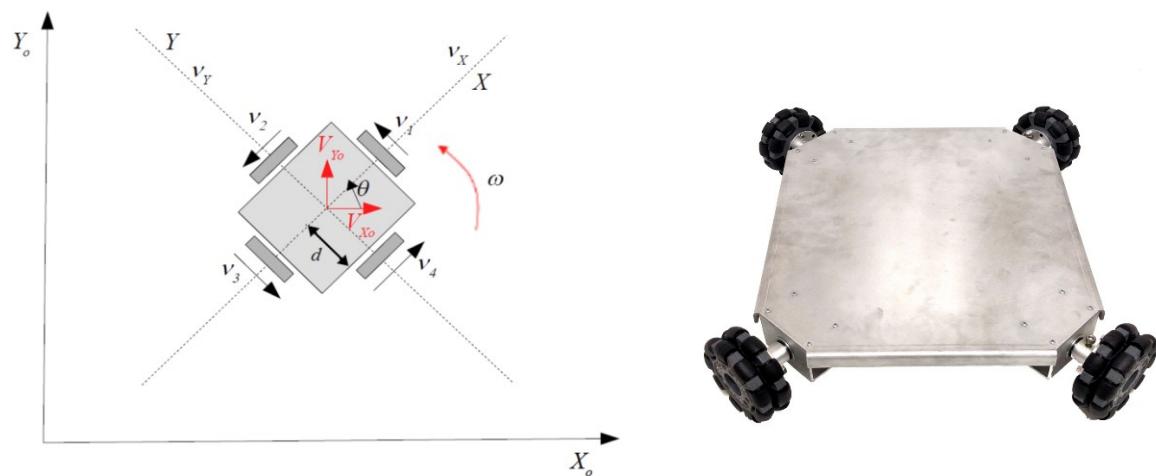


Figura 2-9: Movimientos de locomoción omnidireccional

El inconveniente de este sistema, más allá de su complejo diseño, es que el movimiento en línea recta no está garantizado por sus propiedades mecánicas (Ver Figura 2-9). [9]

## 2.2.2 RESTRICCIONES HOLONÓMICAS Y NO-HOLONÓMICAS

Los robots móviles pueden dividirse en dos grandes grupos, holonómicos y no holonómicos. Es una distinción que está relacionada con la capacidad de desplazamiento. En simples palabras, los sistemas holonómicos son aquellos capaces de modificar su dirección instantáneamente (considerando masa nula), y sin necesidad de rotar previamente. Un vehículo con un sistema de dirección como el de un coche, por ejemplo, no lo es, porque para poder desplazarse en el sentido lateral tiene que realizar varias maniobras previas. Del mismo modo, un robot con 2 ruedas (diferencial) es no-holonómico ya que no puede moverse hacia la izquierda o la derecha. Siempre lo hace hacia adelante en la dirección definida por la velocidad de sus ruedas.

La principal ventaja de un sistema omnidireccional es su maniobrabilidad. Una forma de conseguir este tipo de tracción/dirección es mediante el uso de ruedas omnidireccionales. Un modelo de las mismas se puede apreciar en Figura 2-10.



Figura 2-10: Rueda omnidireccional

Una de las ventajas de un sistema de dirección holonómico frente al sistema de dirección tradicional es que su control es mucho más intuitivo.

Si bien los conceptos anteriores parecen sencillos, su implementación no lo es. Al realizar el modelo matemático de un robot con ruedas omnidireccionales se deben hacer cálculos vectoriales para obtener la variable de control que es la dirección y el sentido de desplazamiento resultante del movimiento de las ruedas en su conjunto. [10]

## 2.3 COMPONENTES

Para un robot autónomo, es fundamental contar con los sensores suficientes para tener conocimiento de su estado (posicionamiento) y de su entorno (mapeo).

### 2.3.1 MAPEO

En el ámbito industrial de la robótica, que se rige por especificaciones estrictas y altos estándares de calidad, se acostumbra usar sensores 2D (laser range-finder) como se muestran en la Tabla 2-1, los cuales son productos utilizados en la robótica para detectar distancias desde el sensor hasta un objeto, y que por sus fines específicos para lo que son diseñados, cuentan con una alta precisión en sus mediciones.



Tabla 2-1: Lasers Range-Finder

Comúnmente en el desarrollo de prototipos robóticos, se utilizan sensores de profundidad 3D por el hecho que son productos de bajo costo fabricados a gran escala, en gran parte debido a que son usados en la industria de los videojuegos para el reconocimiento de gestos y movimientos corporales.

Dado que, a partir de un sensado en 3 dimensiones, definiendo un eje paralelo al suelo, podemos obtener datos en un plano de 2 dimensiones y sus resultados pueden considerarse aceptables para el reconocimiento de obstáculos y creación de un mapa de una habitación, vamos a optar por éste tipo de sensores.

Las alternativas más populares son:

	<b>Asus Xtion</b>	<b>Xbox Kinect</b>
Sensor	Profundidad/RGB	Profundidad/RGB/Micrófono
Campo de visión	58° H, 45° V	57° H, 43° V
Movimiento vertical	No	±27°
Distancia de Uso	Entre 0.8 y 3.5 metros	Entre 1.2 y 3.5 metros
Consumo energético	Menor a 2.5W	12W
Fuente de energía	USB	Necesita fuente externa, ya que USB proporciona 5W
Interfaz	USB 2.0	USB 2.0
Ambiente de operación	Interno	Interno
Dimensiones	18 x 3.5 x 5 (LxWxH)	27,94 x 7,62 x 7,62 (LxWxH)
Precio	\$2789 (Pesos Argentinos) (Noviembre 2016)	\$2890 (Pesos Argentinos) (Noviembre 2016)
Disponibilidad en el país	Baja	Alta

**Tabla 2-2: Sensores de profundidad 3D**

En cuanto a especificaciones de consumo energético el dispositivo Asus Xtion es el idóneo para proyectos de robótica basados en baterías, pero tiene un gran punto en contra que es su baja disponibilidad en el mercado local, haciendo difícil y/o lenta su adquisición.

Siendo de crucial importancia para el avance del proyecto, y considerando el alto nivel de exposición al riesgo Tabla 1-10: RI-06 Dificultad en conseguir determinados componentes) analizado previamente, se ha decidido utilizar el sensor Microsoft Kinect, considerando que existen numerosos prototipos robóticos que lo incluyen, numerosas guías para su uso y se tiene un completo soporte de software para el mismo para diferentes plataformas. Además, cuenta con ciertas características de las

que puede beneficiarse el proyecto “Hermes” en un futuro, como ser el micrófono y el motor de movimiento vertical.

### 2.3.1.1 KINECT

El Kinect es un dispositivo utilizado para los videojuegos, el cual cuenta con distintos tipos de cámaras y sensores para realizar la captura de movimientos en tiempo real. Este dispositivo fue desarrollado por Microsoft para sus consolas de videojuegos Xbox 360, y este es un detalle no menor, ya que implica que el producto se comercializa en distintas partes del mundo, a precios accesibles, contando con sensores avanzados, que, si no fueran ampliamente utilizados por los videojuegos, serían más costosos y más difíciles de conseguir.

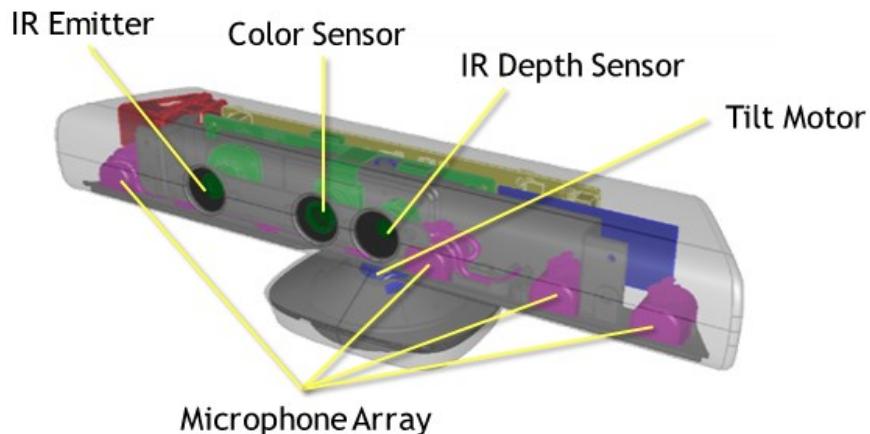


Figura 2-11: Componentes de Microsoft Kinect

Como se puede ver en la Figura 2-11, el Kinect cuenta con una cámara RGB con una resolución de 640x480, la cual hace posible la captura de imágenes a color. Cuenta con un emisor de infrarrojos que emite un patrón que es captado por su cámara infrarroja para el cálculo de la distancia entre un objeto y la lente. Con esto es capaz de calcular y construir un mapa de profundidades.

Entrando más en detalle, el patrón lumínico emitido por su emisor infrarrojo (*IR Emitter*), se lo conoce como *luz estructurada*. Un ejemplo de la misma se muestra en la Figura 2-12.



Figura 2-12: Luz estructurada

La luz estructurada, lo que hace es proyectar un patrón conocido sobre una superficie, y en base a la deformación de ese patrón, puede obtener la profundidad. Entonces, el emisor de infrarrojos emite un patrón de puntos distribuidos aleatoriamente, en donde cada grupo de puntos debe ser distingible de otro grupo de puntos en la misma fila. Luego, la cámara infrarroja captura el patrón de puntos deformados por la geometría de la escena, y el procesador interno de la Kinect es el encargado de calcular la profundidad por el desplazamiento entre el frame de referencia y la imagen actual (Ver Figura 2-13). [11]

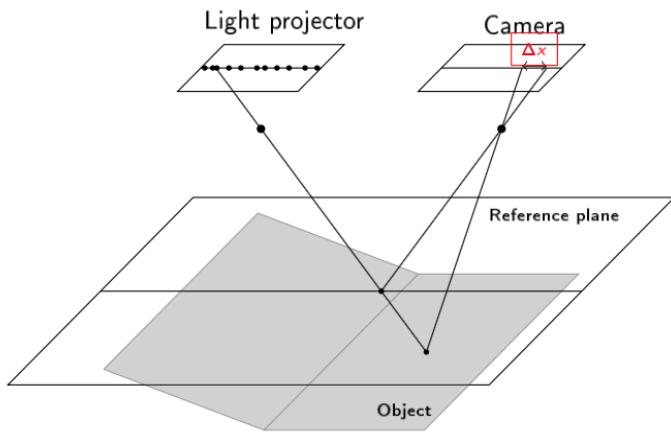


Figura 2-13: Cálculo de distancia en luz estructurada

A su vez, la luz estructurada trabaja conjuntamente con dos técnicas de visión computacional:

- **Profundidad de enfoque:** La profundidad de enfoque se basa en el principio de que las cosas que se ven más borrosas, son las que están más lejos. La Kinect mejora notablemente la precisión de la profundidad de enfoque tradicional. Utiliza una lente especial "astigmática" con una distancia focal diferente en las direcciones x e y, por lo que un círculo proyectado luego se convierte en una elipse dependiendo de la profundidad (Ver Figura 2-14).

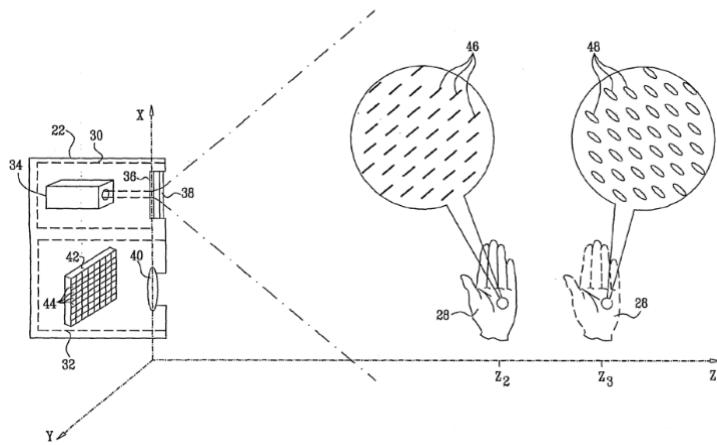


Figura 2-14: Profundidad de enfoque

- **Profundidad de estéreo:** La profundidad de estéreo utiliza paralaje. Esto quiere decir que, si se mira la escena desde otro ángulo, las cosas que están cerca se desplazan más hacia un lado que cosas que están más lejos. La Kinect analiza el cambio del patrón de puntos proyectándose desde un lugar y observando desde otro, como se muestra en la Figura 2-13.

El rango de visión de la Kinect es de 57° horizontal y 43° vertical, con motores en su base que permiten un movimiento que aporta ±27° de visión vertical, y sus cámaras son capaces de capturar 30 frames por segundo.

Éste sensor disponibiliza los datos en forma de **arrays**, uno que describe la profundidad y otro que describe el color de cada pixel capturado en el rango de visión.

### 2.3.2 POSICIONAMIENTO

Un robot móvil que solamente posee sensores de profundidad es capaz de evadir obstáculos, pero difícilmente podrá construir un mapa coherente de la habitación donde se encuentre sin datos de posición, ya que los datos que adquiere a través de dichos sensores cada determinado intervalo de tiempo, necesitan interrelacionarse a través de una variable que es la magnitud del desplazamiento del robot.

Para explicar lo anterior, daremos el siguiente ejemplo mediante la Figura 2-15 que representa datos de profundidad tomados por el robot en dos instantes de tiempo diferentes.



Figura 2-15: Sensado de Microsoft Kinect dos instantes de tiempo diferentes

Si sólo tuviéramos los datos que ha capturado el robot en ambos instantes de tiempo, pero no tenemos información acerca de la distancia recorrida por el robot durante el lapso de tiempo que separa a las dos muestras, no tendremos forma de construir un mapa coherente en base a estos datos no relacionados.

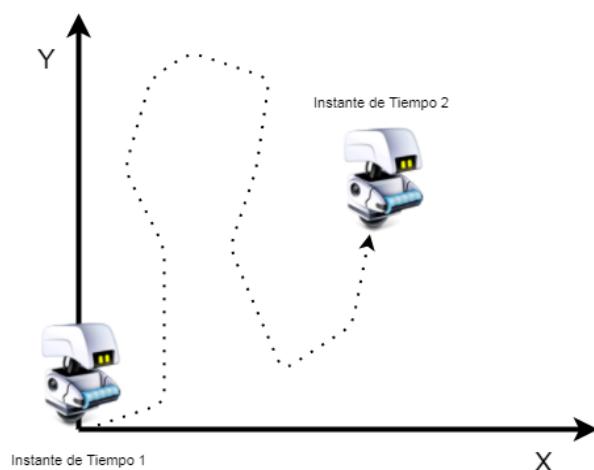
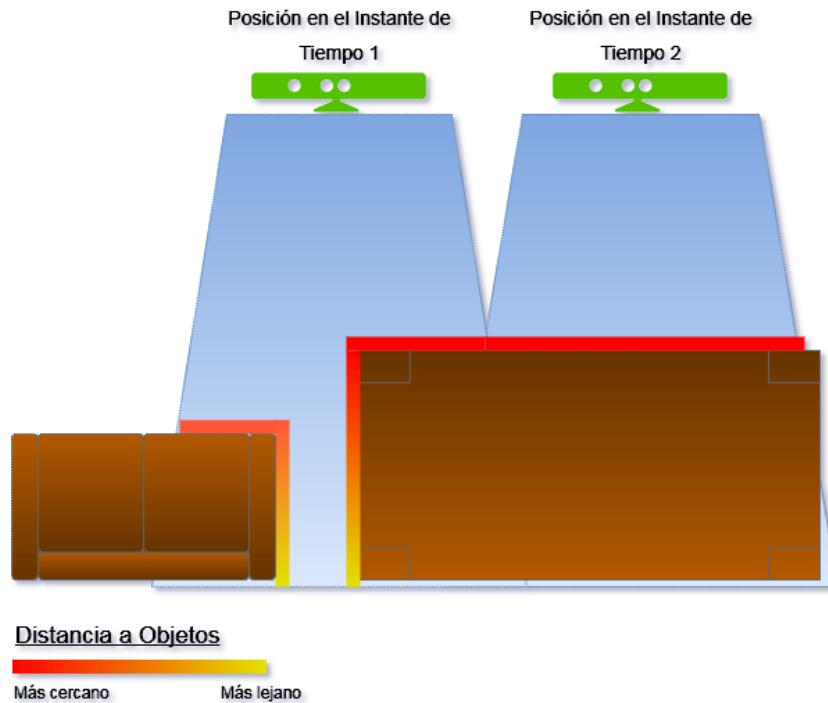


Figura 2-16: Trayectoria de robot en un plano 2D

Ahora bien, si incorporamos al robot la capacidad de odometría, el mismo podrá tener información de la magnitud de sus movimientos en un plano 2D, lo que le permitirá conocer su

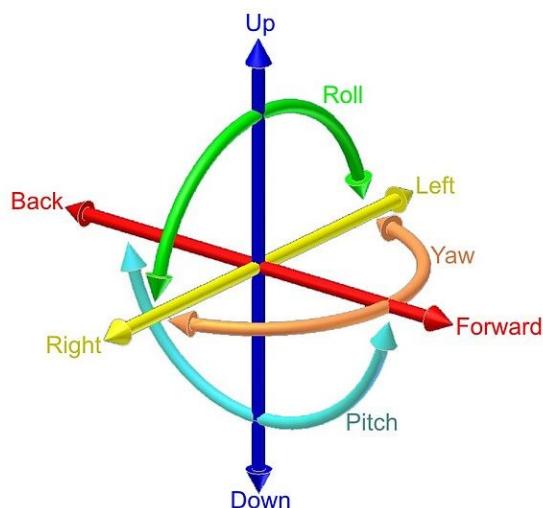
trayectoria y por ende su punto de partida (origen del plano 2D de todo el sistema de posicionamiento y mapeo) y su punto de llegada. De ésta manera, ahora los datos sensados de profundidad tendrán un plano de referencia sobre el cual ubicarse de acuerdo a la posición y ubicación del robot al momento de cada captura, pudiendo construir un mapa coherente y consistente (Ver Figura 2-16).

Contando con información de posición ahora la Figura 2-15 puede interrelacionar los datos y construir un mapa en consecuencia, tal como se muestra a continuación en la Figura 2-17.



**Figura 2-17: Mapa elaborado con distancia a objetos**

En términos matemáticos, si consideramos que el robot tendrá seis grados de libertad, deberá expresarse la ubicación del robot móvil en el espacio a través de tres ejes perpendiculares para la translación ( $x$ ,  $y$ ,  $z$ ) y en tres ejes perpendiculares para la rotación ( $\Theta_x$ ,  $\Theta_y$ ,  $\Theta_z$ ), representados en la Figura 2-18.



**Figura 2-18: Movimientos sobre seis grados de libertad**

En el caso de robots terrestres, diseñados para moverse principalmente sobre superficies llanas, la representación matemática anterior puede simplificarse considerando un espacio bidimensional, donde el robot sólo tendrá 3 grados de libertad, ésto es, 2 ejes perpendiculares para expresar la traslación y 1 eje (Yaw) para expresar su giro sobre su eje vertical. Éste se representa en la Figura 2-19.

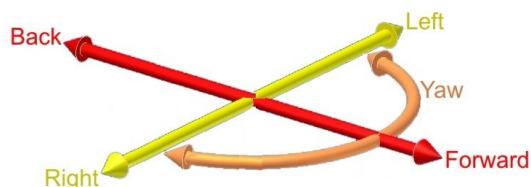


Figura 2-19: Grados de libertad para un robot terrestre

### 2.3.2.1 SENSORES OPTOACOPLADORES

Los sensores optoacopladores (Figura 2-20) son ampliamente utilizados para traducir posiciones o patrones de movimiento de objetos en datos binarios, de modo que se pueden implementar como encoders.

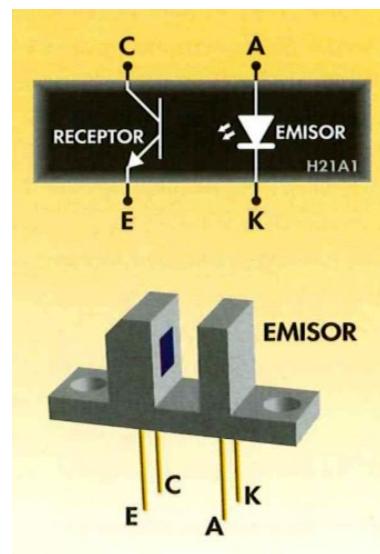


Figura 2-20: Sensor octocoplador

Comúnmente en el mercado, éstos sensores se consiguen en forma de horquilla, con un diodo emisor y con uno receptor enfrentado a él, que cuando se posiciona un cuerpo entre el emisor y receptor, impide que el fototransistor reciba energía lumínica y por ende no conduce, generando a su salida un 0 lógico. Caso contrario, cuando no hay un obstáculo entre medio del emisor y receptor, el fototransistor recibe energía lumínica, y transmite un 1 lógico.

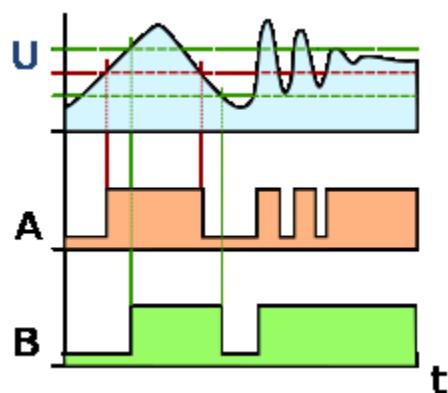


Figura 2-21: Sensado de un sensor octocoplador

Además, se suele agregar un circuito adicional, que filtra ruidos y permite obtener una onda cuadrada, tal como se muestra en la Figura 2-21.

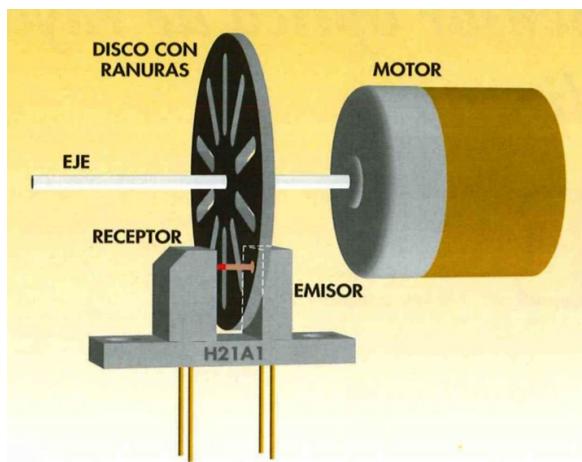


Figura 2-22: Encoder

La aplicación más sencilla para la medición de velocidad, consiste en registrar mediante un circuito contador de eventos, la cantidad de transiciones (por flanco positivo o negativo) entre ceros y unos lógicos que representan el paso por cada una de las ranuras del disco ranurado (implementado según Figura 2-22). Los datos obtenidos de la cantidad de transiciones con respecto al tiempo, la cantidad de ranuras del disco y las dimensiones de la rueda a la que se encuentra acoplado, nos permiten calcular la velocidad de traslación resultante.

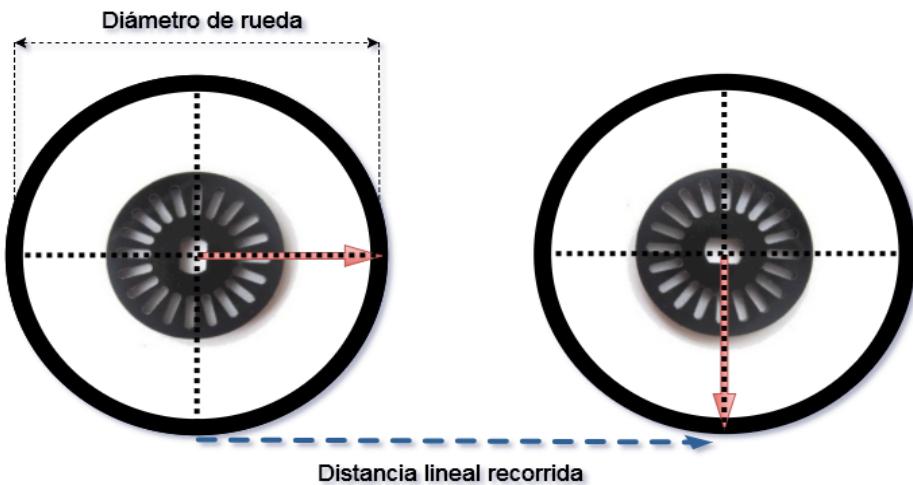


Figura 2-23: Distancia en base al radio de una rueda

Considerando que el disco encoder cuenta con 20 ranuras (como se muestra en la Figura 2-23), la cantidad de ranuras medidas por intervalo de tiempo puede transformarse en velocidad de la siguiente manera:

$$\frac{\text{ranuras contadas}}{\text{cantidad total de ranuras} * (\pi * \text{diámetro})} * \frac{1}{\text{intervalo de tiempo}} = \text{velocidad} = \frac{\text{distancia lineal recorrida}}{\text{intervalo de tiempo}}$$

Ecuación 2.1: Calculo de velocidad

La resolución es la mínima variación de la magnitud medida que da lugar a una variación perceptible de la indicación correspondiente. [12]

La resolución del sistema de medición se calcula mediante la siguiente ecuación:

$$\frac{360}{\text{cantidad de ranuras}} = \text{resolución angular}$$

Ecuación 2.2: Resolución angular

A su vez ésta resolución angular se traduce en resolución lineal multiplicándolo por la circunferencia de la rueda medida:

$$\text{resolución angular} * \text{circunferencia} = \text{resolución lineal}$$

Ecuación 2.3: Resolución lineal

La resolución lineal será el mínimo desplazamiento por rueda que podrá ser detectado.

### 2.3.3 FUENTE DE ALIMENTACIÓN ELÉCTRICA

Una batería (Figura 2-24) es un dispositivo que almacena energía química que puede transformarse en energía eléctrica. Las baterías pueden estar formadas por una sola celda electroquímica, o varias celdas conectadas en serie, aumentando el voltaje resultante como así también la potencia eléctrica (Watts) capaz de entregar. Cuando la reacción química es reversible, es decir que, al suministrarle energía eléctrica, ésta se transforma en energía química, y sin que se produzca una degradación inmediata del sistema, nos encontramos con las baterías recargables, las cuales constituyen indudablemente hoy en día una fuente de energía práctica y eficaz.

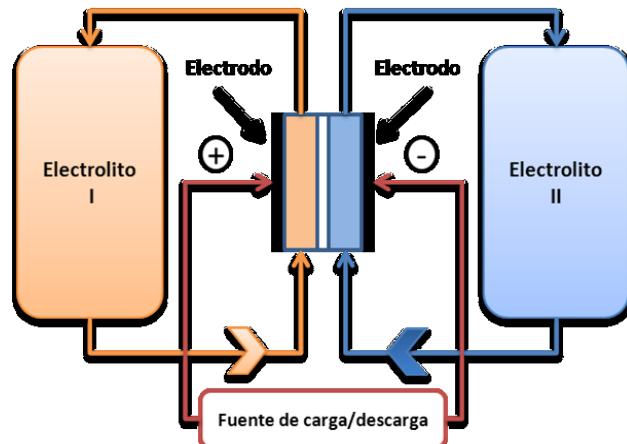


Figura 2-24: Batería recargable

Entre los distintos tipos de baterías recargables, se pueden resaltar las que comúnmente se utilizan en el mundo de la robótica, ellas son las baterías de ion litio (Li-Ion) y las baterías de polímero de litio (LiPo).

#### 2.3.3.1 BATERÍAS DE ION LITIO

La batería de ion litio utiliza como electrolito una sal de Litio contenido en un solvente orgánico (líquido) que proporciona los iones necesarios que circularán desde el cátodo (negativo) hasta el ánodo (positivo) durante la descarga, permitiendo que los electrones puedan moverse por el circuito y proporcionar la energía. Las propiedades de las baterías de Li-ion, como la ligereza de sus componentes, su elevada capacidad energética y resistencia a la descarga, junto con el poco efecto memoria que sufren, o su capacidad para funcionar con un elevado número de ciclos de regeneración, han permitido diseñar baterías ligeras, de pequeño tamaño y variadas formas. Éste tipo de baterías tiene técnicas de fabricación muy pulidas y avanzadas dada su madurez en el mercado, habiendo pasado muchos años desde sus primeras implementaciones (Sony en 1991) y por ende al día de hoy siguen siendo muy utilizadas en la electrónica móvil de consumo masivo como los smartphones, aunque la tendencia marca una migración progresiva hacia las baterías de LiPo en la medida que sus técnicas de fabricación se van perfeccionando.



Figura 2-25: Batería de Ion Litio

#### 2.3.3.2 BATERÍAS DE POLÍMERO DE LITIO

En el caso de las baterías LiPo, su funcionamiento es igual al de las baterías de Litio Ión salvo que la sal de Litio está contenida en una especie de gel (un compuesto polimérico) que hace menos probables los derrames, y a su vez hace que éste tipo de baterías sea más liviano.

Éste compuesto polimérico evita que los electrodos se toquen directamente, el material posee poros microscópicos que permiten que los iones (y no las partículas de los electrodos) migren de un electrodo a otro.

Las baterías están compuestas generalmente de varias celdas idénticas en paralelo para aumentar la capacidad de la corriente de descarga al igual que sucede con las baterías de Litio ion. [13]



**Figura 2-26: Batería de Polímero de Litio**

### 2.3.3.3 COMPARATIVA Y ELECCIÓN

A continuación, se elabora un cuadro comparativo entre las baterías Li-Ion y LiPo, denotando las características de cada una frente a los distintos parámetros en comparación. [14]

Propiedades	Li-Ion	LiPo
Densidad de carga	Alta	Baja
Ciclo de vida	Pierde la capacidad de carga en el transcurso del tiempo	No sufre mucho de la pérdida de capacidad de carga
Probabilidad de explosión	Riesgo de explosión cuando se sobrecarga o avería	Riesgo de ignición cuando se sobrecarga o avería pero no de explosión
Relación Precio/Capacidad de Carga	Muy Buena	Mala en comparación con las de Litio Ion
Velocidad de recarga	Lenta	Rápida
Peso	Pesada	Liviana
Tasa de conversión (Capacidad de convertir la energía almacenada en energía eléctrica)	85% - 95%	75% - 90%

**Tabla 2-3: Comparación de baterías Li-Ion vs LiPo**

La batería para este proyecto será LiPo, principalmente debido a que en la versión anterior del robot Hermes se ha usado una batería de Litio Ion y dos años después de estar almacenada, ha perdido toda su carga y las celdas se han deteriorado al punto de quedar inutilizable.

Con el diseño del sistema robótico final terminado, es decir con todos los componentes bien definidos, se realizará un análisis del consumo eléctrico total y se contrastará con las especificaciones de la batería LiPo a utilizar.

## 2.4 SISTEMA DE CONTROL

Un controlador PID es un mecanismo de control por realimentación que calcula la desviación o error entre un valor medido y un valor deseado. El algoritmo del control PID consiste de tres parámetros distintos: el proporcional, el integral, y el derivativo. El valor proporcional depende del error actual. El valor integral depende de los errores pasados y el valor derivativo es una predicción de los errores futuros. La suma de estos tres componentes es utilizada para ajustar la planta/proceso (Ver Figura 2-27). [15]

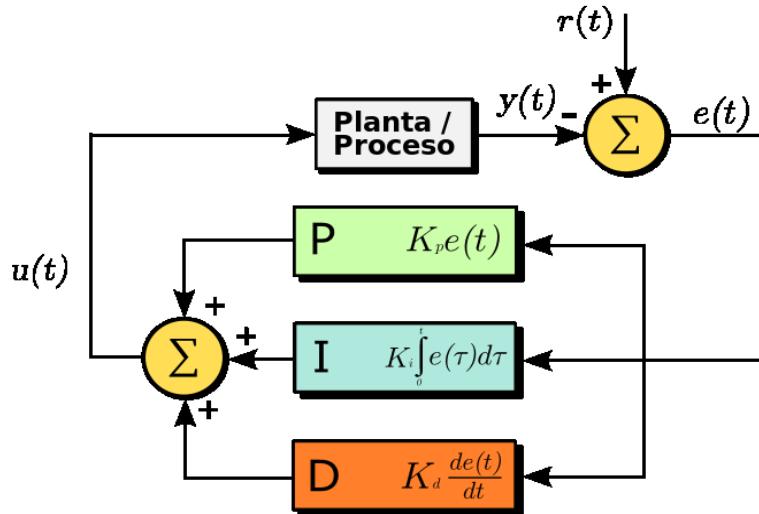


Figura 2-27: Sistema de control PID

A la hora de implementar el sistema de control para el robot omnidireccional, existen diferentes maneras de hacerlo en lo que respecta a las variables que serán controladas. La solución elegida, por ser considerada la de más rápida implementación, es un PID para cada una de las ruedas, donde la variable a controlar serán las velocidades lineales de cada rueda, obtenidas como se ha explicado en la sección Sensores optoacopladores.

### 2.4.1 ANÁLISIS FÍSICO

Para derivar la relación entre los pares de los motores y el movimiento del robot, primero necesitamos analizar la geometría del problema. Para que el robot sea omnidireccional, el número de ruedas debe ser mayor o igual a 3.

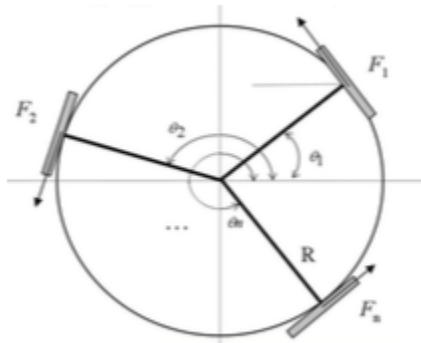


Figura 2-28: Fuerzas de un sistema onmidireccional

Como se muestra en la Figura 2-28, todos los ángulos de los ejes de los motores, son mediciones relativas en base al eje x del sistema del robot. Cada motor aporta una fuerza F, la cual

deriva en un movimiento de translación y un movimiento de rotación. Donde cada fuerza  $F$  es el radio  $R$  del centro del robot a la rueda, multiplicado por el torque del motor.

$$F = R * T \rightarrow Newton = Metro * \frac{Newton}{Metro}$$

#### Ecuación 2.4: Cálculo de fuerza

La variable de interés para el sistema de control a implementar es el movimiento del robot a lo largo de los ejes  $x$  e  $y$ , es decir, sobre un plano bidimensional. Para simplificar la expresión final, se considera la aceleración y velocidad instantáneas del robot con respecto a su propio marco de referencia. Por lo tanto, empezaremos a manejar términos de Magnitudes Euclidianas, en donde se habla de una velocidad de translación y una velocidad de rotación de todo el robot, y no de la velocidad y aceleración de cada uno de sus motores. Por ejemplo, un robot que se está moviendo hacia adelante, cuenta con una velocidad de translación positiva en dirección del eje  $Y$ , y una velocidad de translación igual a 0 en el eje  $X$ .

La aceleración de translación  $\alpha$  y la aceleración de rotación  $\omega$  del centro de masa de nuestro robot, son obtenidas de:

$$\alpha = \frac{1}{M} (f_1 + f_2 + \dots + f_n)$$

#### Ecuación 2.5: Aceleración de translación

$$\omega = \frac{R}{I} (f_1 + f_2 + \dots + f_n)$$

#### Ecuación 2.6: Aceleración de rotación

Donde  $M$  es la masa,  $R$  es el radio del robot,  $f_i$  denota la magnitud de la fuerza  $F_i$ , e  $I$  es el momento de inercia. Los cálculos son posibles utilizando estas expresiones, ya que las fuerzas son tangenciales al marco circular del robot y apuntan en la misma dirección de rotación, de modo que podemos trabajar sólo con los vectores de fuerza. Las magnitudes  $f_1, f_2, \dots, f_n$  pueden ser positivas o negativas, según el sentido de rotación del motor (en sentido contrario a las agujas del reloj o en el sentido de las agujas del reloj).

Se pueden obtener las componentes  $X$  e  $Y$  de la aceleración del robot, considerando la componente de cada fuerza:

$$M\alpha_x = -f_1 \sin\theta_1 - f_2 \sin\theta_2 - \dots - f_n \sin\theta_n$$

$$M\alpha_y = -f_1 \cos\theta_1 - f_2 \cos\theta_2 - \dots - f_n \cos\theta_n$$

#### Ecuación 2.7: Componentes de fuerzas

El momento de inercia para un cilindro homogéneo es  $I = \frac{1}{2}MR^2$ , para un anillo  $I = MR^2$ . Para cualquier distribución de masa en el medio y con la concentración en la periferia,  $I = \alpha MR^2$  con  $0 \leq \alpha \leq 1$ . Por lo tanto, se pueden expresar las ecuaciones de aceleración como una multiplicación de una matriz por un vector:

$$\begin{matrix} a_x \\ a_y \\ \omega \end{matrix} = \frac{1}{M} * \begin{pmatrix} -\sin\theta_1 & -\sin\theta_2 & \dots & -\sin\theta_n \\ \cos\theta_1 & \cos\theta_2 & \dots & \cos\theta_n \\ \frac{MR}{I} & \frac{MR}{I} & \dots & \frac{MR}{I} \end{pmatrix} * \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix}$$

#### Ecuación 2.8: Matriz de aceleraciones

Podemos usar  $I = \alpha M R^2$  y simplificar la matriz usando las mismas unidades (metros sobre segundos al cuadrado) para la aceleración planar y angular. En lugar de utilizar  $\omega$  podemos usar  $R\omega$ :

$$\frac{a_x}{R\omega} = \frac{1}{M} * \begin{pmatrix} -\sin\theta_1 & -\sin\theta_2 & \dots & -\sin\theta_n \\ \cos\theta_1 & \cos\theta_2 & \dots & \cos\theta_n \\ \frac{1}{\alpha} & \frac{1}{\alpha} & \dots & \frac{1}{\alpha} \end{pmatrix} * \begin{pmatrix} f_1 \\ f_2 \\ \vdots \\ f_n \end{pmatrix}$$

Ecuación 2.9: Simplificación de matriz de aceleraciones

Obteniendo una matriz  $3 \times n$ , donde  $n$  es la cantidad de ruedas que tengamos.

Ahora podemos calcular las velocidades finales de las ruedas, y la velocidad del robot, así como su velocidad angular, integrando la Ecuación 2.9: Simplificación de matriz de aceleraciones, respecto al tiempo. Sin embargo, debemos pensar al robot en un espacio Euclíadiano, calcular allí su trayectoria, y a partir de la misma deducir las velocidades de cada rueda. Usando la convención de que la dirección positiva de rotación es la dirección del giro de la mano derecha cuando sostenemos el eje de un motor sobre la mano, obtenemos la siguiente ecuación que relaciona las velocidades Euclidianas con las velocidades de cada motor:

$$\begin{pmatrix} v_1 \\ \vdots \\ v_n \end{pmatrix} = \begin{pmatrix} -\sin\theta_1 & \cos\theta_1 & 1 \\ -\sin\theta_2 & \cos\theta_2 & 1 \\ \vdots & \vdots & \vdots \\ -\sin\theta_n & \cos\theta_n & 1 \end{pmatrix} * \begin{pmatrix} v_x \\ v_y \\ R\omega \end{pmatrix}$$

Ecuación 2.10: Calculo de velocidades individuales

De la cual, se obtendrán las velocidades de cada motor, a partir de la velocidad Euclíadiana del robot. [16]

## 2.5 SISTEMA OPERATIVO

Dado el nivel de madurez del mercado en cuanto a robótica, se necesita un sistema operativo que provea herramientas para realizar un diseño y desarrollo de alta calidad. Imaginemos si cada persona que va a desarrollar un robot, debería desarrollar un driver para cada componente, o utilizar técnicas de simulación no estandarizadas. Para ello existen los sistemas operativos de robótica, que proveen un paquete de herramientas, como visualizadores de datos, drivers, simuladores, algoritmos y mecanismos de comunicación, de manera estandarizada y con enfoque en la modularidad para facilitar la reutilización de algoritmos.

Destacaremos a 3 de los más importantes:

- The Open Robot Control Software Project (Orcos): Es un conjunto portable de librerías C++ para control de robots. Provee componentes de software real-time, máquinas de estado, procesos distribuidos y generación de código.
- Robot Operating System (ROS): Es un framework para escribir software para robot, e incluye varias herramientas y librerías para simplificar este proceso. Fue diseñado para un desarrollo colaborativo, con componentes modulares y una comunidad mundialmente activa.
- Robot Construction Kit (Rock): Es un framework de robótica, basado en Orcos RTT (Real-Time Toolkit). Fue diseñado para ser extensible y contiene una gran cantidad de drivers para aplicaciones existentes.

Para la elección del sistema operativo a utilizar, se hizo foco en los aspectos denotados en la siguiente tabla:

	Orocos	ROS	Rock
Orientado a componentes	Si	Si	Si
Lenguajes de programación	C++	C++, Python, Java	C++, Ruby
Open Source	Si	Si	Si
Comunidad Activa	No	Si	No
Drivers para sensores utilizados	No	Si	Si
Comunicación entre módulos	Comunicación directa para mantener flujo controlado de mensajes	Tópicos	Comunicación directa para mantener flujo controlado de mensajes
Real Time	Si	No	Si

Tabla 2-4: Comparación entre sistemas operativos

Considerando que durante el desarrollo surgirán una gran cantidad de interrogantes que pueden interrumpir el avance del presente proyecto y dado el acotado tiempo de desarrollo que se tiene para lograr los objetivos dentro de un marco de Proyecto Integrador de Carrera, se ha optado por elegir aquella herramienta que cuente con la mayor información posible, ya sea de libros, foros activos en internet, papers, etc.

Después de realizar un análisis considerando lo anterior, se ha decidido utilizar ROS. éste cuenta con un gran soporte por parte de una comunidad en internet muy activa a nivel mundial. Además, también se tuvo en cuenta que maneja lenguajes de programación variados, por lo que se nos facilitará, en este aspecto, el desarrollo.

### 2.5.1 ROS (ROBOT OPERATING SYSTEM)

Robot Operating System (ROS), es una plataforma de desarrollo para aplicaciones robóticas, la cual provee varias características, tales como:

- Capacidades de alta calidad: ROS viene con capacidades listas para usar, como por ejemplo SLAM (Simultaneous Localization and Mapping) y AMCL (Adaptive Monte Carlo Localization) que pueden ser utilizados para la navegación autónoma en robot móviles. Estas capacidades están en constante evolución, por lo que, escribir nuevo código desde cero para dichas capacidades, sería “reinventar la rueda”. Además de esto, las capacidades que brinda son altamente configurables según las necesidades.
- Herramientas: ROS provee herramientas para debuggear, visualizar, y realizar simulaciones. Tales herramientas como rqt\_gui, RViz, y Gazebo son algunas de las herramientas open platform mas potentes para realizar este tipo de actividades.

- Soporte para actuadores y sensores de gama alta: ROS provee drivers e interfaz para varios sensores y actuadores de robótica. Algunos de ellos como Velodyne-LIDAR, Laser Scanners, Kinect, y actuadores como Dynamixel Servos.
- Interoperabilidad: Mediante diferentes tipos de mensajes de comunicación entre nodos independientes entre sí. Estos nodos pueden ser programados en cualquier lenguaje que tenga librerías de ROS. Podemos escribir nodos de alta performance en C o C++ como así también podemos usar otros nodos en Python y Java. Este tipo de flexibilidad no está disponible en otros frameworks.
- Manejo de recursos concurrentes: Manejar un mismo recurso de hardware entre dos procesos diferentes es una cuestión complicada. Pero con ROS se puede acceder a los dispositivos a través del intercambio de tópicos, lo cual es permitido por los drivers provistos por ROS.
- Comunidad activa: Si elegimos una librería o framework, sobre todo de una comunidad open source, una de las principales consideraciones a tener en cuenta es el soporte del software y la comunidad de desarrolladores.

La filosofía de ROS es hacer pequeñas piezas de software (un nodo o más) que puedan funcionar perfectamente en cualquier otro robot. Lo que se obtiene de esta idea es la habilidad de crear funcionalidades que pueden ser compartidas y usadas por otros robots sin ningún tipo de esfuerzo, por lo cual, como dijimos anteriormente, no hay que reinventar la rueda.

ROS fue desarrollado originalmente en 2007 por el laboratorio de inteligencia artificial de Stanford (SAIL) en soporte del proyecto “Stanford AI Robot”. En 2008, el desarrollo continuó principalmente en Willow Garage, un instituto de investigación robótica, con el cual más de 20 instituciones colaboraron dentro del modelo de desarrollo.

Muchas instituciones de investigación utilizan ROS, agregando nuevo hardware y compartiendo su código. Incluso, compañías han empezado a adaptar sus productos para ser utilizados con ROS.

Cada día, más dispositivos son soportados por este framework. Además, gracias a ROS y su filosofía open hardware, las compañías pueden crear mejores sensores y mas baratos.

ROS está basado en una arquitectura de grafos, con una topología centralizada, donde los procesos toman lugar en nodos que pueden recibir o postear valores de un sensor, control, estado, planeamiento, actuador o lo que fuere.

El \*-ros-pkg es una comunidad de repositorio para desarrollar fácilmente librerías de alto nivel. Muchas funcionalidades frecuentemente asociadas a ROS, como la librería de navegación y el visualizador Rviz, son desarrolladas en este repositorio. Estas librerías proporcionan un set de poderosas herramientas para trabajar bajo el entorno de ROS. Los visualizadores, simuladores, y herramientas de debug son las *features* más importantes.

ROS es lanzado bajo los términos de licencia BSD (Berkeley Software Distribution) y es un software open source. Ésto implica que es gratuito para su uso comercial y de investigación. [17]

## 2.6 SISTEMA DE COMUNICACIÓN

Un sistema de comunicación [18] es el conjunto de elementos conformado por transmisores, receptores, y el canal de transmisión. El mensaje original, que es constituido por el transmisor, debe ser transformado a señales eléctricas, y luego se somete a la modulación, en donde la información se imprime sobre una señal portadora para adaptarse a las características del canal transmisor. Por último, el receptor es el encargado de decodificar estas señales eléctricas para transformarla en un mensaje legible.

Los diferentes canales de transmisión se pueden clasificar en:

- Guiados: Este tipo de medio son los que proporcionan un camino físico a lo largo de toda la transmisión. Ejemplos de este medio pueden ser fibra óptica, cables coaxiales, pares trenzados, etc.
- No guiados: Este tipo de medio de transmisión es inalámbrico, es decir, no confinan la información a un espacio definido. Utilizan el aire, mar, tierra como medio de transmisión.

El ancho de banda del canal, determina el espectro de frecuencias, categorías y volumen de la información que el canal puede acomodar en un tiempo determinado. Al aumentar la frecuencia, aumenta la capacidad para contener la información, o sea que, para transmitir mucha información en poco tiempo, se requieren señales de banda ancha. La unidad de medida de la frecuencia de las ondas electromagnéticas es el hertzio, y las magnitudes que se utilizan frecuentemente son:

- Kiloherzios (KHz): miles de ciclos por segundo: 1000 Hz.
- Megahertzios (MHz): millones de ciclos por segundo: 1000000Hz.
- Gigahertzios (GHz): miles de millones de ciclos por segundo: 1000000000Hz.

### 2.6.1 TRANSMISSION CONTROL PROTOCOL

Transmission Control Protocol, o más bien conocido como TCP, es un protocolo de la capa de transporte del modelo TCP/IP. Este es un protocolo orientado a conexión, osea que permite que dos máquinas que están comunicadas controlen el estado de la transmisión pudiendo evitar la saturación de la red. Además, permite que los datos se formen en segmentos de longitud variada para entregarlos al protocolo IP. Y cuenta con un protocolo para indicar el inicio y finalización de la transmisión.

Durante una comunicación, las dos máquinas deben establecer una conexión. La máquina emisora (la que solicita la conexión) se llama cliente, y la máquina receptora se llama servidor. Por eso es que se llama arquitectura Cliente-Servidor.

Las máquinas de dicho entorno se comunican en modo en línea, es decir, que la comunicación se realiza en ambas direcciones. Para posibilitar la comunicación y que funcionen bien todos los controles que la acompañan, los datos se agrupan; es decir, que se agrega un encabezado a los paquetes de datos que permitirán sincronizar las transmisiones y garantizar su recepción.

Cuando se emite un segmento, se lo vincula a un número de secuencia. Con la recepción de un segmento de datos, la máquina receptora devolverá un segmento de datos donde un bit indicador de ACK esté fijado en 1 (para poder indicar que es un acuse de recibo) acompañado por un número de acuse de recibo que equivale al número de secuencia anterior (Ver Figura 2-29).

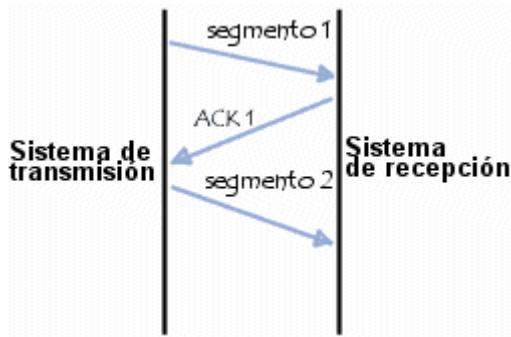


Figura 2-29: Intercambio de paquetes TCP

Además, usando un temporizador que comienza con la recepción del segmento en el nivel de la máquina originaria, el segmento se reenvía cuando ha transcurrido el tiempo permitido, ya que en este caso la máquina originaria considera que el segmento está perdido (Ver Figura 2-30).

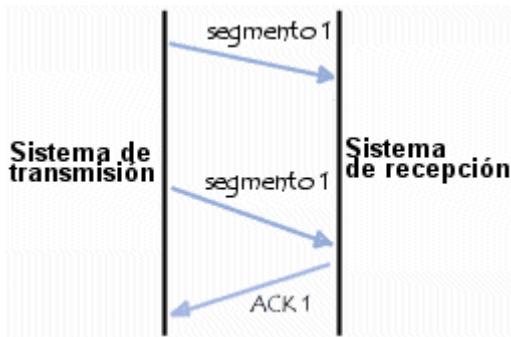


Figura 2-30: Reenvío de paquetes perdidos

Sin embargo, si el segmento no está perdido y llega a destino, la máquina receptora lo sabrá, gracias al número de secuencia, que es un duplicado, y sólo retendrá el último segmento que llegó a destino. [19]

## 2.6.2 BLUETOOTH

Bluetooth [20] es un protocolo estandarizado para enviar y recibir datos sobre una conexión inalámbrica. Algunas de las características de este protocolo son que tiene un bajo consumo de energía, como así también son funcionales para un corto alcance.

El protocolo bluetooth opera a 2.4GHz, esta es la misma banda donde operan los protocolos WiFi, ZigBee, o RF. Contiene una colección de reglas y especificaciones estandarizadas para diferenciarse del resto de protocolos.

La forma de operar de este protocolo es utilizando un modelo maestro-esclavo para controlar quien y cuando de los dispositivos pueden enviar datos. Con este modelo, un único maestro puede ser conectado para una cantidad de hasta 7 esclavos como se muestra en la siguiente Figura 2-31.

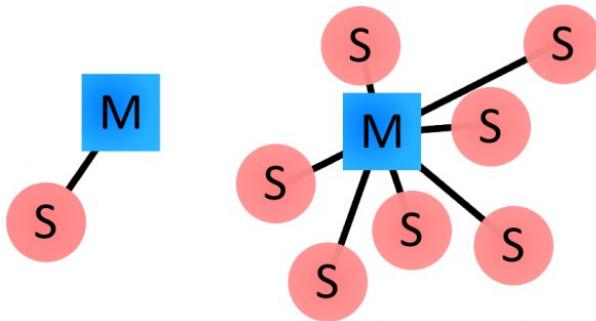


Figura 2-31: Modelo Maestro-Esclavo

El dispositivo maestro coordina la comunicación a lo largo de la red. Este puede enviar datos a cualquiera de sus esclavos, como así también pedir datos de cualquiera de ellos. Los dispositivos esclavos solamente permiten transmitir y recibir datos del dispositivo maestro, es decir, no es posible intercambiar datos entre dispositivos esclavos.

## 2.6.3 WI-FI

La especificación IEEE 802.11 (ISO/IEC 8802-11) [21] es un estándar internacional que define las características de una red de área local inalámbrica (WLAN). Con Wi-Fi se pueden crear redes de área local inalámbricas de alta velocidad siempre y cuando el equipo que se vaya a conectar no esté muy alejado del punto de acceso. En la práctica, Wi-Fi admite ordenadores portátiles, equipos de escritorio, asistentes digitales personales (PDA) o cualquier otro tipo de dispositivo que requiera de conectividad a Internet.

El estándar 802.11 establece los niveles inferiores del modelo OSI para las conexiones inalámbricas que utilizan ondas electromagnéticas, por ejemplo:

- La capa física ofrece tres tipos de codificación de información.
- La capa de enlace de datos compuesta por dos subcapas: control de enlace lógico (LLC) y control de acceso al medio (MAC).

La capa física define la modulación de las ondas de radio y las características de señalización para la transmisión de datos mientras que la capa de enlace de datos define la interfaz entre el bus del equipo y la capa física, en particular un método de acceso parecido al utilizado en el estándar Ethernet, y las reglas para la comunicación entre las estaciones de la red. El estándar 802.11 tiene tres capas físicas que establecen modos de transmisión alternativos:

Capa física (PHY)	SSS	HSS	Infrarrojo
----------------------	-----	-----	------------

Tabla 2-5: Capas físicas del estándar 802.11

Cualquier protocolo de nivel superior puede utilizarse en una red inalámbrica Wi-Fi de la misma manera que puede utilizarse en una red Ethernet. El estándar 802.11 en realidad es el primer estándar y permite un ancho de banda de 1 a 2 Mbps. El estándar original se ha modificado para optimizar el ancho de banda (incluidos los estándares 802.11a, 802.11b y 802.11g, denominados estándares físicos 802.11) o para especificar componentes de mejor manera con el fin de garantizar mayor seguridad o compatibilidad.

Los estándares 802.11a, 802.11b y 802.11g, llamados "estándares físicos", son modificaciones del estándar 802.11 y operan de modos diferentes, lo que les permite alcanzar distintas velocidades en la transferencia de datos según sus rangos.

Estándar	Frecuencia	Velocidad	Rango
WiFi a (802.11a)	5 GHz	54 Mbit/s	10 m
WiFi B (802.11b)	2,4 GHz	11 Mbit/s	100 m
WiFi G (802.11g)	2,4 GHz	54 Mbit/s	100 m

Tabla 2-6: Comparación entre diferentes estándares

# CAPÍTULO 3

## 3 ITERACIÓN 0

Conforme a los objetivos y requerimientos funcionales y no funcionales para el siguiente proyecto, y realizada una previa elección de la metodología de trabajo, resta elaborar un plan de trabajo, el cual indique los pasos a seguir para el cumplimiento de cada objetivo y/o requerimiento, en base a la metodología elegida.

Entonces, para una metodología de trabajo iterativa, se definirán a grandes rasgos las acciones a realizar en cada iteración junto con el Director del Proyecto Integrador, de tal manera que cada una pueda realizarse en un plazo aproximado de 30 días, dando un plazo de 6 meses para cumplimentar todos los requerimientos.

El orden de las acciones a realizar en las iteraciones ha sido definido en función de su impacto en la disminución de riesgos. El objetivo es disminuir lo más rápido posible los mismos.

Se debe tener en cuenta que al final de una iteración se realizarán las pruebas pertinentes y pueden detectarse fallas u oportunidades de mejoras, las cuales se atenderán en una nueva iteración únicamente si éstos no permiten avanzar con la siguiente iteración planificada.

	Iteración	Acciones
1	Correr framework o sistema de desarrollo elegido sobre PC y/o sistema embebido a utilizar	<ul style="list-style-type: none"><li>• Descarga e instalación del S.O. en la PC</li><li>• Descarga e instalación del S.O. en el sistema embebido</li><li>• Realización de ejemplos proporcionados por el creador del framework, para corroborar el correcto funcionamiento</li></ul>
2	Modelado 3D y simulación del robot	<ul style="list-style-type: none"><li>• Realización de modelo 3D en software de diseño gráfico</li><li>• Realización del modelado en un lenguaje descriptivo</li><li>• Prueba y testeo del modelo corriendo sobre un software de simulación</li></ul>
3	Puesta en funcionamiento de los sensores y actuadores a utilizar	<ul style="list-style-type: none"><li>• Conexión de sensores</li><li>• Conexión de actuadores</li><li>• Prueba y medición de datos para correcto funcionamiento</li></ul>
4	Sistema autolocalización y mapeo (SLAM)	<ul style="list-style-type: none"><li>• Sistema de localización</li><li>• Sistema de mapeo</li></ul>
5	Sistema de control	<ul style="list-style-type: none"><li>• Estudio del modelo físico para el caso omnidireccional</li><li>• Determinación de coeficientes PID</li><li>• Implementación del sistema de control</li></ul>
6	Sistema comunicación de mando	<ul style="list-style-type: none"><li>• Control por teclado</li><li>• Control por joystick</li><li>• Control por smartphone</li></ul>

Tabla 3-1: Plan de acción frente a cada iteración

### 3.1 MATRIZ DE TRAZABILIDAD DE ITERACIONES - REQUERIMIENTOS/PRUEBAS

En el siguiente gráfico se da una vista general de cómo se irán cumpliendo los requerimientos funcionales en la medida que se avanza en el proyecto acorde a las iteraciones planificadas en la Tabla 3-1: Plan de acción frente a cada iteración. Véase también la sección de Requerimientos funcionales y Pruebas de Requerimientos Funcionales.

	RF1	RF2	RF3	RF4	RF5	RF6	RF7	RF8	RF9	Tiempo (Meses)
Iteración 1										
Iteración 2						PRF6	PRF7-1	PRF8-1		
Iteración 3		PRF2-1		PRF4-1						
Iteración 4		PRF2-2 PRF2-3	PRF3							
Iteración 5	PRF1			PRF4-2						
Iteración 6					PRF5		PRF7-2	PRF8-2	PRF9	
Sistema Final										

Tabla 3-2: Matriz de trazabilidad de iteraciones - requerimientos/pruebas

#### Códigos Alfabéticos

- RF → Requerimiento Funcional
- PRF → Prueba de Requerimiento Funcional

#### Códigos de Color

-  → Requerimiento no cumplido
-  → Requerimiento parcialmente cumplido
-  → Requerimiento totalmente cumplido

Puede observarse que los requerimientos funcionales 2, 4, 7 y 8 se resuelven completamente luego de dos o más iteraciones. Ésto se debe principalmente a la interdependencia de los requerimientos funcionales, los cuales pueden cumplirse parcialmente luego de una iteración.

Aquellas pruebas que poseen una flecha verde hacia abajo, son las que pudiendo realizarse al finalizar la iteración donde se encuentran, se decide hacerlas a modo de pruebas de integración cuando el robot se encuentre completamente terminado e implementado.

# CAPÍTULO 4

## 4 ITERACIÓN 1: PUESTA A PUNTO DE ROS

### 4.1 INTRODUCCIÓN

En la presente iteración se hará foco en el framework ROS, principalmente su puesta en funcionamiento tanto en las PCs que formarán parte del sistema final de monitoreo, como así también en los sistemas embebidos directamente involucrados con el funcionamiento del robot.

### 4.2 REQUERIMIENTOS

En ésta primera etapa de desarrollo, se sentarán las bases para el cumplimiento de los requerimientos no funcionales, principalmente aquellos estrictamente necesarios:

- Debe poderse interactuar con el robot desde múltiples plataformas (PC, Joystick, Smartphones)
- Debe contar con documentación de código y de proyecto
- Debe contar con pruebas y tests de funcionalidades
- Debe utilizarse un Sistema Operativo específico para robótica
- Debe tener suficientes entradas para sensores
- Debe tener suficientes salidas para actuadores
- Debe permitir el desarrollo modular

### 4.3 DESARROLLO

#### 4.3.1 SELECCIÓN DE LA PLACA PRINCIPAL DE DESARROLLO

Existe poca información acerca de los requisitos computacionales necesarios para un óptimo desempeño de lo que se desea implementar sobre el Hermes III en este proyecto integrador (Sensor 3D, Giroscopio, Encoders, Comunicación serial).

Según la documentación del framework ROS [22], los siguientes sistemas embebidos han sido probados satisfactoriamente y cuentan con soporte oficial:

- ❖ Snapdragon Flight
- ❖ Inforce IFC6410
- ❖ eInfochips Eragon600
- ❖ Nvidia Jetson TK1
- ❖ Odroid C1, X, U2, U3, XU3 and XU4
- ❖ SolidRun CuBox-i Pro
- ❖ BeagleBone Black
- ❖ Parallelia
- ❖ RadxaRock
- ❖ Pandaboard ES
- ❖ Gumstix Overo Fire
- ❖ Wandboard Quad
- ❖ Raspberry Pi 2

- ❖ Google Nexus 5
- ❖ DJI Manifold
- ❖ elInfochips Eragon 410
- ❖ Dragonboard 410c
- ❖ Orange Pi 2
- ❖ Orbbec Persee

En primera instancia se optó por la instalación de ROS sobre la placa de desarrollo Odroid U3, ya que el laboratorio de Arquitectura de Computadoras de la Facultad contaba previamente con una, pero al momento de realizar pruebas con el sensor de profundidad Kinect, ésta funcionaba con una velocidad de captura de 11 fotogramas por segundo, por lo cual se optó por descartar esta opción debido a su baja performance en cuanto a procesamiento de imágenes.

El criterio de desestimación de la placa Odroid U3 se debe a que en un algoritmo de detección de obstáculos y posicionamiento por imágenes es fundamental la velocidad de captura/procesamiento de las mismas. Sencillamente, no será lo mismo capturar una imagen cada 90 ms (10 fps) que cada 33 ms (30 fps). Considérese el siguiente ejemplo:

Teniendo un robot desplazándose a una velocidad de 1 m/s, tendremos:

- **Caso 1 (10fps)**  
El robot se moverá a ciegas unos 10 centímetros.
- **Caso 2 (30 fps)**  
El robot se moverá a ciegas unos 3,3 centímetros.

El valor de distancia de movimiento a ciegas define el margen de distancia mínimo a los objetos que deberá tener el robot para no impactarlos. En la Figura 4-1 se muestra la relación entre los FPS del dispositivo sensor y la distancia de recorrido a ciegas. De todas formas, si un obstáculo se interpone súbitamente dentro de éste rango de distancia durante su desplazamiento a ciegas, producirá un choque.

Movimiento a Ciegas vs FPS

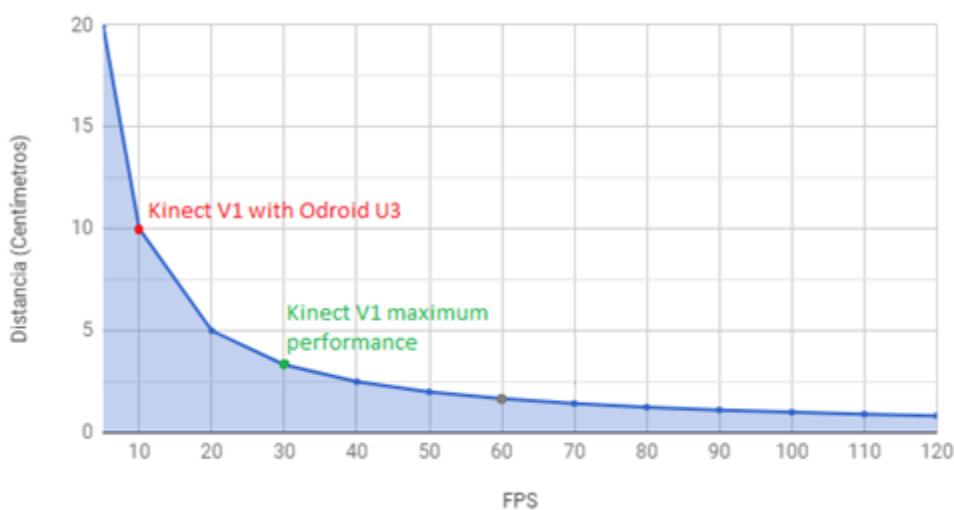


Figura 4-1: Distancia a ciegas vs FPS

Mínimamente se busca una placa de desarrollo que logre explotar al máximo la velocidad de captura del sensor de imágenes y profundidad que se utilice. Cabe destacar que, en su mayoría, estos

dispositivos ya poseen una limitación por hardware de velocidad máxima de captura de imágenes posible, por lo que no se pueden obtener rendimientos más allá de los especificados por el fabricante.

Por lo tanto, se procede a buscar una placa de desarrollo que posea mejoras específicas para las tareas de mayor costo computacional del presente trabajo, las cuales son “procesamiento de imágenes” y “operaciones de punto flotante” para lo que al sistema de control se refiere. Entre las analizadas, la más elegida entre la comunidad para la robótica de bajo costo y para tecnologías SLAM es la Nvidia Jetson TK1 la cual se procede a analizar a continuación.

#### 4.3.1.1 NVIDIA JETSON TEGRA K1

El SoC que utiliza la placa de desarrollo NVIDIA Jetson TK1 es el que se muestra en la siguiente Figura 4-2:



Figura 4-2: SoC Nvidia Tegra K1

Cuyas características son:

Tegra K1	
GPU	
NVIDIA® Kepler™ Architecture	192 NVIDIA CUDA® Cores
<b>CPU</b>	
CPU Cores and Architecture	NVIDIA 4-Plus-1™ Quad-Core ARM Cortex-A15 "r3"
Max Clock Speed	2.3 GHz
<b>Memory</b>	
Memory Type	DDR3L and LPDDR3

Max Memory Size	8 GB (with 40-bit address extension)
<b>Display</b>	
LCD	3840x2160
HDMI	4K (UltraHD, 4096x2160)
<b>Package</b>	
Package Size/Type	23x23 FCBGA 16x16 S-FCCSP 15x15 FC PoP
Process	28 nm

**Tabla 4-1: Características Nvidia Jetson Tegra K1**

Al analizar la arquitectura del SoC Tegra K1, se pueden destacar importantes características de diseño de los que podrá beneficiarse el proyecto.

### 1) Mayor Autonomía



**Figura 4-3: Procesador “NVIDIA 4-Plus-1™ Quad-Core ARM Cortex-A15”**

El procesador cuenta con un modo de ultra-ahorro de energía en el que trabajará exclusivamente un núcleo extra de bajo consumo cuando el sistema se encuentra bajo poca demanda computacional. Ésto podría significar para el robot Hermes III mayor autonomía de su batería (Ver Figura 4-3).

## 2) Alivio de Trabajo de CPU

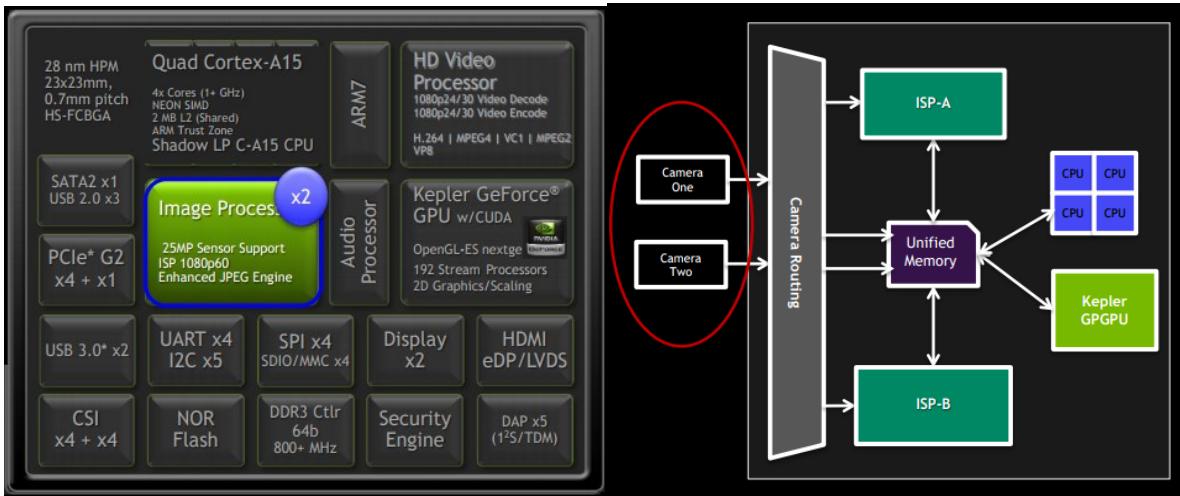


Figura 4-4: Dual ISP

Posee doble ISP (Image Signal Processor), las cuales en simples palabras son preprocesadores de imágenes que alivian la carga de CPU al filtrar la información procedente de cámaras de video, interpretando los datos de sus sensores y transformándolos en imágenes RGB que son disponibilizadas ya sea a la CPU o GPU indistintamente por medio de una “Memoria Unificada”. Adicionalmente estos módulos tienen la capacidad de manejar el auto-foco, el balance de blancos, el ajuste de exposición, etc (Ver Figura 4-4 y Figura 4-5).

Éstas características para el proyecto Hermes III son fundamentales, porque hacen a un procesamiento de imágenes potencialmente más veloz, brindando incluso la posibilidad de procesar las imágenes captadas por las cámaras totalmente en la GPU, aprovechando sus 192 núcleos CUDA.

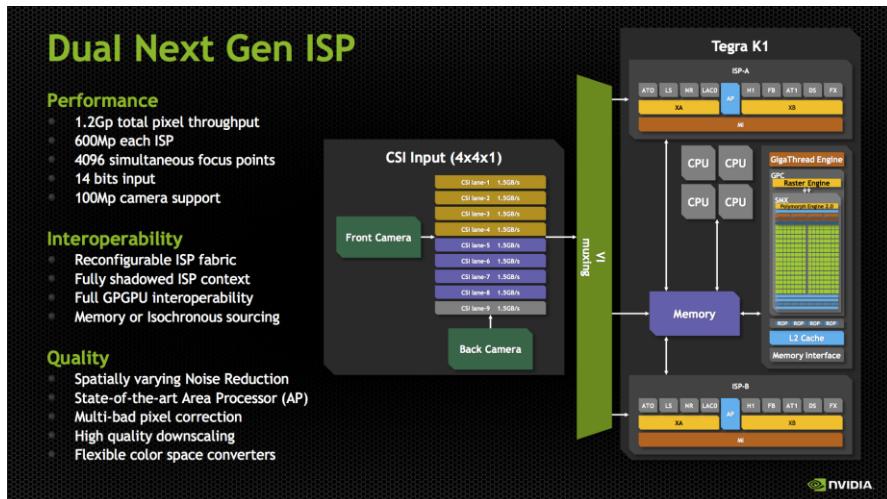


Figura 4-5: Dual Next Gen ISP

### 3) GPU de Alto Rendimiento

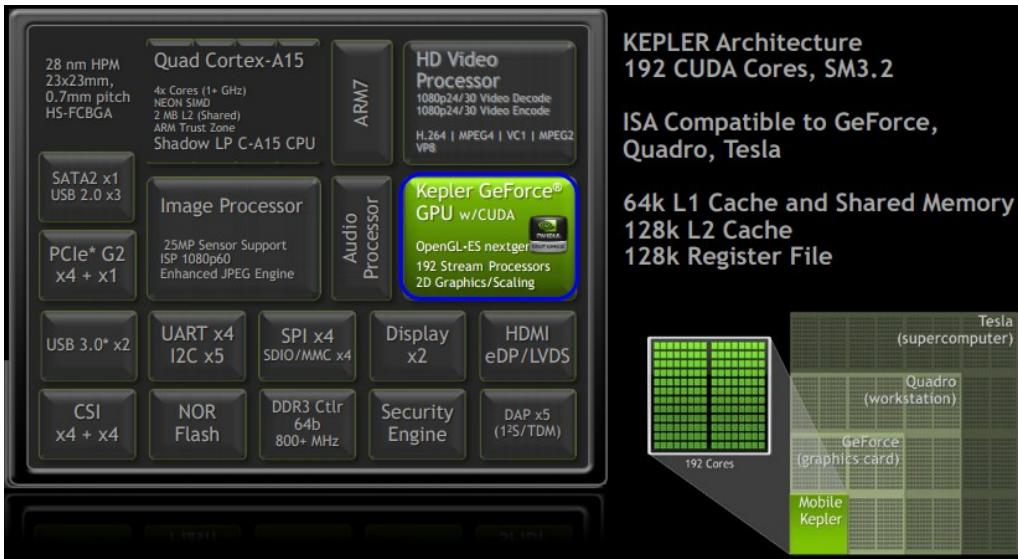


Figura 4-6: GPU Nvidia con CUDA Cores

Por último, y no por ello menos importante, se tiene la GPU NVIDIA de 192 núcleos CUDA con micro-arquitectura Kepler y una litografía de 28 nanómetros desarrolladas especialmente para el procesamiento de imágenes y renderizado 3D sin desatender el consumo de energía, ya que el SoC está destinado a dispositivos móviles (Ver Figura 4-6). [23]

En el proyecto integrador probablemente no se logre aprovechar al 100% sus capacidades, dado que no se hará un análisis de mejora en cuanto a la eficiencia de los algoritmos a utilizar que ya son provistos por el Framework ROS. En trabajos futuros se podrán modificar para obtener el máximo provecho del SoC Tegra K1 del sistema embebido utilizado, por ejemplo, aprovechando al máximo el procesamiento en paralelo de sus 192 núcleos CUDA.

#### 4.3.2 SELECCIÓN DE VERSIÓN DEL SISTEMA OPERATIVO

En la sección Sistema Operativo se han planteado las características del framework ROS que lo destacaban por sobre otras opciones en el mercado.

Dicho framework trabaja sobre un sistema operativo GNU/Linux, específicamente la distribución denominada “Ubuntu”, y dada su naturaleza Open Source cuenta con distintas versiones que se van lanzando anualmente y con distintos tiempos de duración de soporte.

Según recomendaciones del Director del presente Proyecto Integrador, no siempre es recomendable utilizar las últimas versiones lanzadas de software, ya que las mismas al agregar funcionalidades nuevas y tener poco tiempo de uso dentro de la comunidad Open Source, poseen muchos problemas latentes aún no descubiertos y por ende no resueltos. En el peor de los casos, alguno de éstos problemas sin resolver pueden ser bloqueantes para el avance del proyecto Hermes 3. Es por ello que se procede a realizar un análisis comparativo de las actuales versiones del Framework ROS que aún cuentan con soporte.

##### 4.3.2.1 CRITERIO GENERAL DE PONDERACIÓN

A la hora de analizar y ponderar las versiones de ROS, se ha priorizado la posibilidad de tener la menor cantidad de inconvenientes e incompatibilidades durante el desarrollo del Hermes 3, para lograr cumplir con todos los requerimientos en el menor tiempo posible y a su vez lograr construir un robot funcionalmente estable.

Referencia de ponderación

Factor de Relevancia	Puntuación
1: Bajo	1: Mal candidato
2: Medio	2: Candidato aceptable
3: Alto	3: Buen candidato

Tabla 4-2: Referencia de ponderación

Factor de Relevancia	Propiedades	ROS Indigo Igloo	ROS Jade Turtle	ROS Kinetic Kame	Criterio de Puntuación
2	Fin de Soporte	2019	2017	2021	Versión con más años de Soporte
		<b>Puntuación: 2</b>	<b>Puntuación: 1</b>	<b>Puntuación: 3</b>	
3	Versión de Sistema Operativo sobre el que trabaja	Ubuntu 14.04 LTS (Trusty)	Ubuntu 15.04 (Vivid Vervet) Ubuntu 14.04 (Trusty)	Ubuntu 16.04 (Xerial)	Sistema Operativo Oficial de Nvidia Tegra TK1 (Linux4Tegra - Based on Ubuntu 14.04)
		<b>Puntuación: 3</b>	<b>Puntuación: 3</b>	<b>Puntuación: 1</b>	
1	Versiones de software que utiliza	Gazebo 2.0 OpenCV 2.0 OctoMAP 1.7 Python 2.7 Cmake 2.8.11	Gazebo 5.0 OpenCV 2.0 OctoMAP 1.8 Python 2.7 Cmake 2.8.12	Gazebo 7.0 OpenCV 3.1 OctoMAP 1.8 Qt 5.3 PyQt 5 gcc 4.9 Python 2.7 Cmake 3.0.2	Capacidades relevantes para el proyecto
		<b>Puntuación: 3</b>	<b>Puntuación: 3</b>	<b>Puntuación: 3</b>	
3	Cantidad de documentación y tópicos en el foro oficial de ROS (basados en etiquetas al 31/08/16)	698 tópicos	69 tópicos	79 tópicos	Versión con mayor cantidad de documentación y tópicos en foros
		<b>Puntuación: 3</b>	<b>Puntuación: 1</b>	<b>Puntuación: 1</b>	
2	Funcionalidades (Packages al 23/05/16)	2149 Paquetes	1016 Paquetes	524 Paquetes	Mayor cantidad de funcionalidades
		<b>Puntuación: 3</b>	<b>Puntuación: 2</b>	<b>Puntuación: 1</b>	
	TOTAL $\Sigma$ relevancia* puntuación	2*2 3*3 1*3 3*3 2*3 ----- <b>31</b>	2*1 3*3 1*3 3*1 2*2 ----- <b>21</b>	2*3 3*1 1*3 3*1 2*1 ----- <b>17</b>	

Tabla 4-3: Comparación frente a diferentes versiones de ROS

De acuerdo a los valores finales de ponderación, se utilizará la versión de ROS “Indigo Igloo”, debido principalmente a su madurez en la comunidad, la cantidad de debates en torno al mismo y su soporte extendido, que lo hacen idóneo para el proyecto.

#### 4.3.3 INSTALACIÓN DE ROS “INDIGO IGLOO”

La placa de desarrollo NVIDIA Jetson Tegra K1 viene con un sistema operativo preinstalado basado en Ubuntu 14.04 (Linux4Tegra), por ende, se pasará a explicar directamente la instalación del Framework sobre dicha plataforma, la cual es totalmente compatible.

Los siguientes pasos intentan describir resumidamente lo realizado sobre la placa de desarrollo, siendo válido también para realizarlo sobre cualquier sistema que posea una instalación de Ubuntu 14.04.

##### 4.3.3.1 PASOS A SEGUIR SOBRE UBUNTU 14.04

- 1) Lo primero que se debe hacer es habilitar los repositorios universe, restricted y multiverse, lo cual permite al sistema Operativo instalar fuentes de terceros, como es el caso de ROS. Para ello se busca “Software & Updates” en Ubuntu, y se habilita las opciones antes mencionadas (Ver Figura 4-7).

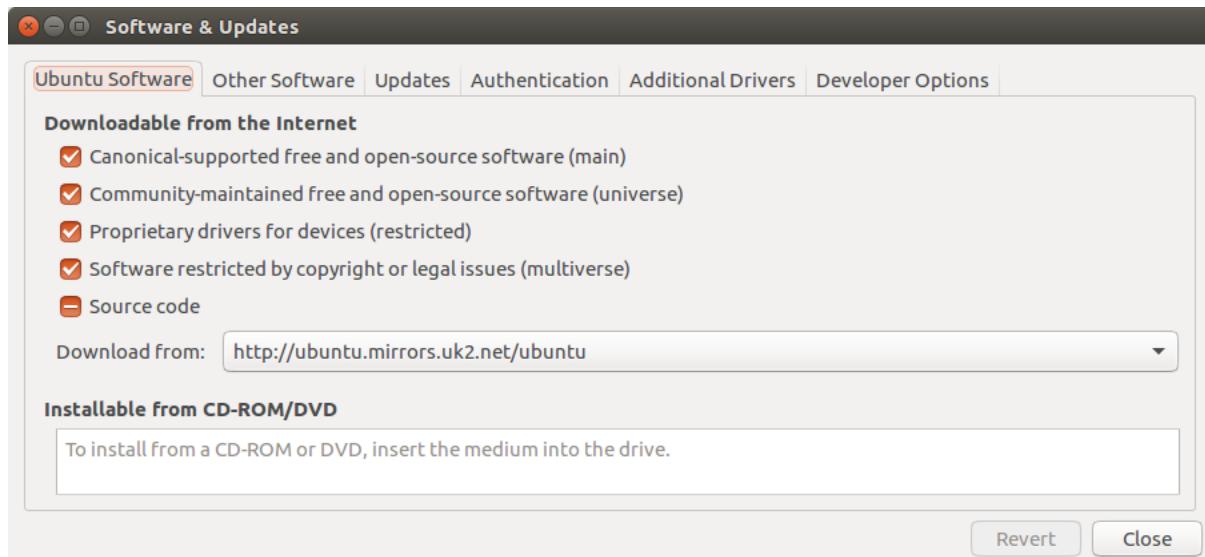


Figura 4-7: Repositorios de paquetes Ubuntu

- 2) Luego, se actualiza los paquetes de Ubuntu:

```
$ sudo apt-get update && sudo apt-get upgrade
```

3) Se setea la localidad:

```
$ sudo update-locale LANG=C LANGUAGE=C LC_ALL=C LC_MESSAGES=POSIX
```

4) Se edita el archivo "source.list" para aceptar software de ROS:

```
$ sudo sh -c 'echo "deb http://packages.ros.org/ros/ubuntu trusty main" > /etc/apt/sources.list.d/ros-latest.list'
```

5) Se importan las llaves

```
$ sudo apt-key adv --keyserver hkp://ha.pool.sks-keyservers.net:80 --recv-key 0xB01FA116
```

6) Se actualizan los paquetes nuevamente:

```
$ sudo apt-get update
```

7) Se debe instalar las librerías y herramientas de ROS básicas que no contienen GUI:

```
$ sudo apt-get install ros-indigo-ros-base
```

8) Antes que se pueda usar ROS, se necesitará instalar e inicializar "rosdep" que habilita la instalación sencilla de dependencias permitiendo que los códigos de ROS compilen, y además es necesario para otros componentes *core* de ROS:

```
$ sudo apt-get install python-rosdep  
$ sudo rosdep init  
$ rosdep update
```

9) Es conveniente añadir las variables de entorno de ROS en las sesiones de bash. Se realiza tanto para el "user" como para "root":

```
$ echo "source /opt/ros/indigo/setup.bash" >> ~/.bashrc  
$ source ~/.bashrc
```

10) Se instala rosinstall, el cual es usado frecuentemente para descargar fácilmente paquetes mediante la línea de comandos:

```
$ sudo apt-get install python-rosinstall
```

11) Para instalar el software de monitoreo RVIZ y modelos de simulación de robots predeterminados:

```
$ sudo apt-get install ros-indigo-rviz  
$ sudo apt-get install ros-indigo-robot-model
```

[17]

#### **Exclusivo del sistema embebido NVIDIA Jetson TK1 (arquitectura ARM)**

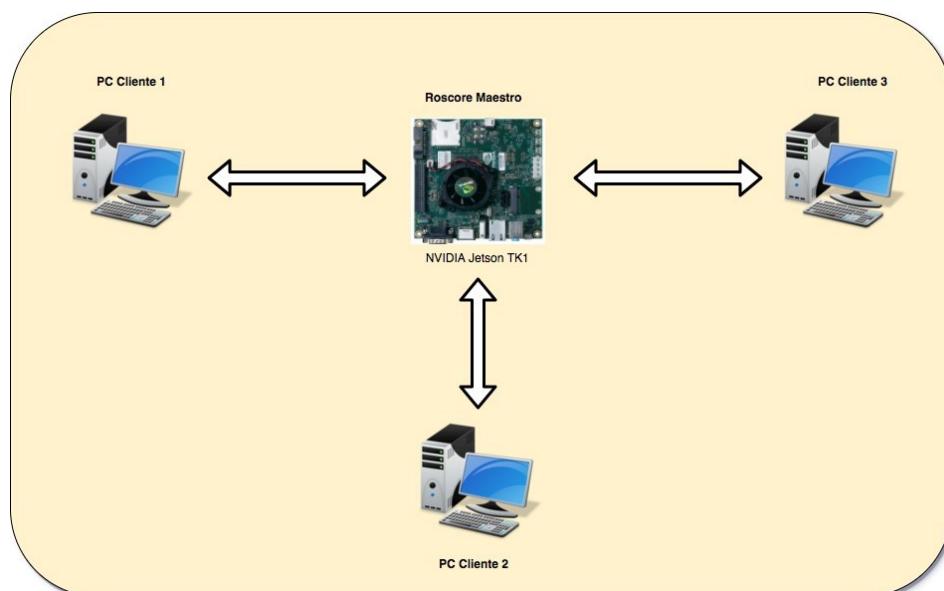
El software de monitorización RVIZ, con el que se verán los mapas 3D y el modelo simulado del robot en tiempo real, consume muchos recursos computacionales y no está pensado para ejecutarse en arquitecturas ARM. Sin embargo, la documentación oficial de ROS aclara que particularmente la placa de desarrollo Jetson TK1 es capaz de ejecutar el software RVIZ dada la potencia de su GPU. Para ello debe hacerse lo siguiente:

```
$ echo "unset GTK_IM_MODULE" >> ~/.bashrc  
$ source ~/.bashrc  
(Tanto en 'root' como en 'user')
```

#### **4.3.4 INTERCONEXIÓN ENTRE LOS DISTINTOS NODOS DE ROS**

ROS está diseñado para una comunicación distribuida (Ver Figura 4-8). Un nodo bien programado, no depende del host donde se vaya a ejecutar, es decir, se puede ubicar en cualquiera mientras se tenga una correcta interconexión entre los componentes que conforman el ecosistema de ROS. Para ello se debe tener en cuenta que:

- Solo se puede tener un nodo maestro. Por lo que se debe elegir una de las máquinas para que sea el dispositivo Maestro.
- Todos los nodos tienen que ser configurados para usar el mismo dispositivo maestro, lo cual se realiza a través de la variable de entorno ROS\_MASTER\_URI.
- Debe haber una comunicación bidireccional entre todos los pares de máquinas.
- Cada máquina debe presentarse a sí misma mediante un nombre o dirección IP válida en la red con la cual puedan identificarse mutuamente.



**Figura 4-8: Conexión roscore maestro-esclavo**

Para la puesta a punto de todos los dispositivos que formarán parte del sistema final hay que configurar lo siguiente:

- En la Nvidia Jetson, se la designa como nodo maestro:

```
$ export ROS_MASTER_URI=http://IP_MAESTRO:11311
$ export ROS_IP=IP_MAESTRO
```

- En las PCs de monitoreo o cualquier cliente que se quiera tener:

```
$ export ROS_MASTER_URI=http://IP_MAESTRO:11311
$ export ROS_IP=IP_CLIENTE
```

Para poder averiguar la IP de algún dispositivo basta con correr el comando:

```
$ hostname -I
```

Tener en cuenta que puede editarse el archivo “`~/.bashrc`”, como se ha explicado anteriormente, para no realizar la configuración de las IP cada vez que se ejecute una nueva terminal de comandos.

#### 4.4 PRUEBA

Dado que no se han definido pruebas específicas para los requerimientos no funcionales, se considerará suficiente el hecho de tener una versión de Sistema Operativo estable, diseñada específicamente para la robótica y compatible con la placa de desarrollo seleccionada.

A modo de demostración, se adjunta la siguiente captura de pantalla de la terminal (Figura 4-9) corriendo el proceso principal del Framework ROS sobre el sistema operativo Ubuntu 14.04, lo cual es indicativo de que están dadas todas las condiciones para comenzar a utilizar todas las herramientas que el mismo nos provee.

```
roscore http://turtlebot-N56VB:11311/
turtlebot@turtlebot-N56VB:~$ roscore
... logging to /home/turtlebot/.ros/log/5f237222-c341-11e7-8daf-6c71d957a22b/roslaunch-turtlebot-N56VB-2801.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

started roslaunch server http://turtlebot-N56VB:42501/
ros_comm version 1.11.21

SUMMARY
========
PARAMETERS
  * /rostdistro: indigo
  * /rosversion: 1.11.21
NODES
auto-starting new master
process[master]: started with pid [2813]
ROS_MASTER_URI=http://turtlebot-N56VB:11311/

setting /run_id to 5f237222-c341-11e7-8daf-6c71d957a22b
WARNING: Package name "hermesIII" does not follow the naming conventions. It should start with a lower case letter and only contain lower case letters, digits and underscores.
process[rosout-1]: started with pid [2826]
started core service [/rosout]
```

Figura 4-9: Roscore corriendo sobre Ubuntu 14.04

## 4.5 RESULTADOS

Se ha definido que se utilizará, para el proyecto Hermes 3, la placa de desarrollo NVIDIA Jetson TK1 junto con el sistema operativo Ubuntu 14.04 y el Framework Robot Operating System (ROS) en su versión “Indigo Igloo”.

La compatibilidad con múltiples arquitecturas de procesadores del framework utilizado, ha permitido que se instale y configure el mismo entorno de trabajo en otras computadoras que se utilizarán para la monitorización y que formarán parte del ecosistema de ROS del robot Hermes 3.

## 4.6 RIESGOS MITIGADOS

El desarrollo de esta iteración significó seguir el plan de mitigación definido en la Tabla 1-17: Exposición al riesgo) para determinados riesgos.

A medida que se avanza con el proyecto, al final de cada iteración, la incertidumbre baja y las probabilidades de ciertos riesgos se van mitigando.

En ésta iteración se puede percibir que los riesgos atacados fueron:

ID	Riesgo	Probabilidad	Impacto	Exposición Final Iteración	Exposición Inicio Iteración
RI-01	Sistema Operativo inestable	10%	2	0,2	0,6
RI-02	Incompatibilidad o avería de componentes	25%	2	0,5	1
RI-03	Intercomunicación de componentes ineficiente o ineficaz	30%	3	0,9	2,1
RI-04	Prestaciones insuficientes de componentes	40%	3	1,2	1,5
RI-05	Modificación de los requerimientos del proyecto	30%	4	1,2	1,2
RI-06	Dificultad en conseguir determinados componentes	30%	4	1,2	3,6
RI-07	Excesivo tiempo para cumplir los objetivos del proyecto	60%	5	3	3,5
RI-08	Reducción de la fuerza de trabajo	10%	3	0,3	0,4

Tabla 4-4: Riesgos mitigados en iteración 1

## 4.7 CONCLUSIONES

A lo largo de esta iteración se logró aplicar los requerimientos de software y hardware elegidos para el proyecto. Se logró la puesta en marcha de ROS sobre todos los dispositivos, sentando las bases para posteriormente enfocarse en el desarrollo de las funcionalidades específicas del Hermes III.

En cuanto a la mitigación de riesgos se ha tenido en cuenta lo siguiente:

- RI-01: La puesta en funcionamiento del sistema operativo junto con el framework ROS en el sistema embebido principal no produjeron mayores inconvenientes a nivel de software. Por ello, y sumando a los numerosos proyectos de robótica que existen utilizando la misma combinación de software y hardware, podemos concluir que el sistema operativo es robusto.
- RI-02: La cantidad de documentación en la nube existente sobre la utilización de diversos componentes destinados para robótica y que son completamente compatibles con ROS y Nvidia Jetson, ha disminuido dicha probabilidad ya que a su vez se ha comprobado que el framework ROS posee una gran cantidad de librerías que crece día a día y satisface con creces nuestros requerimientos.
- RI-03: Se ha estudiado la comunicación entre los principales componentes que tendrá nuestro sistema (NVIDIA Jetson - Arduino - Kinect), y se ha visto la performance de las comunicaciones, obteniendo en todos los casos resultados aceptables para un correcto funcionamiento del sistema final a desarrollar.
- RI-04: Dado lo mencionado anteriormente y las pruebas realizadas en la presente iteración, se concluye que los componentes más importantes del sistema cumplen los requerimientos hasta el momento. **Debe tenerse en cuenta que en las siguientes iteraciones se irán agregando nuevas funcionalidades y sensores que exigirán aún más al sistema en su conjunto, por ende, este riesgo disminuye muy levemente.**
- RI-06: Durante la iteración se han conseguido componentes muy específicos:
  - **La placa de desarrollo principal NVIDIA Jetson TK1**
  - **La placa de desarrollo Arduino Mega 2560**
  - **El sensor de profundidad y color Microsoft Kinect**

Estos componentes (a excepción de Arduino Mega) son provistos por sus fabricantes oficialmente y no existen imitaciones en el mercado, por ende, son difíciles de reemplazar. En el caso particular de la NVIDIA Jetson TK1, ésta tuvo que ser adquirida directamente de NVIDIA, ya que no existen proveedores tercerizados que las vendan oficialmente, mucho menos en Argentina. En cuanto a los componentes restantes de conseguir (baterías, sensores infrarrojos, contadores), existen numerosos fabricantes y proveedores de los mismos alrededor del mundo, por ende conseguirlos y/o reemplazarlos es más sencillo. Por esta razón el riesgo en cuestión, disminuye considerablemente en esta etapa del proyecto.

- RI-07: Este riesgo disminuirá al final de cada iteración siempre que los tiempos estipulados para cumplir con los objetivos del loop se cumplan. En este caso se ha demorado lo previsto.
- RI-08: En la medida que se avanza en el proyecto, menos impacta que disminuya el nivel de colaboración de un miembro del equipo, debido a que cada vez son menos las tareas pendientes.

# CAPÍTULO 5

## 5 ITERACIÓN 2: MODELADO Y SIMULACIÓN DEL ROBOT

### 5.1 INTRODUCCIÓN

La primera fase en la creación de un robot es su diseño y modelado. Ésto puede hacerse utilizando herramientas CAD tales como AutoCAD, Solid Works, Blender, entre otros. Uno de los principales propósitos del modelado de un robot es la simulación.

Las herramientas de simulación permiten verificar los posibles defectos en los diseños y validar el funcionamiento del robot antes de que éste pase a la etapa de manufactura.

Se sabe que, al trabajar con robots, se requiere de un espacio físico acondicionado, instrumentos de medición adecuados, recargas de baterías, etc. Además, los robots en etapas tempranas de desarrollo, debido a varios factores como condiciones del entorno, manufactura o defectos de diseño, tienden a fallar, por lo que hacer reiteradas pruebas con el mismo con una versión experimental puede acarrear costos económicos elevados, que se pueden evitar utilizando robots simulados. En las simulaciones, podemos modelar la realidad con distintos niveles de precisión y se pueden realizar numerosas pruebas sin el riesgo de avería de un componente de hardware del robot.

Todo lo simulado carecerá de validez y utilidad si los modelos matemáticos, que pretenden representar aspectos de la realidad física, no son los correctos.

Una de las posibilidades que ofrece ROS es realizar, de una manera parametrizable, un modelo matemático de un robot, y luego nos permite simularlo. A lo largo de esta iteración, se hará foco en las herramientas para tal fin que nos brinda el Framework.

### 5.2 REQUERIMIENTOS

Los requerimientos funcionales asociados a esta iteración son los siguientes:

RF6	Debería realizarse un modelo de simulación previo a la implementación
RF7	Debería tenerse un sistema de monitorización remoto de los sensores del robot
RF8	Podría realizarse un modelo tridimensional del robot

Tabla 5-1: Requerimientos funcionales de iteración 2

## 5.3 DESARROLLO

ROS incorpora en su meta-paquete llamado "robot\_model" todo lo necesario para la descripción de un modelo tridimensional de un robot mediante un archivo en formato URDF (Unified Robot Description Format), que pasará a describirse a continuación.

### 5.3.1.1 URDF

El Formato de Descripción de Robot Unificado (URDF) es un estándar XML de ROS para la representación de modelos robóticos.

Permite describir las siguientes características de un robot:

- La dinámica y cinemática
- La representación visual
- El modelo de colisión

#### 5.3.1.1.1 SINTAXIS URDF

Las siguientes etiquetas son comúnmente utilizadas en un archivo URDF:

- <link>: Esta etiqueta permite describir a cada cuerpo rígido que conforma el robot, definiendo su aspecto visual y sus propiedades físicas.

Particularmente el diseño visual, que implica tamaño, forma y color del componente, puede hacerse de manera manual o se puede importar un archivo en formato "Collada (.dae)" que se obtiene a partir de los formatos más comunes de archivos 3D de las herramientas CAD que actualmente existen en el mercado.

#### Sintaxis

```
<link name="<name of the link>">
    <inertial>.....</inertial>
    <visual> .....</visual>
    <collision>.....</collision>
</link>
```

#### Representación gráfica

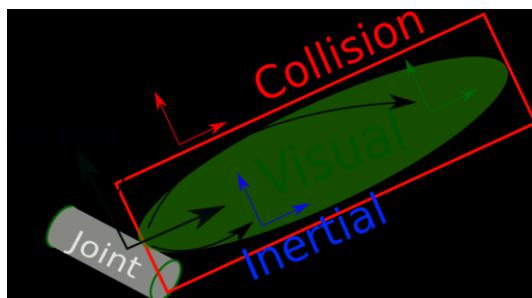


Figura 5-1: "Link" de una pieza

En la Figura 5-1 tenemos la sección “Visual” que representa un cuerpo rígido del robot y el área remarcada en rojo es la sección “Collision” que determina cuándo el simulador detectará una

colisión. Ambas secciones no necesariamente coinciden perfectamente, muchas veces debido a que la sección de colisión es la que representa el mayor costo computacional para el simulador, y con el fin de hacer la ejecución del modelo mas performante, se eligen formas geométricas simplificadas respecto a la visual.

- <joint>: Esta etiqueta representa la unión entre dos cuerpos rígidos (links) del robot. Podremos definir la cinemática y dinámica de las uniones, los límites de movimiento de las mismas y sus velocidades. Tenemos diferentes tipos de uniones tales como “revolute”, “continuous”, “prismatic”, “fixed”, “floating” y “planar”, de los cuales puede obtenerse información detallada en la documentación oficial de ROS.

### Sintaxis

```
<joint name="name of the joint">
  <parent link="link1"/>
  <child link="link2"/>
  <calibration .... />
  <dynamics damping .... />
  <limit effort .... />
  <origin .... />
</joint>
```

### Representación Gráfica

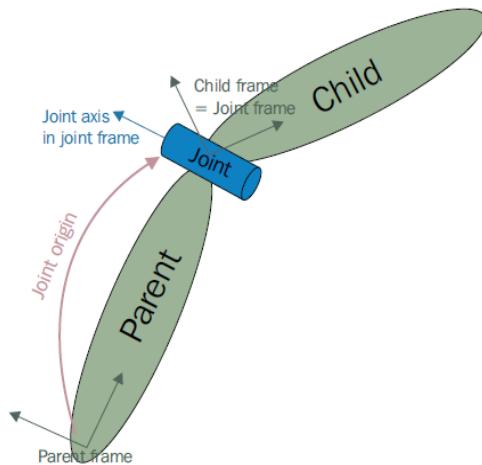


Figura 5-2: "Joint" de dos piezas

Un “joint” en URDF se forma entre dos “links”, el primero es llamado “parent link” y el segundo “child link” (Ver Figura 5-2).

- <robot>: Esta etiqueta encapsula a todo el modelo del robot que puede ser representado a través de URDF.

#### Sintaxis

```
<robot name="<name of the robot>">
  <link> ..... </link>
  <link> ..... </link>

  <joint> ..... </joint>
  <joint> ..... </joint>
</robot>
```

#### Representación Gráfica

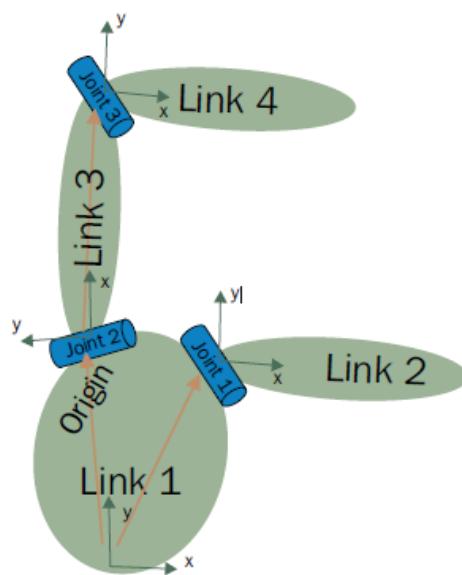


Figura 5-3: Modelo ejemplo de unión de varias piezas

El modelo final del robot que se desea modelar a través de URDF termina siendo una unión de “links” a través de “joints”.

- <gazebo>: Esta etiqueta es utilizada cuando se incluyen los parámetros de simulación para el software Gazebo dentro del URDF. Podremos indicar al simulador qué plugins predefinidos usar, qué propiedades de materiales debe utilizar, entre otros.

#### Sintaxis

```
<gazebo reference="link_1">
  <material>Gazebo/Black</material>
  <plugin name="plugin_name" filename="file_name" >
    ...
  </plugin>
</gazebo>
```

[24]

---

### 5.3.2 MODELO URDF DEL HERMES

Con el fin de comprender completamente las capacidades de las herramientas que brinda el Framework para la realización del diseño 3D y el modelo de simulación, se tomará como base al Hermes II, cuyas características y aspecto visual son bien conocidas. Luego, habiendo logrado un correcto comportamiento cinemático y dinámico del modelo, principalmente del sistema de locomoción, se modificará únicamente el aspecto visual para adecuarlo al diseño del futuro Hermes III que no distará mucho del modelo actual.

---

#### 5.3.2.1 CREACIÓN DE ARCHIVOS STL (STANDARD TRIANGLE LANGUAGE)

Es un formato de archivo informático de diseño asistido por computadora (CAD) que define geometría de objetos 3D, excluyendo información como color, texturas o propiedades físicas que sí incluyen otros formatos CAD.

Es el formato estándar para las tecnologías de fabricación aditiva. Utiliza una malla de triángulos cerrada para definir la forma de un objeto. Cuanto más pequeños son estos triángulos, mayor será la resolución y el tamaño del fichero final. Es aconsejable llegar a una solución de compromiso entre la resolución y el peso del fichero.

Los archivos STL pueden crearse a partir de dos clases de datos: nube de puntos o modelo CAD (superficies o sólidos) y casi todos los softwares pueden realizar una exportación a dicho formato.

Las nubes de puntos provienen del digitalizado de un modelo o entorno. Para la creación del STL, el software tratará de unir estos puntos de forma óptima teniendo en cuenta el procedimiento de digitalizado.

Para la generación del fichero STL del Hermes se utilizará el software de modelado 3D de Autodesk llamado “123D Design” dada su simplicidad de uso.

Se procede entonces al diseño de cada uno de los cuerpos rígidos que conforman el Hermes y que posteriormente serán relacionados entre sí por medio de un archivo URDF como se ha explicado anteriormente.

### 5.3.2.1.1 RUEDAS OMNIDIRECCIONALES

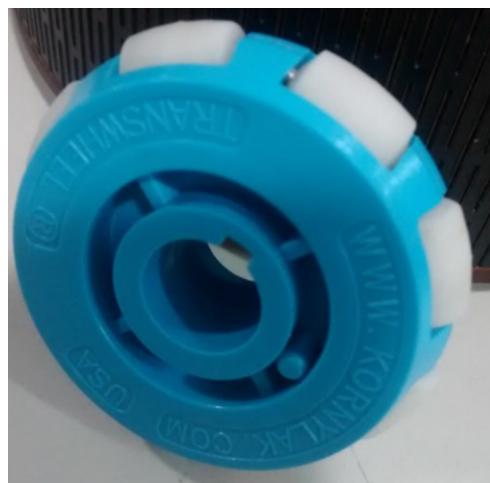


Figura 5-4: Rueda onmidireccional

Las ruedas omnidireccionales (Figura 5-4) están conformadas por dos elementos bien definidos. Por un lado, tenemos el cuerpo propiamente dicho en color celeste y por otro lado tenemos los rodillos en color blanco. Se necesita modelar a ambos por separado ya que las físicas de movimiento que tiene cada uno son diferentes.

#### Cuerpo de las ruedas omnidireccionales (Figura 5-5)

Dimensiones:

- Radio: 2.5 cm
- Ancho: 1.6 cm
- Cavidades donde se ubican los rodillos giratorios: 1.2 cm x 1.2 cm

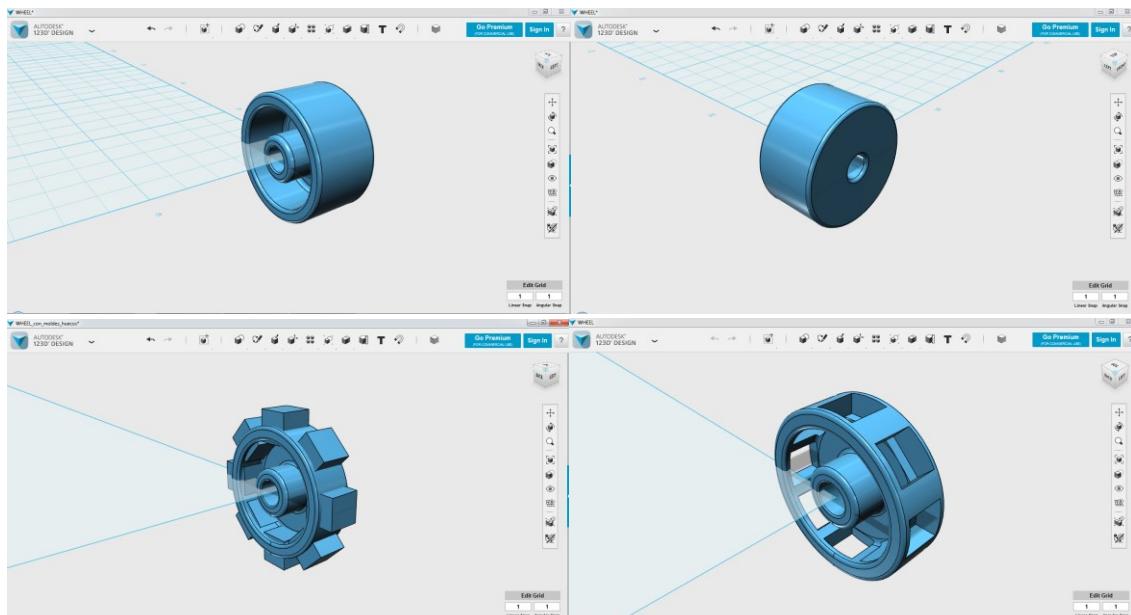


Figura 5-5: Diseño 3D de rueda onmidireccional

### Rodillos de las ruedas omnidireccionales (Figura 5-6)

Dimensiones:

- Radio: 3 mm
- Ancho: 6 mm
- Largo: 10 mm

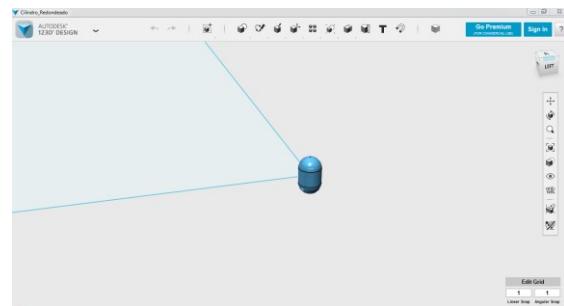


Figura 5-6:Rodillo de rueda onmidireccional

#### 5.3.2.1.2 CHASIS

Del mismo modo, se procedió a realizar el diseño 3D del chasis, por el momento igual al del Hermes II (Figura 5-7).



Figura 5-7: Chasis de Hermes II

Dimensiones:

- Compartimiento inferior: 23 cm x 23 cm x 4.5 cm
- Compartimiento superior: 16 cm x 16 cm x 3 cm
- Radio de las esquinas del compartimiento inferior: 8 cm
- Radio de las esquinas del compartimiento superior: 6 cm

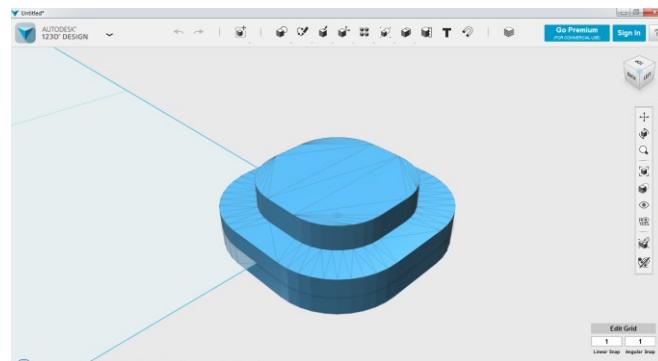


Figura 5-8: Diseño 3D de chasis de Hermes II

### 5.3.2.2 CREACIÓN DE ARCHIVOS COLLADA (COLLABORATIVE DESIGN ACTIVITY)

COLLADA es un formato de archivo de intercambio para aplicaciones 3D interactivas. Es administrado por el consorcio de tecnología sin fines de lucro, el Grupo Khronos, y ha sido adoptado por la ISO como una especificación disponible públicamente, ISO / PAS 17506.

Collada define un esquema XML estándar abierto para el intercambio de activos digitales entre diferentes aplicaciones de software de gráficos que de otro modo podrían almacenar sus activos en formatos de archivo incompatibles. Los documentos COLLADA que describen los activos digitales son archivos XML, por lo general identificados con una extensión de archivo ".dae".

#### 5.3.2.2.1 CONVERSIÓN DE ARCHIVOS STL A COLLADA

La conversión puede realizarse a través de software de diseño de animaciones 3D tales como "Autodesk Maya". Dado que no se justifica la instalación de este tipo de software únicamente para la conversión de archivo que se necesita, se realiza a través de un servicio online. Existen muchas opciones gratuitas. En este caso fue utilizado el siguiente:

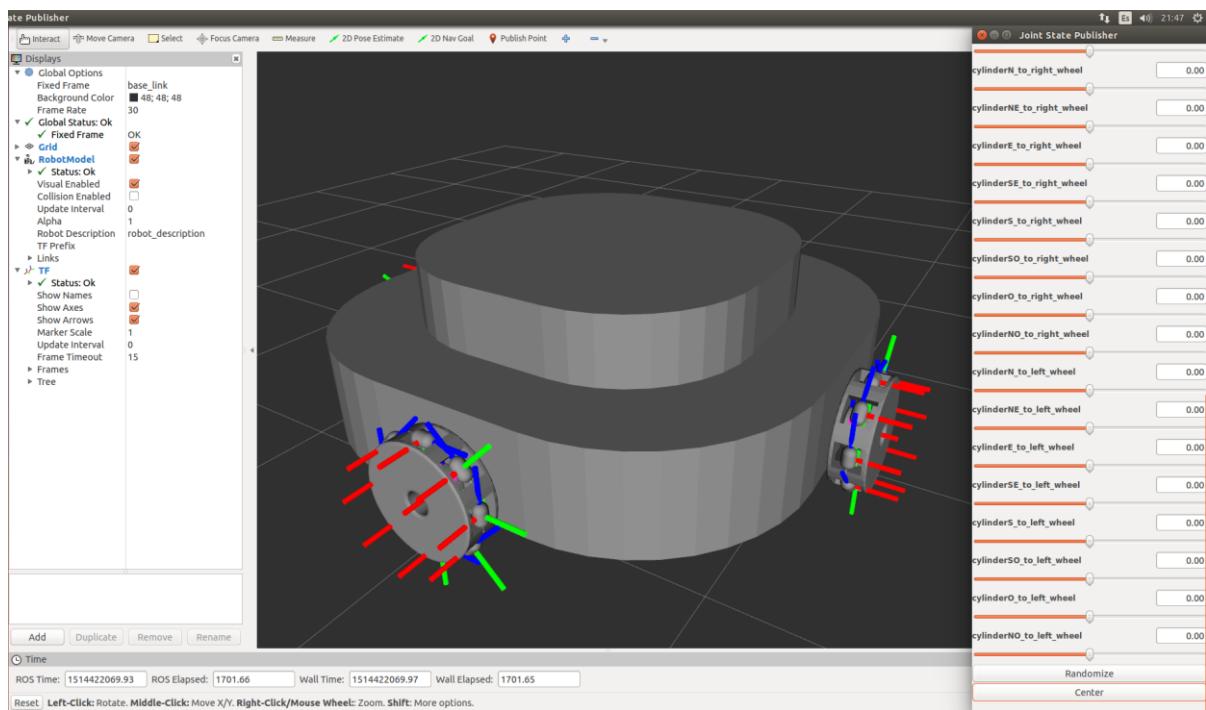
→ <http://www.greentoken.de/onlineconv/>

Una vez obtenidos los archivos de los "cuerpos rígidos" que componen al Hermes, éstos pueden modelarse y unirse entre sí con las propiedades físicas correspondientes, mediante un archivo URDF.

### 5.3.2.3 VISUALIZACIÓN DEL MODELO EN SOFTWARE DE MONITOREO

Una vez escrito el URDF del Hermes II, que hace uso de los archivos "Collada", se procede a visualizarlo.

Se ejecuta entonces el software de monitoreo RVIZ con el archivo URDF del Hermes II, para validar que la configuración de "links" y "joints" estuviese correcta. Incluso, mediante una GUI con *sliders*, podremos generar movimiento de los cuerpos rígidos sobre sus propios ejes según las propiedades que se haya definido para cada "Joint", como puede apreciarse en la siguiente Figura 5-9.



**Figura 5-9: Movimiento y visualización de cuerpos rígidos**

El archivo único que se ejecuta y permite obtener todo el entorno de prueba en el software RVIZ de la imagen anterior, es posible gracias a una herramienta del framework llamada “roslaunch” que se escribe en lenguaje XML y describe los nodos de ROS que se desean ejecutar, como así también parámetros de lanzamiento de los mismos. En particular, el utilizado en el ejemplo anterior puede verse en el anexo A.2 display.launch.

Cabe destacar que al ejecutarse RVIZ, el mismo lee por defecto un archivo de configuración llamado “default.rviz” ubicado en “`~/.rviz/`”. Por ende para poder visualizar correctamente el modelo del Hermes II, es necesario modificar este archivo por defecto o indicar al software dónde se encuentra el archivo de configuración personalizado, el cual se presenta en el anexo A.3 config.rviz. Por cuestiones de extensión del código de configuración, no se incluyen los ejes de coordenadas de cada cuerpo rígido que conforma el robot, pero ésto puede agregarse fácilmente en RVIZ seleccionando en la esquina inferior izquierda, el botón “Add” y luego seleccionando la opción con el nombre “TF”.

### 5.3.3 SIMULACIÓN

Los simuladores se basan en la dinámica de cuerpos rígidos para realizar los cálculos matemáticos de transformación necesarios para obtener movimientos coherentes de los objetos en el espacio tridimensional.

Se ha mencionado en una sección anterior, explicando la estructura de un archivo URDF, que el área de colisión que se define de un modelo tridimensional tenía a ser una figura geométrica simple. Esto no es trivial, dado que a mayor complejidad, mayor es el requerimiento de cálculo computacional de la herramienta de simulación, que llevado al extremo produce una baja considerable en la performance del programa.

#### 5.3.3.1 GAZEBO

ROS se integra perfectamente con el simulador 3D Gazebo instalando los paquetes necesarios. Éstos permiten una comunicación bidireccional entre el simulador y el ecosistema ROS,

es decir, Gazebo pasa a ser productor y consumidor de datos simulando ser un robot que implementa ROS, escuchando los tópicos a los que se suscribe y a su vez enviando los mensajes que produce el robot virtual, todo dentro de un entorno físico tridimensional simulado elegido y dispuesto por el usuario.

A fines de probar las capacidades de éste simulador, se procede a instalar los paquetes ROS para interactuar con el simulador Gazebo y un paquete de ejemplo para probar dicha interacción.

Instalación de “gazebo\_ros\_pkgs”:

```
user@hostname$ sudo apt-get install ros-indigo-gazebo-ros-pkgs ros-indigo-gazebo-ros-control
```

Instalación de los paquetes de prueba “turtlebot\_gazebo”:

```
user@hostname$ sudo apt-get install ros-indigo-turtlebot-gazebo
```

Por último, se ejecuta la simulación:

```
user@hostname$ source /opt/ros/indigo/setup.bash  
user@hostname$ roslaunch turtlebot_gazebo turtlebot_empty_world.launch
```

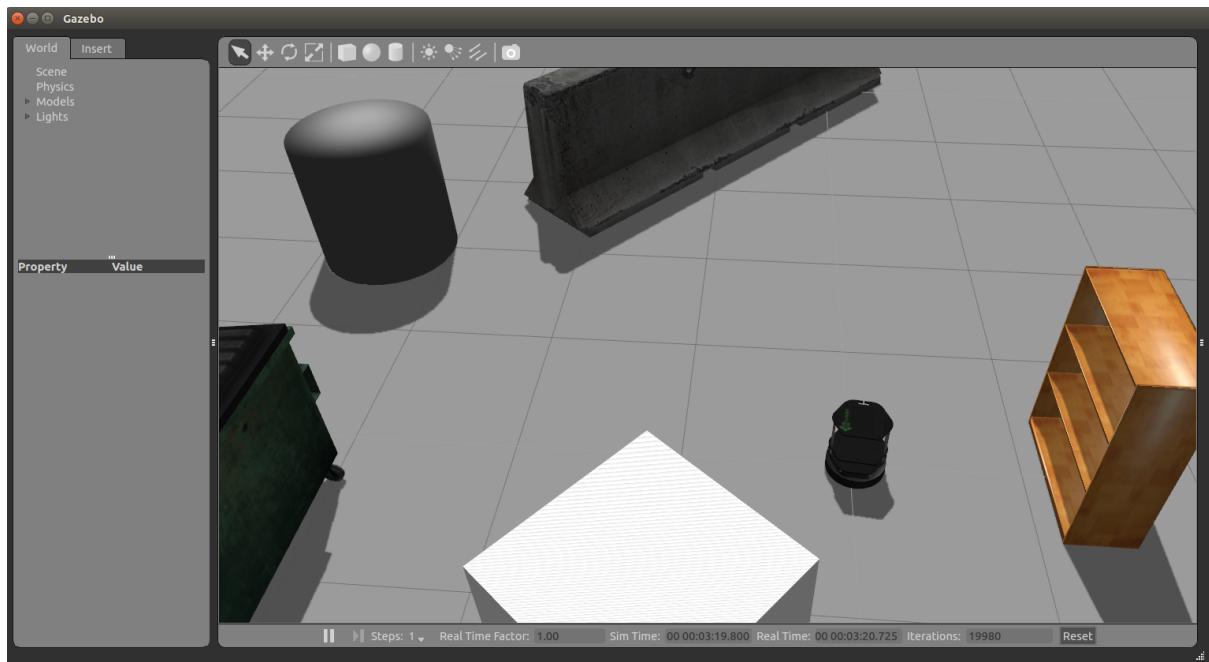


Figura 5-10: Simulación de turtlebot

Como resultado se obtiene lo que puede observarse en la Figura 5-10 y Figura 5-11, en donde se tiene un robot denominado “turtlebot” en un entorno con objetos que el mismo detecta a través de sus sensores simulados y los transmite en forma de tópicos de ROS que pueden ser visualizados por la herramienta de monitorización RVIZ, como si de un robot real se tratase.

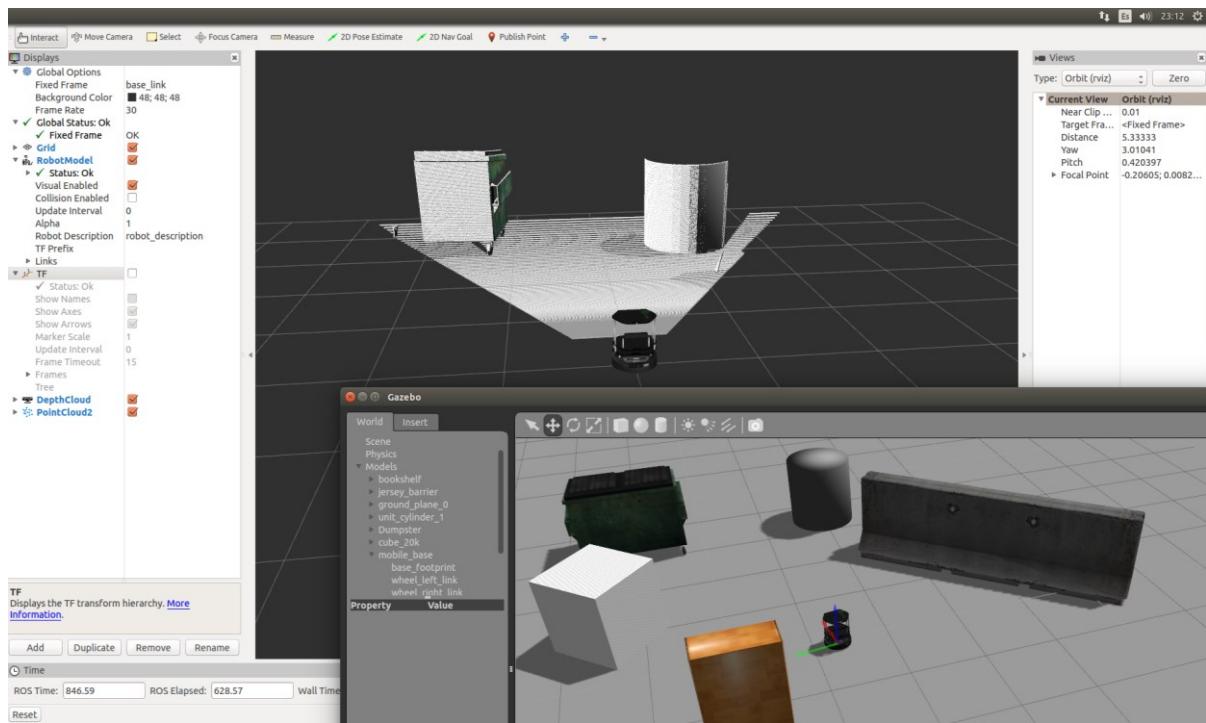


Figura 5-11: Visualización de sensores sobre simulación de turtlebot

Todo ésto es posible gracias al nivel de abstracción que posibilita una arquitectura robótica basada en el paso de mensajes normalizados, de tal manera que a un nodo receptor de un tópico ROS en la red no le hace falta ni tampoco tiene forma de saber si ese mensaje fue producido por un microcontrolador Arduino, por una placa de desarrollo Nvidia Jetson, por el simulador Gazebo o por una simple línea de código ejecutada en cualquier computadora de la red con Ubuntu + ROS:

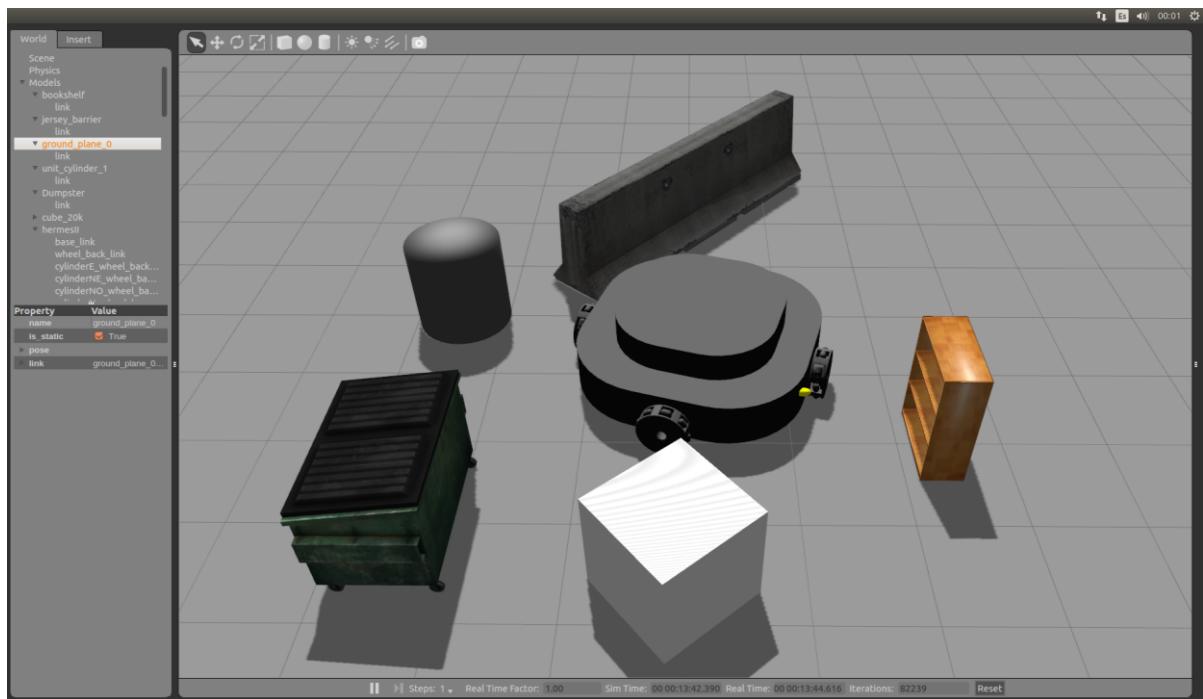
```
user@hostname$ rostopic pub my_topic std_msgs/String "hello there"
```

### 5.3.3.2 SIMULANDO EL ROBOT HERMES

Para la simulación del robot Hermes II se ha creado un archivo “roslaunch” que prepara al simulador Gazebo para trabajar con el modelo que se ha diseñado (Figura 5-12).

Se ejecuta entonces “gazebo.launch” que se incorpora en el anexo A.4 gazebo.launch:

```
user@hostname$ ~/catkin_ws/src/hermesIII/launch$ rosrun gazebo gazebo.launch
```

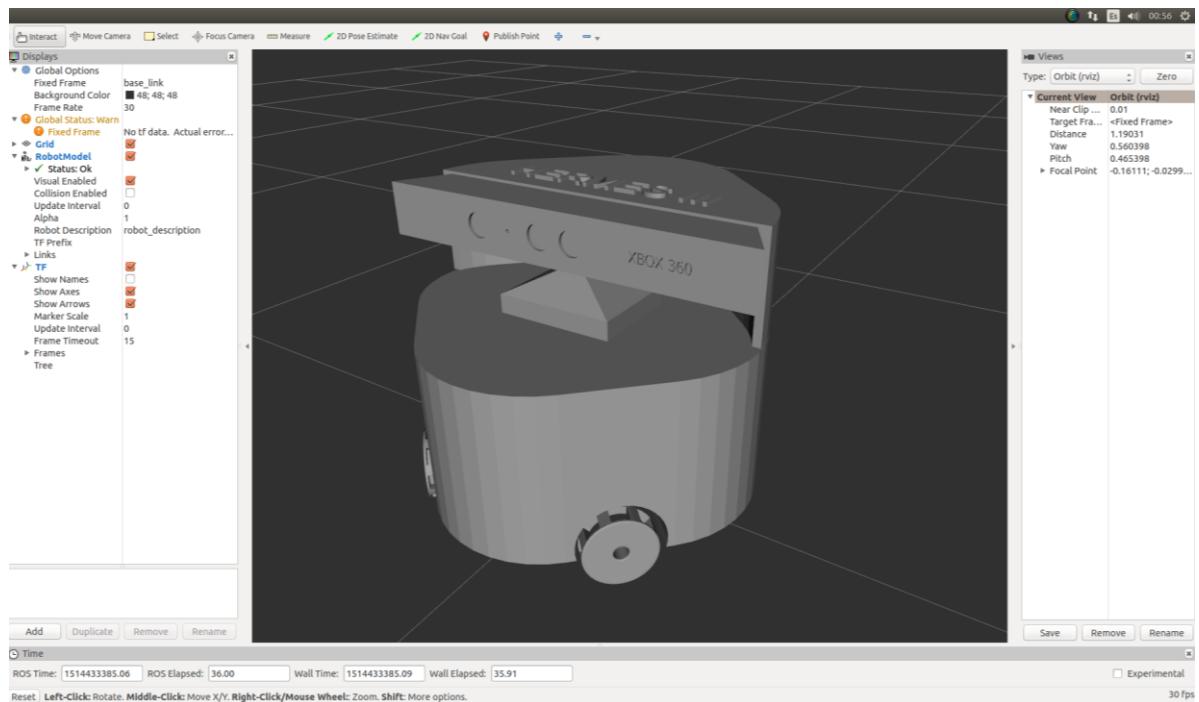


**Figura 5-12: Simulación de Hermes II**

Al desplazar el modelo 3D a través de su entorno, se ha detectado que el comportamiento de las ruedas omnidireccionales es por momentos errático y genera movimientos bruscos en el modelo. Se concluye que el uso de figuras de colisión muy complejas y detalladas hacen que el simulador tenga errores de cálculo en las dinámicas de los cuerpos. Debido a ésto, se decide aumentar la precisión de la simulación, obteniendo mejoras en las dinámicas, pero disminuyendo considerablemente el rendimiento del programa, el cual comienza a ralentizarse y no correr en tiempo real dado el volumen de datos que debe manejar.

Dada las dificultades y limitaciones que se han encontrado para la correcta representación del movimiento de las ruedas omnidireccionales, se ha acordado con el Director del Proyecto Integrador, continuar con las siguientes iteraciones para no retrasar el avance del Hermes 3.

Entonces se termina por diseñar un modelo URDF de lo que será el Hermes 3 (anexo A.5 modelo\_HermesIII.urdf), detallando solo su aspecto visual, sin propiedades físicas, ya que no será utilizado en el simulador Gazebo. El modelo será utilizado únicamente en el software de monitoreo RVIZ para representar al robot dentro del mapa que el mismo genere.



**Figura 5-13: Visualización de Hermes III**

Lo mostrado en la Figura 5-13 ha sido posible ejecutarlo mediante el *launchfile* “display\_hermesIII.launch” del anexo A.6 *display\_hermesIII.launch*.

## 5.4 PRUEBAS

Caso de prueba PRF6	
Objetivo	Modelar las propiedades físicas del robot en un software especializado, de tal manera de poder probar su dinámica de movimiento.
Prerrequisitos	<ul style="list-style-type: none"> <li>Se debe contar con el modelo 3D de las partes del robot que se desea simular</li> <li>Cada parte móvil del modelo 3D debe estar configurado con sus respectivos grados de libertad y propiedades físicas.</li> <li>Contar con el launch file “gazebo.launch”</li> <li>Se debe tener instalado: <ul style="list-style-type: none"> <li>Framework ROS</li> <li>Software Gazebo</li> <li>Paquete “gazebo_ros”</li> <li>Paquete “turtlebot_gazebo”</li> </ul> </li> </ul>
Pasos	Ver sección “Simulando al Robot Hermes”
Resultado esperado	El robot simulado debe tener un comportamiento dinámico y cinemático parametrizable.
Resultados obtenidos	<ul style="list-style-type: none"> <li>El modelo del robot se visualiza correctamente en el simulador Gazebo y cuenta con las propiedades físicas configuradas.</li> <li>La simulación no realiza cálculos precisos de la dinámica de cuerpos rígidos pequeños y complejos.</li> </ul>
Códigos de prueba necesarios	Anexos A.1 / A.5

Tabla 5-2: Prueba nro. 1 de iteración 2

Caso de prueba PRF7-1	
Objetivo	Visualizar el modelo 3D realizado del Hermes 3 en el software de monitorización.
Prerrequisitos	<ul style="list-style-type: none"> <li>Se debe contar con el modelo 3D de las partes que conforman el robot que se desea simular</li> <li>Se debe tener instalado: <ul style="list-style-type: none"> <li>Framework ROS</li> <li>Software RVIZ</li> </ul> </li> <li>Se debe tener configurado correctamente el software RVIZ cargando el archivo “config.rviz” provisto en los anexos.</li> <li>Contar con el launch file “display_hermesII.rviz” y “display_hermesIII.rviz”</li> </ul>
Pasos	Ver sección “Visualización del modelo en software de monitoreo”
Resultado esperado	Se debe ver en el software de monitorización el

	modelo 3D en la ubicación actual del robot acorde a las coordenadas de ubicación suministradas por el mismo.
Resultados obtenidos	<ul style="list-style-type: none"> <li>El modelo se visualiza correctamente y permite ser utilizado para representar la posición en tiempo real del robot durante su funcionamiento.</li> <li>Se pueden hacer pruebas de los grados de libertad del robot en caso de tenerlos configurados.</li> </ul>
Códigos de prueba necesarios	Anexos A.2 / A.3 / A.5 / A.6

Tabla 5-3: Prueba nro. 2 de iteración 2

Caso de prueba PRF8-1	
Objetivo	Visualizar el diseño 3D creado en cualquier software CAD.
Prerrequisitos	<ul style="list-style-type: none"> <li>Contar con el modelo 3D del robot en formato STL.</li> <li>Se debe tener instalado: <ul style="list-style-type: none"> <li>Autodesk 123D™</li> <li>Sistema Operativo Windows 7 o superior</li> </ul> </li> </ul>
Pasos	Ver sección “Creación de archivos STL (Standard Triangle Language)”
Resultado esperado	Se debe visualizar el modelo realizado en cualquier software CAD.
Resultados obtenidos	<ul style="list-style-type: none"> <li>El modelo en formato STL puede visualizarse en casi cualquier software CAD e incluso puede visualizarse con herramientas disponibles en la nube. Particularmente Github posee un visualizador de los archivos STL subidos al repositorio.</li> <li>El formato STL puede ser convertido a otros formatos de modelado 3D.</li> </ul>
Códigos de prueba necesarios	<a href="https://github.com/hmalatini/hermes3/blob/dev/src/hermesIII/model/stl/Hermes3_2017.stl">https://github.com/hmalatini/hermes3/blob/dev/src/hermesIII/model/stl/Hermes3_2017.stl</a>

Tabla 5-4: Prueba nro. 3 de iteración 2

## 5.5 RESULTADOS

Se logró realizar una simulación correcta del Hermes II, sin embargo se tuvo inconvenientes a la hora de simular las ruedas omnidireccionales, ya que el cálculo de las propiedades físicas de inercia e interacción con otros cuerpos rígidos de cada rodillo no eran precisos y ésto provocaba que los mismos giraran erróneamente. Ajustando la precisión de los cálculos de parámetros físicos, la simulación mejoraba, pero su performance disminuía considerablemente, perdiendo su ejecución en tiempo real. Se ha investigado en foros oficiales del Framework ROS pero no se encontraron respuestas concretas para solucionar este problema.

## 5.6 RIESGOS SUPERADOS

El análisis de mitigación de riesgos al finalizar la presente iteración es el siguiente:

ID	Riesgo	Probabilidad	Impacto	Exposición Final Iteración	Exposición Inicio Iteración
RI-01	Sistema Operativo inestable	8% 	2	0,16	0,2
RI-02	Incompatibilidad o avería de componentes	25%	2	0,5	0,5
RI-03	Intercomunicación de componentes ineficiente o ineficaz	30%	3	0,9	0,9
RI-04	Prestaciones insuficientes de componentes	40%	3	1,2	1,2
RI-05	Modificación de los requerimientos del proyecto	30%	4	1,2	1,2
RI-06	Dificultad en conseguir determinados componentes	30%	4	1,2	1,2
RI-07	Excesivo tiempo para cumplir los objetivos del proyecto	50% 	5	2,5	3
RI-08	Reducción de la fuerza de trabajo	10%	3	0,3	0,3

Tabla 5-5: Riesgos mitigados en iteración 2

## 5.7 CONCLUSIONES

Se concluye que el modelado y la simulación es un paso importante dentro del desarrollo de un robot. Ésto permite probar los algoritmos de funcionamiento del robot, incluso antes de tenerlo construido físicamente.

Dado los tiempos de realización del proyecto, se ha decidido seguir adelante y no se ha profundizado en configuraciones avanzadas del simulador Gazebo, el cual no ha dado resultados estables con el modelo de ruedas omnidireccionales, pero se presume que se debe a una mala configuración de parámetros. También es posible que en versiones más recientes del software Gazebo, no se tengan dichos inconvenientes.

En cuanto al análisis de mitigación de riesgos se ha considerado lo siguiente:

- RI-01: La utilización del software de monitoreo RVIZ y del software de simulación Gazebo sin problemas en el sistema embebido NVIDIA Jetson, ratifican aún más la estabilidad del sistema operativo dada las exigencias computacionales de dichos programas.
- RI-07: No haber logrado una simulación satisfactoria de las ruedas omnidireccionales no es un hecho bloqueante para continuar con las siguientes iteraciones del proyecto, por lo que se ha decidido continuar y por ende respetar el tiempo previsto de la presente iteración.

# CAPÍTULO 6

## 6 ITERACIÓN 3: IMPLEMENTACIÓN DE SENsoRES Y ACTUADORES

### 6.1 INTRODUCCIÓN

Los sensores son una parte fundamental de cualquier robot móvil, porque son los que permiten adquirir datos acerca del estado del robot y del estado de su entorno en cada instante de tiempo, lo cual es esencial para concebir movimientos controlados en el espacio.

Además, se implementarán los actuadores, que serán los responsables de dotar de movilidad al robot. Éstos serán motores de corriente continua que funcionarán acorde a los datos producidos por los sensores luego de ser procesados por el sistema de control.

En el presente capítulo se desarrollarán las capacidades sensoriales del robot Hermes III y en posteriores iteraciones se procesarán los datos que se obtengan de los mismos para lograr un movimiento controlado.

### 6.2 REQUERIMIENTOS

Los requerimientos funcionales asociados a esta iteración son los siguientes:

RF2	Debe poder auto localizarse
RF3	Debe poder realizar un mapa
RF4	Debe tener un sistema de control de movimiento eficaz

Tabla 6-1: Requerimientos funcionales de iteración 3

### 6.3 DESARROLLO

Para el proyecto Hermes III se busca utilizar la cantidad justa y necesaria de sensores, por el hecho que sobredimensionar los requerimientos en este aspecto complejiza innecesariamente el sistema y en la robótica ésto conlleva consecuencias, tales como: mayor consumo de energía, mantenimiento más costoso y sistema más propenso a errores.

Siguiendo la premisa anterior, se comenzará por la puesta en funcionamiento de los sensores necesarios para aplicar técnicas de SLAM (Simultaneous Localization And Mapping) basadas en capturas de imágenes RGB y de profundidad, lo cual nos permitirá cumplir con los primeros dos requerimientos.

Para el último requerimiento, el robot no puede basarse únicamente en los datos de su entorno para controlar sus movimientos, dado que la calidad de éstos datos depende de muchos factores y variables sobre los que no se tiene control, haciéndolo dependiente de condiciones externas para poder moverse correctamente. Por ésta razón es necesario sentir de la manera más directa posible, las

velocidades de cada una de las ruedas que son las encargadas de producir el movimiento del robot. Ésto se implementará mediante “*encoders*”.

### 6.3.1 ACTUADORES

#### 6.3.1.1 MOTORES DE CORRIENTE CONTÍNUA



Figura 6-1: Motores de C.C.

Los motores DC que serán utilizados (Figura 6-1), están diseñados específicamente para movilizar las ruedas de un robot. Ésto se hace evidente por su estructura con caja reductora, que permite tener un mayor torque sacrificando velocidad, y puede verse un eje blanco que sobresale a ambos lados del motor, lo cual brinda la posibilidad de colocar una rueda de un lado y un encoder del otro para sensar la velocidad de giro de la misma.

Estos motores son del tipo “*Brushed*”, los cuales no son los más eficientes del mercado, pero sí son los mas económicos y con buenos resultados en cuanto a torque a bajas revoluciones (Ver Figura 6-2).

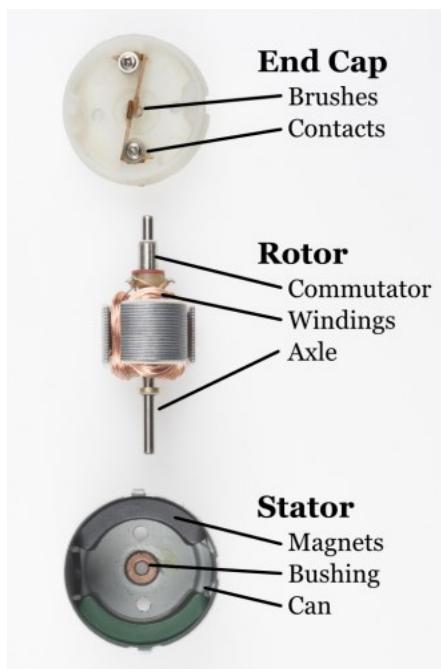


Figura 6-2: Componentes de un motor

Existiendo muchas herramientas *online* en internet [25] que calculan rápidamente el torque necesario de los motores acorde a las características físicas del robot a diseñar, se ha utilizado uno de

ellos, el cual ha arrojado los siguientes resultados de torque para las características previstas para el Hermes III(Figura 6-3, Figura 6-4 y Figura 6-5).

Eficiencia del motor	80	%	
Torque del motor	3.91621.	Kilogramo	Centímetro
Velocidad requerida	1	m/s	
Aceleración requerida	0.5	m/s <sup>2</sup>	
Diámetro de la rueda	5	Centímetro	
Velocidad del motor	250	rpm	

**Figura 6-3: Parámetros de cálculo**

Eficiencia del motor	60	%	
Torque del motor	5.22161.	Kilogramo	Centímetro

**Figura 6-4: Torque del motor con eficiencia del 60%**

Eficiencia del motor	80	%	
Torque del motor	3.91621.	Kilogramo	Centímetro

**Figura 6-5: Torque del motor con eficiencia del 80%**

A grandes rasgos, los cálculos matemáticos que realizan por detrás estas herramientas se basan en cuestiones fundamentales de la física clásica, específicamente de la rama de la dinámica de los cuerpos, solicitando datos tales como:

- Peso del robot: para deducir su inercia.
- Velocidad y aceleración: para deducir su momento lineal o moméntum.
- Características mecánicas (diámetro de las ruedas, eficiencia del motor, velocidad del motor): para deducir cómo será la transformación de la energía eléctrica, que alimenta a los motores, en energía mecánica que moverá al robot.

[26]

Se consideró una eficiencia mínima del motor de un 60%, aunque en la realidad para los motores *brushed* la eficiencia suele encontrarse entre un 75% y un 80%. El valor seleccionado se debe a que se reutilizarán motores de prototipos anteriores del robot Hermes y por ende pueden contar con cierto desgaste que disminuye su eficiencia.

Investigando acerca de las características técnicas de los motores a utilizar, los cuales son bastante estándar dentro del ámbito de la robótica con Arduino, se encuentran las siguientes especificaciones:

#### 6.3.1.1.1 CARACTERÍSTICAS TÉCNICAS

- Rango de alimentación: 3 a 6 V
- Velocidad: 125 rpm @ 6V
- Consumo sin carga: 70mA @ 6V
- Tamaño motoreductor: 6.5x2.2x1.85cm
- Peso motoreductor: 28.4g
- **Torque de bloqueo: 5.5 kg-cm @ 6V (Máximo torque o torque de bloqueo)**

Se ha resaltado lo referente al torque, dado que cumpliría con los requisitos calculados anteriormente, haciendo del motor idóneo para el presente proyecto.

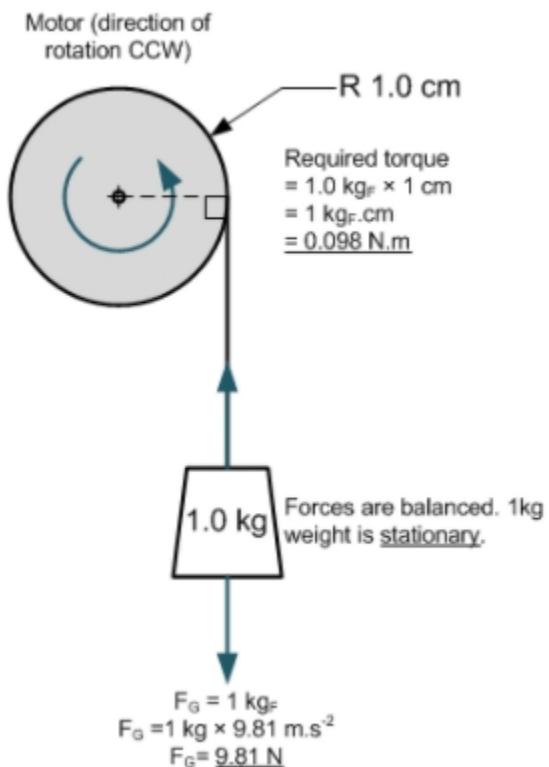


Figura 6-6: Cálculo de fuerzas

Con 5.5 kg-cm significa que el motor con una rueda de 1 cm de radio puede soportar un peso de hasta 5.5kg dispuestos según la Figura 6-6.

### 6.3.1.1.2 CONTROLADORES (DRIVERS)



Figura 6-7: Controlador de motores

Para la alimentación de los motores será necesario un circuito integrado especial que permita manipular de manera segura la corriente eléctrica de los motores y a su vez brinde la posibilidad de controlar la polaridad de los bornes donde se conectan los mismos para cambiar su sentido de giro.

Esta solución se ofrece en el mercado en un solo integrado, comúnmente denominado “Driver Dual para Motores L298N” (Figura 6-7) el cual posee a su vez un regulador de voltaje LM7805 para alimentar la parte lógica del integrado L298n. A continuación, en la Figura 6-8 se presenta un esquemático del dispositivo.

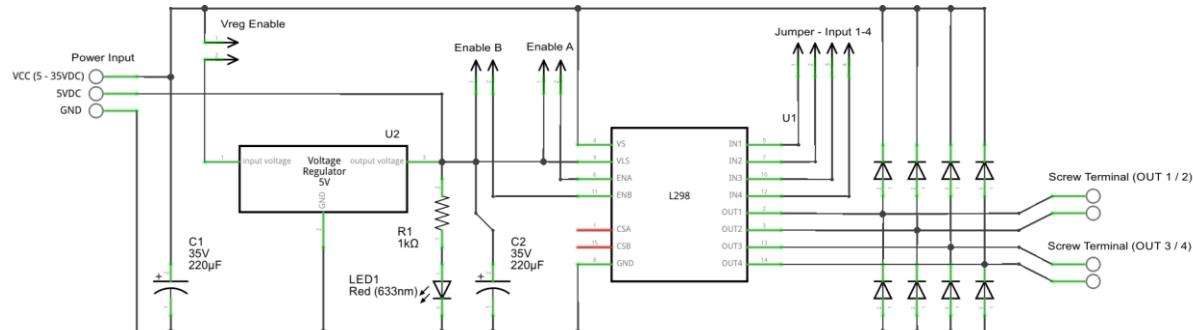


Figura 6-8: Diagrama esquemático del controlador

Otras de las grandes ventajas que nos ofrece este integrado, es que al poseer un regulador de voltaje con salida de 5V al exterior, podremos utilizarlo para alimentar a los sensores y otros embebidos de bajo consumo que se utilicen.

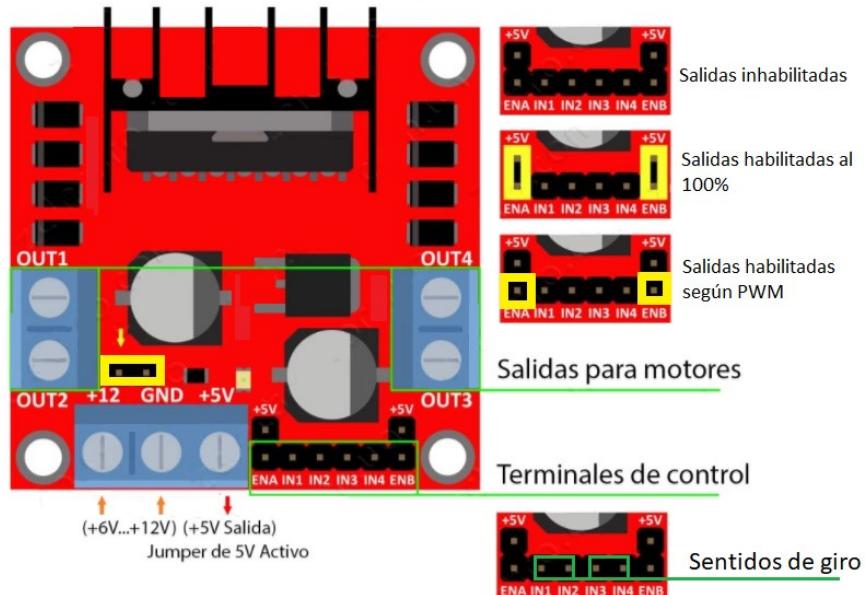


Figura 6-9: I/O del controlador

En la Figura 6-9 se puede apreciar que debe colocarse un jumper para activar la salida de 5V del regulador y además se muestra cómo deben ser utilizados los terminales de control. [27]

El sentido de giro de un motor estará definido por dos pines cuyos valores establecerán la polaridad de los terminales de alimentación del motor respectivamente. Existen dos pares de pines, uno por cada motor.

### 6.3.2 SENSOR DE IMÁGENES Y PROFUNDIDAD

Éste sensor será fundamental para dotar al robot Hermes 3 de la capacidad de detectar obstáculos en su camino, realizar un mapa de su entorno y auto localizarse.

#### 6.3.2.1 KINECT

La puesta en funcionamiento del sensor Microsoft Kinect no es una tarea trivial. Deben instalarse las librerías correctas, compatibles con la arquitectura ARM, Ubuntu 14.04 y ROS Indigo Igloo, para poder obtener datos de éste sensor sobre la NVIDIA Jetson TK1.

A continuación, se detallará los pasos a seguir que permitirán una correcta instalación y funcionamiento del sensor dentro del ecosistema ROS.

Primero se instalará todo el paquete de ROS “openni”, lo cual facilitará la tarea posterior de integrar los datos obtenidos del sensor Microsoft Kinect con el ecosistema de ROS:

```
$sudo apt-get install ros-indigo-openni*
```

Luego procederemos a la compilación manual de las librerías “libfreenect”. Para ésto se necesitará de antemano en el sistema, ciertas herramientas de compilación y dependencias de librerías, las cuales deben instalarse:

```
$ sudo apt-get install cmake freeglut3-dev pkg-config build-essential libxmu-dev libxi-dev libusb-1.0-0-dev
```

Habiendo realizado lo anterior con éxito, se procede a descargar el repositorio de la librería “libfreenect”:

```
$ git clone git://github.com/OpenKinect/libfreenect.git
```

El proceso de compilación a seguir es el siguiente:

```
$ cd libfreenect  
$ mkdir build  
$ cd build  
$ cmake ..  
$ make  
$ sudo make install
```

---

### 6.3.2.2 AUTOSUSPENDIDO DEL USB

Existe un inconveniente con la configuración por defecto de los puertos USB en la NVIDIA Jetson, que se apagan cuando no están siendo utilizados para ahorrar energía. Si esto sucede cuando el sensor Kinect está conectado, el dispositivo no será detectado por el sistema embebido.

Una solución es ejecutar un comando automáticamente en cada booteo que altera este comportamiento. Para ello se debe editar el script de booteo del sistema “/etc/rc.local” de la siguiente manera:

```
$ sudo gedit /etc/rc.local
```

Luego, añadiendo esta línea de código cerca del final del archivo pero antes de la línea “exit”:

```
# Disable USB auto-suspend, since it disconnects some devices such as webcams on  
Jetson TK1.
```

```
echo -1 > /sys/module/usbcore/parameters/autosuspend
```

Se guarda el archivo y Luego se reinicia la NVIDIA Jetson para asegurarnos que los cambios tengan efecto:

```
$ sudo reboot
```

---

### 6.3.2.2.1 PROGRAMAS DE PRUEBA DEL SENSOR DE PROFUNDIDAD

Una vez que la Jetson se encuentra encendida, se conecta el sensor Kinect y se procede a ejecutar programas de ejemplo que vienen incluídos en la descarga del repositorio “libfreenect”.

```
$ cd libfreenect/build/bin
```

Dentro de dicha carpeta encontraremos variados programas de prueba, de los cuales ejecutaremos el siguiente para verificar el funcionamiento del sensor de profundidad.

```
$ sudo ./freenect-glview
```

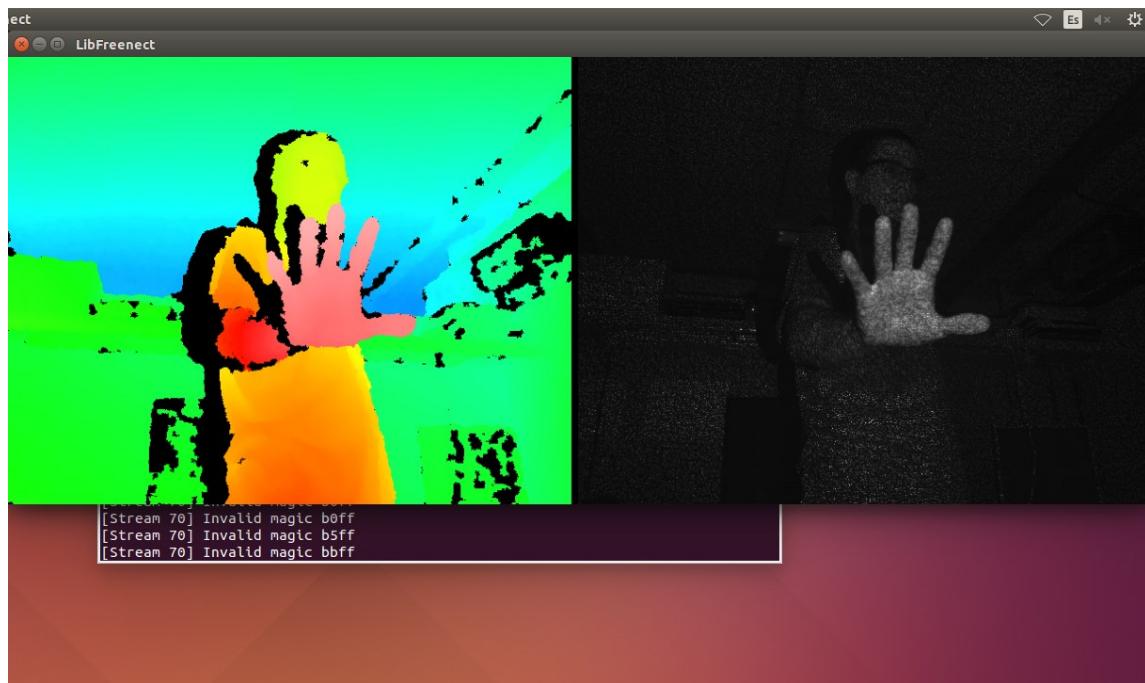


Figura 6-10: Imágenes obtenidas del sensor de profundidad y del sensor IR por parte de Kinect

Aquellas áreas oscuras que se aprecian en la imagen izquierda de la Figura 6-10, son indicativos de que el patrón de luz infrarroja emitido por el dispositivo Kinect, no es detectado por su sensor infrarrojo y por lo tanto no se tienen datos de dichas áreas. Ésto se debe principalmente a dos razones, por un lado a las propiedades poco reflectivas de ciertos objetos y por el otro al efecto sombra que se genera por la superposición de objetos delante del sensor, siendo que la fuente emisora infrarroja y el sensor infrarrojo no se encuentran en el mismo punto. Por las propiedades de la imagen, se deduce que la emisión de luz se encuentra a la derecha y el receptor a la izquierda, generando el efecto sombra sobre el cuerpo. La ubicación de las lentes puede verse en la sección Kinect.

Ésto es un dato no menor, porque implica que el robot Hermes III puede no detectar determinados obstáculos y ésto se traduce en una limitación en cuanto a las propiedades del entorno en el cual el robot podrá desenvolverse correctamente.

#### 6.3.4 SENSOR DE VELOCIDAD

Para los sensores de velocidad de las ruedas no se contará con un sensor de caja negra como el Microsoft Kinect, el cual se conecta y disponibiliza los datos pre procesados de sus sensores.

Para éste caso, se diseñarán y desarrollarán todas las etapas de implementación del sensor de velocidad, en dos partes bien diferenciadas:

1. Obtención de datos del fenómeno físico a medir
2. Procesamiento, transformación y disponibilización de los datos

##### 6.3.4.1 1. OBTENCIÓN DE DATOS DE UN FENÓMENO FÍSICO

Se ha reutilizado una plataforma robótica existente en el laboratorio, perteneciente a un prototipo del Hermes II que intentó utilizar los sensores optoacopladores para el control de velocidad de las ruedas, pero no se tuvo éxito en ese entonces.

Esta plataforma, que se muestra en la Figura 6-11, será refuncionalizada para ser usada en el Hermes III, ahorrando tiempo de diseño y fabricación de una nueva base robótica.

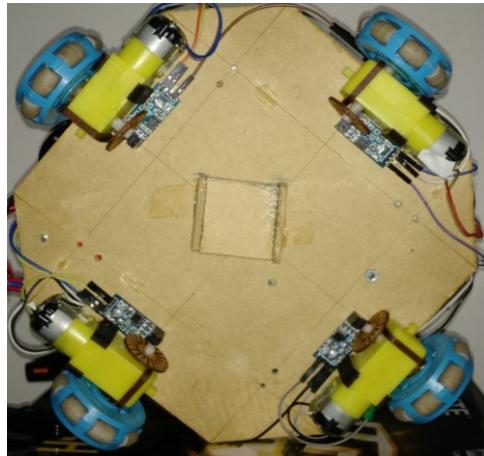


Figura 6-11: Vista inferior del prototipo refuncionalizado

Para contabilizar el giro de las ruedas, se utiliza el sensor óptico FC-03 que implementa el integrado LM393. Su implementación se muestra a continuación en la Figura 6-12.



Figura 6-12: Vista del sensor acoplado al motor

#### 6.3.4.1.1 MEDICIÓN DE LA SEÑAL CUADRADA DE LOS SENSORES

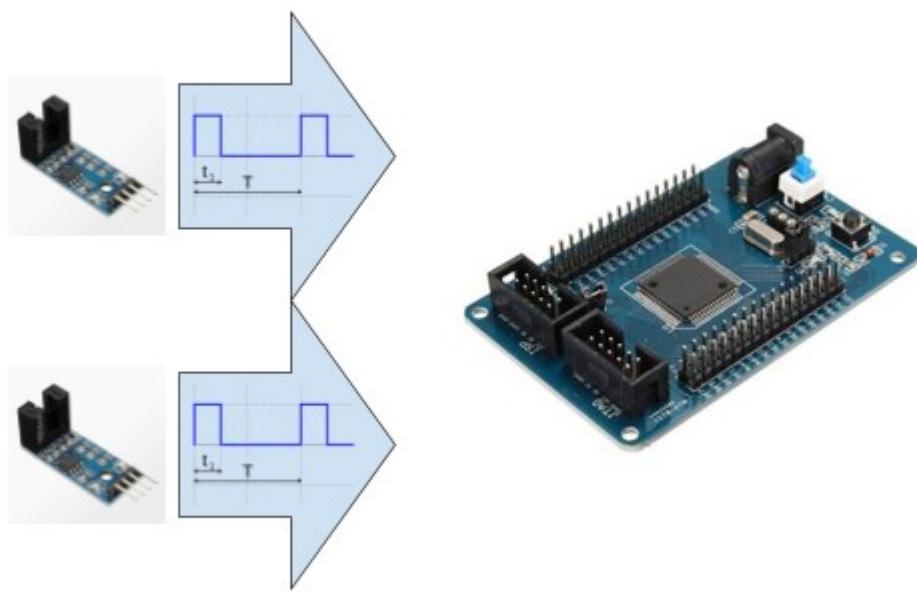


Figura 6-13: ATmega128 M128 AVR Minimum Development Board

Las ondas cuadradas generadas por dos sensores FC-03 podrán ser procesadas por un microcontrolador AVR ATmega 128A, haciendo uso de sus 2 Timers de 16 bits en modo contador asíncrono (Ver Figura 6-13). Dado que cada rueda cuenta con un sensor optoacoplador y se tiene una cantidad limitada de contadores en el integrado en cuestión, se utilizarán dos placas de desarrollo simultáneamente.

En primera instancia, se deberán configurar los procesadores de cada placa para utilizar el oscilador de cristal externo de 8Mhz que poseen.

En base a la hoja de datos del microcontrolador, los FUSES deben quedar configurados de la siguiente manera:

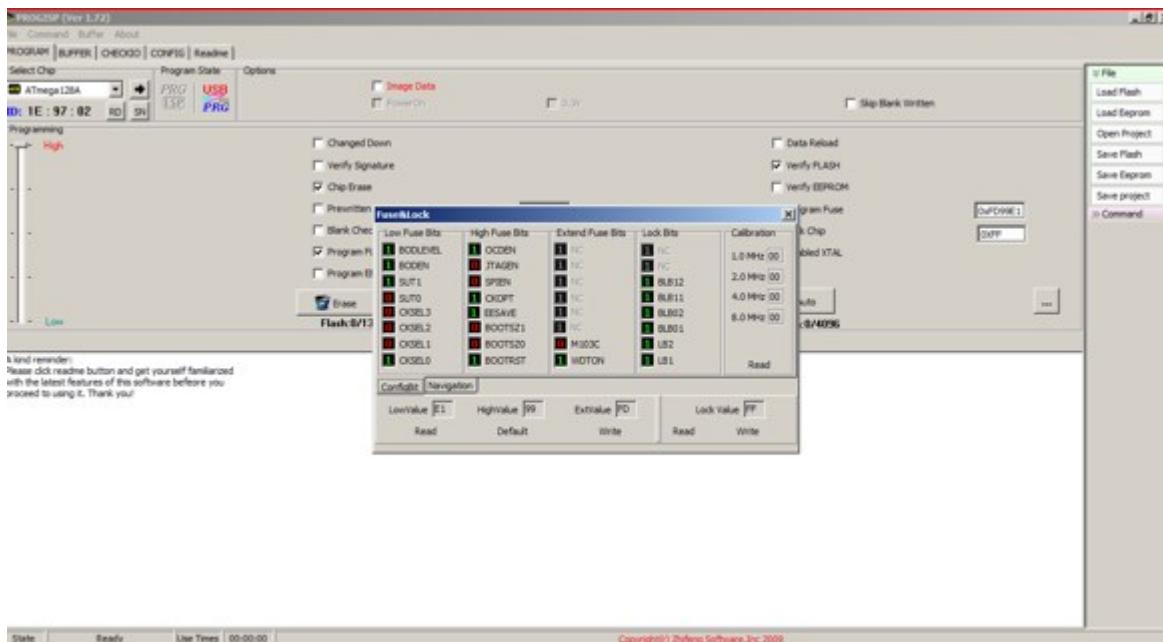


Figura 6-14: Configuración de FUSES del microcontrolador

El programa utilizado para configurar los FUSES del microcontrolador y grabar los programas en formato HEX, es “PROGISP” en su versión 1.72.



Figura 6-15: Programador ISP genérico

A su vez, para conectar las placas de desarrollo a la PC, se hace uso de un programador ISP genérico para microcontroladores de la familia ATMEtal que internamente posee un procesador ATmega 8. Una imagen de la misma se muestra en la Figura 6-15.

Teniendo listas las configuraciones iniciales, se podrá desarrollar el programa a ejecutar en el procesador Atmega 128A en lenguaje C++ y en la IDE de Arduino, habiendo instalado “MegaCore” de MCUDude disponible en repositorio público de Github [28]. Con ello, podremos generar el archivo HEX que luego será grabado en el microcontrolador con el programa y dispositivo mencionados anteriormente (Ver Figura 6-16).

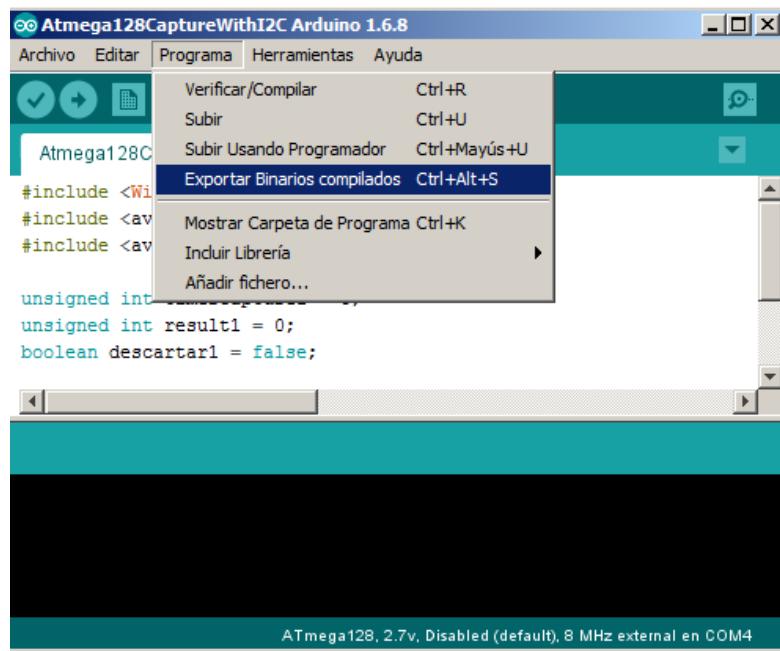


Figura 6-16: Generación de binarios mediante IDE Arduino

A continuación se presenta el diagrama de flujo del programa en la Figura 6-17, cuyo código se presenta en el anexo B.1 Atmega128CaptureWithI2C.ino, en el cual podrá verse la lógica de interrupciones utilizada para la medición del período de una onda cuadrada.

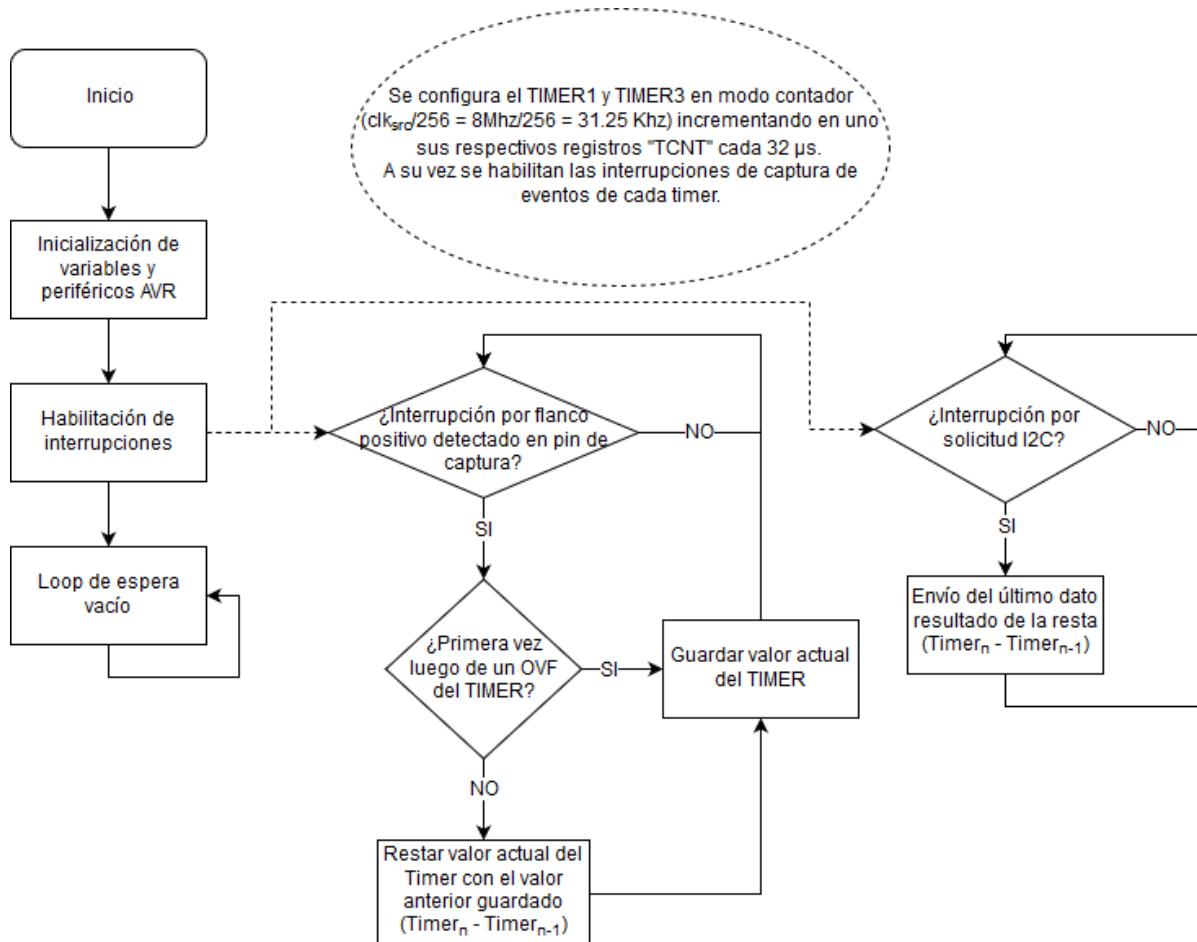


Figura 6-17: Diagrama de flujo del programa que corre sobre el microcontrolador ATmega128

Se tienen dos fuentes de interrupción:

1. A través de los pines de captura de cada TIMER del procesador, al que estarán conectados los respectivos sensores FC-03.
2. A través de los pines SCL y SDA del protocolo I<sup>2</sup>C.

La solicitud de información por I<sup>2</sup>C será realizada por un microcontrolador de la plataforma Arduino, que obtendrá la información del período de la señal cuadrada, la transformará en dato de velocidad traslacional lineal acorde a las dimensiones de la rueda en cuestión y la enviará en formato de tópicos ROS a través de la interfaz Serial por la que estará conectada a la NVIDIA Jetson TK1.

### 6.3.4.2 2. PROCESAMIENTO, TRANSFORMACIÓN Y DISPONIBILIZACIÓN DE LOS DATOS

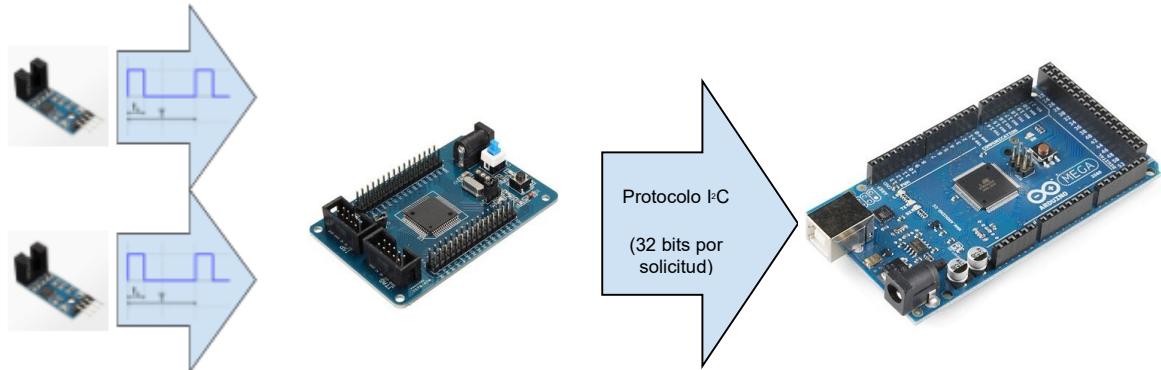


Figura 6-18: Flujo de datos entre ATmega128 y Arduino

Por cada solicitud de información, el procesador ATmega 128A disponibiliza los datos de sus dos sensores. Cada dato es de 16 bits, por ello el total de 32 bits por solicitud (Ver Figura 6-18).

A continuación, en la Figura 6-19, se presenta un diagrama de flujo que explica el programa que se ejecuta en la placa de desarrollo Arduino Mega 2560, cuyo código se expone en el anexo B.2 arduino\_controller\_hermes3\_v2.5\_with\_atmega128a.ino

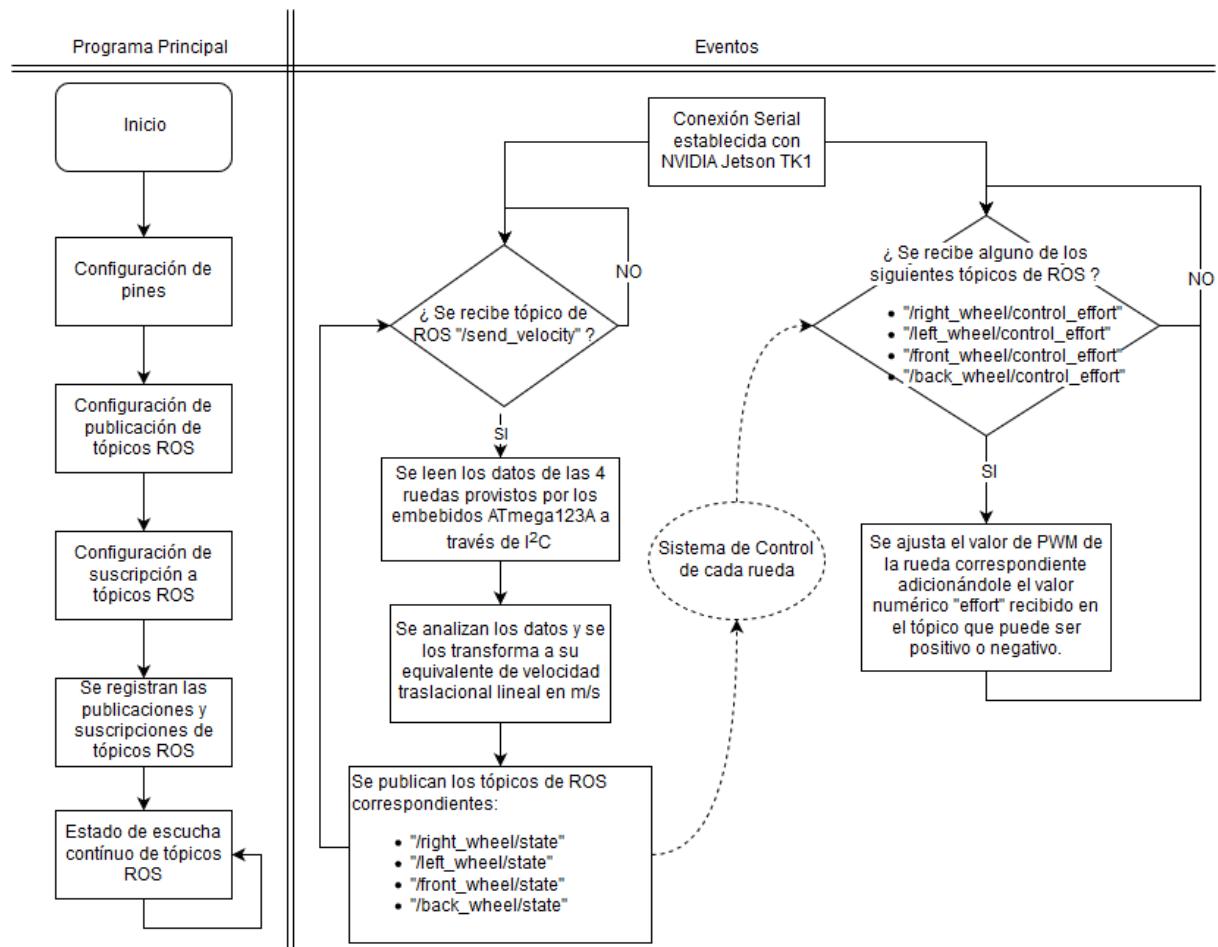


Figura 6-19: Diagrama de flujo del programa que corre sobre Arduino

Los eventos pueden considerarse como interrupciones que son atendidas por el sistema embebido Arduino. La configuración de cada tópico al que se suscribe, implica definir la función que será llamada al recibirla, que se programará como si fuese una rutina de atención de una interrupción.

Es muy importante destacar que la placa Arduino proveerá la información de las velocidades de las ruedas de manera pasiva. Ésto quiere decir que la placa NVIDIA será la que solicitará los datos al microcontrolador publicando el tópico ROS “/send\_velocity”. Por cada vez que lo haga, obtendrá los datos de velocidad de las cuatro ruedas en ese instante de tiempo. Mas adelante, ésta solicitud se configurará para hacerla de manera automática a una determinada frecuencia, para lograr un correcto funcionamiento del sistema de control de velocidad a implementar.

#### 6.3.4.2.1 OBTENCIÓN DE VELOCIDAD LINEAL EQUIVALENTE

El dispositivo Arduino recibe por parte de los microcontroladores ATmega128a, datos numéricos de 16 bits por cada rueda. Éste número binario representa la cantidad de ciclos de reloj del TIMER que han pasado entre el paso de una ranura a otra del *encoder*.

Dada la configuración del prescaler, como se verá mas adelante, cada  $32\mu s$  el valor del TIMER se incrementa en uno, y en base a ello puede calcularse sencillamente el período de la señal cuadrada mediante la siguiente ecuación:

*(Valor de 16 bits recibido por I2C) \*  $32\mu s$  = Período de la señal cuadrada de un encoder*

**Ecuación 6.1: Período de la señal cuadrada de un encoder**

Por ende, si se conoce la cantidad de ranuras que posee el *encoder* y el perímetro de circunferencia de la rueda, podremos obtener la velocidad de la misma realizando los siguientes cálculos:

Perímetro de la rueda:

$$2 * \pi * radio = 2 * \pi * 2.5cm = 15.71 cm$$

**Ecuación 6.2: Perímetro de la rueda**

Perímetro entre cada ranura:

$$\frac{15.71 cm}{cantidad\ de\ ranuras\ del\ encoder} = \frac{15.71 cm}{20} = 0.78 cm = 0.0078 m$$

**Ecuación 6.3: Distancia entre cada ranura de la rueda**

Entonces, teniendo además el tiempo que transcurre entre una ranura y la siguiente (período de señal cuadrada), podemos obtener fácilmente la velocidad de giro de la rueda.

$$\frac{0.0078}{\text{período señal cuadrada}} = \frac{\text{metros}}{\text{segundos}} = \text{velocidad de la rueda}$$

Ecuación 6.4: Velocidad de la rueda

#### 6.3.4.2.2 DETERMINACIÓN DEL PRESCALER

La determinación del valor del prescaler no es una tarea trivial. Ha sido seleccionado en base a la precisión de las mediciones que se desean tener. Cuanto más pequeño fuese el período del TIMER, tendremos mayor precisión en la medición de tiempo, pero se llegará mucho antes al *overflow*. Esto último nos limita el período de tiempo máximo que podrá medirse dado el algoritmo que se aplica en las placas ATmega 128a, que descarta la medición de un valor si entre medio ha ocurrido un *overflow*.

Por ende, con un prescaler definido en 256 y una fuente de reloj de 8Mhz, se tendrá un período mínimo medible de 32μs.

$$\frac{\text{prescaler}}{\text{clk}_{src}} = \frac{256}{8 \text{ MHz}} = 32\mu\text{s}$$

Ecuación 6.5: Período de prescaler

Con éste valor, se llegará al valor máximo de un registro de 16 bits a los 2.1 segundos.

$$(32 * 10^{-6})s * 2^{16} = 2.1 \text{ segundos}$$

Ecuación 6.6: Tiempo hasta la cuenta máxima

$$\frac{0.0078 \text{ metros}}{2.1 \text{ segundos}} = 0.0037 \frac{\text{metros}}{\text{segundos}}$$

Ecuación 6.7: Error de velocidad

La velocidad mínima define la precisión de las mediciones, pudiendo tener un error de ± 0.0037 m/s teniendo un error de lectura de ± 1 en el registro de 16 bits que recibe la placa Arduino.

Manejando éstos rangos de valores y considerando que, por experimentación sobre los motores, la velocidad mínima de funcionamiento de las ruedas será de 0.2 m/s y la velocidad máxima de 0.8 m/s antes que los motores comiencen a deteriorarse, podemos concluir que la precisión de las mediciones de velocidad son aceptables para los requerimientos del proyecto Hermes III.

---

### 6.3.4.3 TÓPICOS ROS

#### 6.3.4.3.1 ARDUINO ROS\_SERIAL

---

Para lograr que la placa de desarrollo NVIDIA pueda conectarse por serial e interpretar los tópicos publicados por Arduino, deben instalarse:

```
$ sudo apt-get install ros-indigo-rosserial-arduino  
$ sudo apt-get install ros-indigo-rosserial
```

#### Compilación de la librería para Arduino

Una vez instalados los paquetes anteriores en la NVIDIA Jetson, se procede a compilar la librería ROS que se utilizará para la programación de microcontroladores Arduino.

```
$ cd <sketchbook>/libraries
```

Teniendo previamente instalada la IDE de Arduino, “<sketchbook>” es la ubicación donde se almacenan los programas de la IDE y donde se encuentra la carpeta “libraries”.

Si previamente ya existiese una librería “ros\_lib”, la borraremos ya que ésta puede presentar problemas de compatibilidad con la versión actual de ROS que se esté utilizando.

```
$ rm -rf ros_lib
```

Luego se deberá compilar la librería de C++ con el ejecutable “make\_libraries.py” incluído en uno de los paquetes descargados anteriormente.

```
$ rosrun rosserial_arduino make_libraries.py
```

Habiendo hecho ésto, se tendrá la carpeta de la librería “ros\_lib” (como se muestra en la Figura 6-20) que deberá ubicarse entre las librerías del IDE Arduino (independientemente del sistema operativo sobre el que esté instalado) y al iniciar el IDE nos aparecerán todos los códigos ejemplo de ROS.

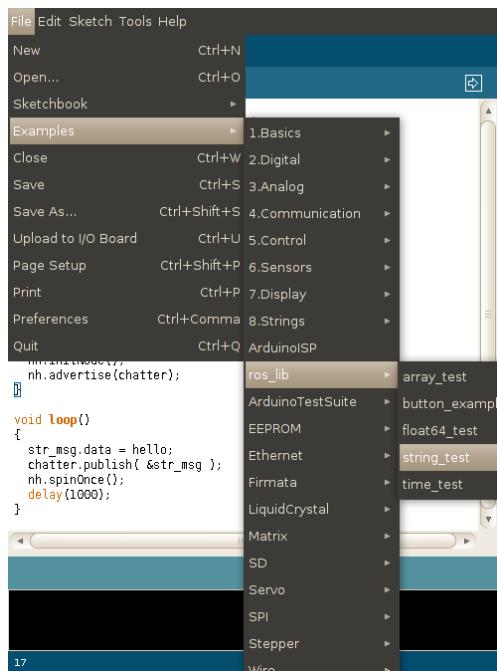


Figura 6-20: Códigos de ejemplo de ROS sobre Arduino

Luego deben seguirse los siguientes pasos para lograr que los tópicos de velocidad de las ruedas publicadas por Arduino puedan ser escuchados por cualquier nodo de ROS en la red a través de la NVIDIA Jetson. [29]

#### 6.3.4.3.2 INTERCONEXIÓN ARDUINO - NVIDIA JETSON TK1

Una vez grabado el programa del anexo

B.2 arduino\_controller\_hermes3\_v2.5\_with\_atmega128a.ino” en la Arduino Mega, debe acoplarse este dispositivo a la red o ecosistema de ROS para que pueda interactuar con los demás nodos que conforman el sistema Hermes III. Para ello, habiendo realizado correctamente los pasos anteriores, se tienen todas las condiciones dadas para únicamente conectar la placa de desarrollo Arduino por medio de un puerto serial USB a la placa NVIDIA Jetson y correr en ésta última el siguiente comando en consola:

```
$ rosrun rosserial_python serial_node.py /dev/serial/by-id/usb-
Arduino_www.arduino.cc_0042_5533834353935110A0B0-if00 _baud:=115200
```

Existen dos datos importantes a considerar en el comando anterior. El primero de ellos es el nombre del dispositivo arduino conectado por el puerto serial, el mismo se puede averiguar fácilmente viendo el contenido de la carpeta “/dev/serial/by-id” en el embebido NVIDIA. Luego, la segunda cuestión a tener en cuenta es el *baud rate* que por defecto viene preestablecido en 57600, pero que, por cuestiones de velocidad en transferencia de datos, se ha incrementado a 115200 baudios. Para ello se deben modificar las líneas 64 y 78 del archivo “ArduinoHardware.h” presente en la carpeta “<sketchbook>/libraries/ros\_lib” antes de compilar y cargar el código en la placa de desarrollo Arduino.

```

ubuntu@tegra-ubuntu:~$ roscl hermesIII/script/
ubuntu@tegra-ubuntu:~/hermes3/src/hermesIII/script$ ./arduino_ros.sh
[INFO] [WallTime: 1497396647.113000] ROS Serial Python Node
[INFO] [WallTime: 1497396647.130385] Connecting to /dev/serial/by-id/usb-Arduino_www.arduino.cc__0042_5533834353935110A0B0-if00 at 115200 baud
[INFO] [WallTime: 1497396649.293864] Note: publish buffer size is 512 bytes
[INFO] [WallTime: 1497396649.294707] Setup publisher on /right_wheel/state [std_msgs/Float64]
[INFO] [WallTime: 1497396649.302926] Setup publisher on /back_wheel/state [std_msgs/Float64]
[INFO] [WallTime: 1497396649.313245] Setup publisher on /left_wheel/state [std_msgs/Float64]
[INFO] [WallTime: 1497396649.323746] Setup publisher on /front_wheel/state [std_msgs/Float64]
[INFO] [WallTime: 1497396649.329189] Setup publisher on /PWM_LeftValue [std_msgs/Int16]
[INFO] [WallTime: 1497396649.335547] Setup publisher on /PWM_RightValue [std_msgs/Int16]
[INFO] [WallTime: 1497396649.347482] Setup publisher on /PWM_BackValue [std_msgs/Int16]
[INFO] [WallTime: 1497396649.354381] Setup publisher on /PWM_FrontValue [std_msgs/Int16]
[INFO] [WallTime: 1497396649.364133] Note: subscribe buffer size is 512 bytes
[INFO] [WallTime: 1497396649.365208] Setup subscriber on /send_velocity [std_msgs/UInt8]
[INFO] [WallTime: 1497396649.374329] Setup subscriber on /left_wheel/control_effort [std_msgs/Float64]
[INFO] [WallTime: 1497396649.391031] Setup subscriber on /right_wheel/control_effort [std_msgs/Float64]
[INFO] [WallTime: 1497396649.400508] Setup subscriber on /front_wheel/control_effort [std_msgs/Float64]
[INFO] [WallTime: 1497396649.418665] Setup subscriber on /back_wheel/control_effort [std_msgs/Float64]

```

Figura 6-21: Comunicación de Arduino con roscore

De esta manera con las configuraciones anteriores realizadas, se podrá lograr una comunicación a una velocidad de 115200 baudios entre ambas placas de desarrollo y particularmente la Arduino Mega pasará a ser un nodo de ROS más en la red, “hablando” el mismo idioma que los demás componentes que conforman el robot Hermes III (Ver Figura 6-21).

#### 6.3.4.4 DIAGRAMA DE SECUENCIA

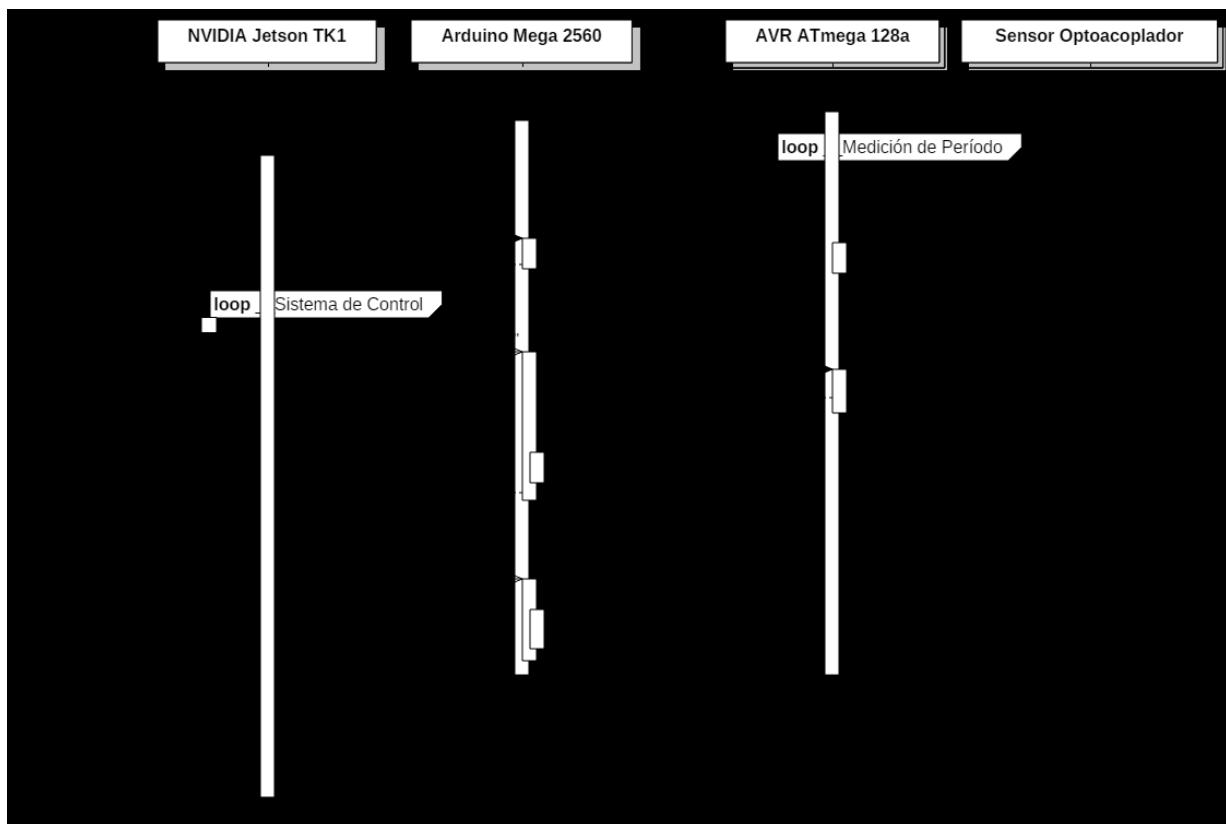
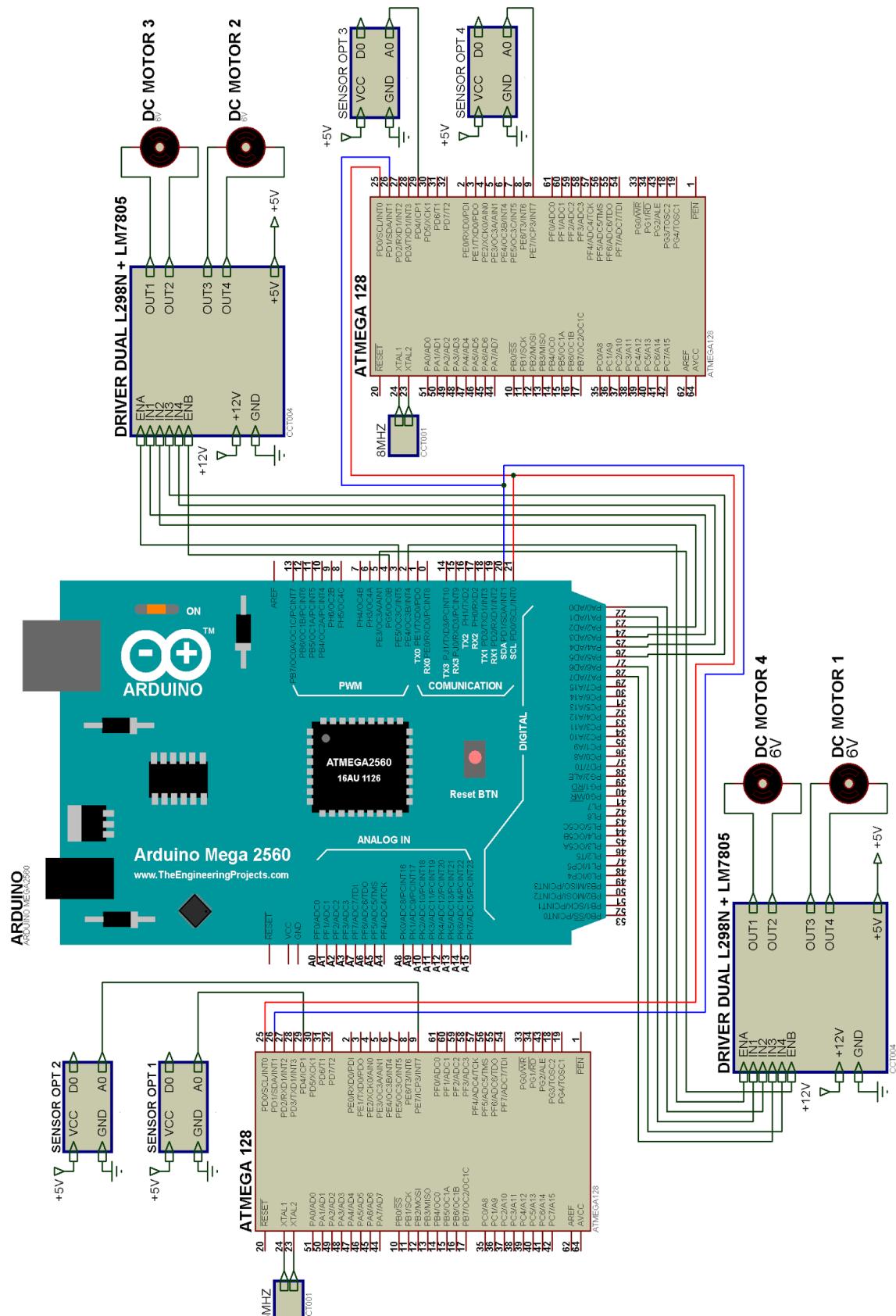


Figura 6-22: Diagrama de secuencia entre los distintos microcontroladores

### 6.3.5 DIAGRAMA ESQUEMÁTICO



**Figura 6-23:** Diagrama esquemático de conexión entre Arduino y ATmega128

## 6.4 PRUEBAS

Caso de prueba PRF2-1	
Objetivo	Conectar el sensor de profundidad y que el sistema embebido principal procese los datos de la misma.
Prerrequisitos	<ul style="list-style-type: none"> <li>Se debe contar con los drivers “libfreenect” del sensor Microsoft Kinect V1 correctamente instalados en la placa NVIDIA Jetson TK1.</li> <li>Se debe tener correctamente alimentado el dispositivo Kinect acorde a sus especificaciones y conectarla a uno de los puertos USB de la placa NVIDIA Jetson.</li> <li>Contar con los programas de prueba de la librería “libfreenect”</li> </ul>
Pasos	Ver sección “Sensor de imágenes y profundidad”
Resultado esperado	Se puede visualizar los datos del sensor de profundidad mediante algún software de monitorización.
Resultados obtenidos	<ul style="list-style-type: none"> <li>Se visualizó correctamente los datos de profundidad producidos por el sensor Kinect.</li> </ul>
Códigos de prueba necesarios	Incluídos en la librería “libfreenect”

Tabla 6-2: Prueba nro. 1 de iteración 3

Caso de prueba PRF4-1	
Objetivo	Conectar los motores con sus respectivos <i>encoders</i> y que los datos que produzcan puedan ser procesados por el microcontrolador elegido.
Prerrequisitos	<ul style="list-style-type: none"> <li>Tener realizadas las conexiones acorde a la sección “Diagrama Esquemático” del presente capítulo.</li> <li>Tener instalados correctamente en el robot Hermes III los motores con sus respectivos <i>encoders</i> y sensores.</li> <li>Tener configurados los FUSES de las placas de desarrollo ATmega 128a para su correcto funcionamiento.</li> <li>Contar con una PC con sistema operativo Windows para grabar las placas ATmega.</li> <li>Tener instalado, en la PC con Windows, el software necesario para generar los códigos fuentes de los programas de las placas ATmega y Arduino.</li> <li>Contar con el programador ISP y los controladores necesarios para grabar las placas ATmega.</li> <li>Contar con la IDE Arduino y los controladores correspondientes para la placa Arduino Mega 2560.</li> </ul>

	<ul style="list-style-type: none"> <li>• Tener grabados los programas provistos en el presente capítulo en los respectivos microcontroladores.</li> <li>• Lograr una exitosa conexión entre la placa de desarrollo Arduino y la placa NVIDIA Jetson según la sección “Interconexión Arduino - NVIDIA Jetson TK1”</li> </ul>
Pasos	<p>Habiendo ejecutado exitosamente los requisitos de la presente prueba, basta con ejecutar los siguientes comandos en la NVIDIA Jetson, en el orden dado y en diferentes terminales:</p> <pre>→ roscore → rostopic echo /right_wheel/state → rostopic pub /send_velocity std_msgs/UInt8 1</pre> <p>Y deberá verse la respuesta por parte de la placa Arduino en la terminal donde se ha ejecutado el segundo comando dado anteriormente.</p>
Resultado esperado	Se debe poder obtener la velocidad de una rueda.
Resultados obtenidos	<ul style="list-style-type: none"> <li>• La placa de desarrollo Arduino publicaba datos de velocidad coherentes con la realidad.</li> <li>• La placa de desarrollo Arduino respondía correctamente a todas las solicitudes de datos dadas por el tópico de ROS “/send_velocity”, emitidas por la placa NVIDIA Jetson.</li> </ul>
Códigos de prueba necesarios	Anexo B

Tabla 6-3: Prueba nro. 2 de iteración 3

## 6.5 RESULTADOS

Al finalizar la presente iteración se ha logrado poner en funcionamiento todos los componentes de hardware que formarán parte del robot Hermes III, habiendo desarrollado las interfaces de comunicación necesarias centralizando todo en la NVIDIA Jetson TK1, sobre la cual se trabajará en las siguientes iteraciones, desarrollando software, específicamente nodos de ROS, para lograr la integración de todas las partes y que funcionen armónicamente como un único sistema.

Se han realizado pruebas sobre cada uno de los componentes (sensores y actuadores) de manera independiente y todos se han logrado integrar correctamente al ecosistema de ROS.

## 6.6 RIESGOS SUPERADOS

El análisis de mitigación de riesgos al finalizar la presente iteración es el siguiente:

ID	Riesgo	Probabilidad	Impacto	Exposición Final Iteración	Exposición Inicio Iteración
RI-01	Sistema Operativo inestable	6% 	2	0,12	0,16
RI-02	Incompatibilidad o avería de componentes	10% 	2	0,2	0,5
RI-03	Intercomunicación de componentes ineficiente o ineficaz	25% 	3	0,75	0,9
RI-04	Prestaciones insuficientes de componentes	20% 	3	0,6	1,2
RI-05	Modificación de los requerimientos del proyecto	30%	4	1,2	1,2
RI-06	Dificultad en conseguir determinados componentes	0% 	4	0	1,2
RI-07	Excesivo tiempo para cumplir los objetivos del proyecto	40% 	5	2	2,5
RI-08	Reducción de la fuerza de trabajo	10%	2 	0,2	0,3

Tabla 6-4: Riesgos mitigados en iteración 3

## 6.7 CONCLUSIONES

Como podrá observarse en la tabla anterior, la presente iteración ha sido de crucial importancia para el proyecto Hermes III. Haberlo terminado exitosamente en tiempo y forma ha reducido drásticamente muchos riesgos latentes.

A continuación se procede a explicar las razones por las cuales se han mitigado determinados riesgos:

- RI-01: El hecho de lograr la puesta en funcionamiento de los actuadores y la disponibilización de los datos generados por los sensores dentro del ecosistema de ROS en forma de tópicos a una alta frecuencia y sin ningún sobresalto en el sistema operativo, ratifican su estabilidad.
- RI-02: Dado que se ha incorporado exitosamente todo el hardware necesario para el funcionamiento del Hermes III, acorde a sus requerimientos, y ninguno de los componentes se ha averiado ni ha presentado signos de deterioro durante las pruebas, el riesgo RI-02 se ha reducido drásticamente dado que se descarta la incompatibilidad y sólo queda latente la probabilidad de avería de algún componente.
- RI-03: Se reduce levemente este riesgo por los buenos resultados obtenidos en las pruebas en cuanto a la frecuencia de publicación de los datos provenientes de los sensores y a los tiempos de reacción de los actuadores ante comandos, los cuales se consideraron aceptables, pero es meramente una suposición. Cuando se implemente el sistema de control en una posterior iteración, se podrá comprobar si efectivamente la intercomunicación es eficiente y eficaz.
- RI-04: Éste riesgo se reduce a la mitad, dado que las prestaciones de los componentes adicionados al Hermes III tienen un gran potencial y las pruebas han demostrado que las prestaciones son las esperadas según las especificaciones técnicas de cada componente.
- RI-06: La probabilidad de no conseguir determinados componentes se hace cero, dado que se han conseguido todos los componentes necesarios en cuanto a sensores, actuadores y placas de desarrollo.
- RI-07: El tiempo para la finalización del proyecto Hermes III es cada vez menos incierto en la medida que se avanzan en las iteraciones cumpliendo lo planificado. En este caso, no se ha demorado más de un mes, que es lo correspondiente a una iteración.
- RI-08: Se ha disminuido el impacto de la pérdida de fuerza de trabajo porque la finalización de la presente iteración ha significado un hito para el proyecto Hermes III, debido a su criticidad y al esfuerzo que ha implicado. El volumen de avance fue muy significativo.

# CAPÍTULO 7

## 7 ITERACIÓN 4: SLAM (SYSTEM LOCALIZATION AND MAPPING)

### 7.1 INTRODUCCIÓN

Ya contando con la capacidad de sensar profundidad de campo mediante el dispositivo Microsoft Kinect V1, resta lograr que dicha información fuese utilizada para generar un mapa tridimensional del entorno del robot y a su vez lograr posicionarlo dentro del mismo.

Existen numerosos algoritmos en el ecosistema ROS que se nutren de los datos de sensores de profundidad para generar los resultados anteriormente mencionados.

### 7.2 REQUERIMIENTOS

En la presente iteración se atenderán los siguientes requerimientos funcionales:

RF2	Debe poder auto localizarse
RF3	Debe poder realizar un mapa

Tabla 7-1: Requerimientos funcionales iteración 4

### 7.3 DESARROLLO

Para el desarrollo de esta iteración se utiliza un algoritmo desarrollado por el ingeniero Mathieu Labbé, denominado RTAB Map (*Real-Time Appearance-Based Mapping*). El mismo fue elegido por producir los mejores resultados entre los diferentes algoritmos de SLAM que se han probado (ORB-SLAM, hector\_mapping y gmapping).

ORB-SLAM particularmente es un algoritmo desarrollado por Raúl Mur-Artal, Juan D. Tardós, J. M. M. Montiel y Dorian Gálvez-López. El mismo tiene una precisión excelente con el sensor Kinect, teniendo incluso la capacidad de realizar un SLAM solamente con una cámara RGB convencional, sin contar con sensor de profundidad ni cámaras estéreo. Sin embargo, su gran desventaja con respecto al proyecto Hermes III, es que no está desarrollado como un módulo destinado a integrarse directamente al ecosistema de ROS. Debido a ésto último se ha trabajado en el código fuente del algoritmo logrando publicar tópicos ROS de odometría, pero se perdía considerablemente fluidez de ejecución con respecto al algoritmo original.

Por otro lado, se ha probado el algoritmo “hector\_mapping”, creado por una Universidad Alemana y reconocido como un paquete oficial de ROS. La desventaja del mismo es que está pensado para funcionar con sensores de profundidad profesionales de alta precisión y por ende al probarlo con el sensor Kinect, se han obtenido muy malos resultados.

Finalmente, al experimentar con el algoritmo RTAB Map, el cual es totalmente compatible con ROS, se han obtenido muy buenos resultados y por ende se ha elegido como el software que dotará al Hermes de las capacidades de SLAM.

#### 7.3.1 REAL TIME APPEARANCE BASED MAPPING

RTAB-Map consiste en un enfoque de SLAM de reconstrucciones RGB-D basado en un detector bayesiano global de cierre de bucle (*loop closure*). Este detector utiliza un método conocido

como bolsa de palabras (*Bag of Words*) para determinar la probabilidad de que una nueva imagen provenga de una ubicación anterior o posterior. Las principales ventajas de este método son su facilidad de uso y su eficiencia computacional, imprescindible cuando se procesa información de forma masiva.



## RTAB-Map: Real-Time Appearance-Based Mapping

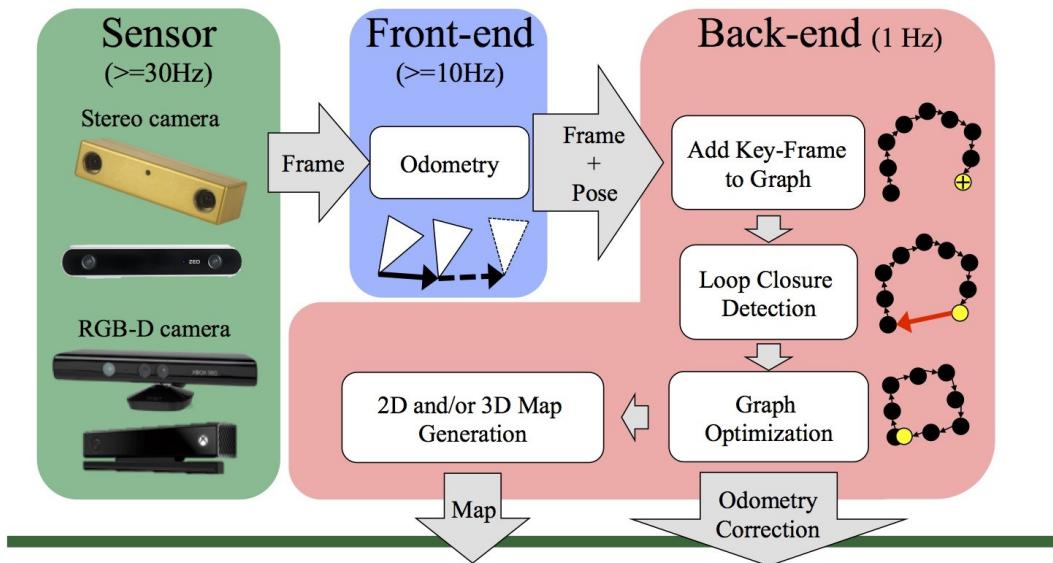


Figura 7-1: Algoritmo de RTAB-Map

Cuando se acepta una hipótesis de cierre de bucle, se añade una nueva restricción al gráfico del mapa, entonces un optimizador gráfico minimiza los errores en el mapa. Para gestionar la memoria, se limita el número de ubicaciones utilizadas para la detección de cierre de bucles y en la optimización del mapeado, de modo que las restricciones en tiempo real en entornos a gran escala siempre se respetan (Ver Figura 7-1). [30]

### 7.3.1.1 INSTALACIÓN

Para la instalación de RTAB Map sobre la NVIDIA Jetson es necesario ejecutar en consola el siguiente comando:

```
$ sudo apt-get install ros-indigo-rtabmap-ros
```

Luego, para que el mismo pueda ser utilizado, es necesario tener instalado correctamente el driver del sensor Microsoft Kinect. En éste caso se utilizará el driver “*libfreenect*”:

```
$ rosrun freenect_launch freenect.launch depth_registration:=true
```

Luego, para correr RTAB Map se ejecutará el siguiente comando:

```
$ rosrun rtabmap_ros rtabmap.launch rtabmap_args:="--delete_db_on_start"
rviz:=true rtabmapviz:=false
```

Cabe destacar que el argumento “delete\_db\_on\_start” es utilizado para arrancar el mapeado sobre escribiendo cualquier mapa que pueda existir previamente. Incluso si se quiere limpiar el mapa en tiempo de ejecución se puede llamar al siguiente servicio ROS:

```
$ rosservice call /rtabmap/reset
```

El mapa generado es guardado en `~/.ros/rtabmap.db`, y RTAB Map provee una herramienta para la visualización de los mismos el cual puede ser ejecutada mediante el comando:

```
$ rtabmap-databaseViewer ~/.ros/rtabmap.db
```

### Aclaración Importante

La localización que se calcula siempre es relativa al punto de partida del robot, es decir, al poner en funcionamiento el algoritmo RTAB Map.

En algunas ocasiones puede darse que el algoritmo no sea capaz de ubicar la posición relativa en base a las imágenes que capture dada la baja resolución del dispositivo Kinect. Por más que el robot vuelva sobre sus propios pasos para sensar zonas conocidas, no siempre el algoritmo es capaz de recuperarse y volver a detectar la posición del mismo. En esos casos será necesario reiniciar la odometría para redefinir el punto de partida, llamando al siguiente servicio ROS:

```
$ rosservice call /rtabmap/reset_odom
```

Puede obtenerse mayor información acerca de los tópicos, servicios, nodos y parámetros que ofrece el paquete RTAB-Map, ingresando a su wiki oficial [http://wiki.ros.org/rtabmap\\_ros](http://wiki.ros.org/rtabmap_ros).

### 7.3.2 CONFIGURANDO RTAB MAP EN HERMES III

La configuración de RTAB Map varía de acuerdo a la cantidad de sensores que posea el robot en cuestión. En el caso del proyecto Hermes III se manejará el esquema mostrado en la Figura 7-2.

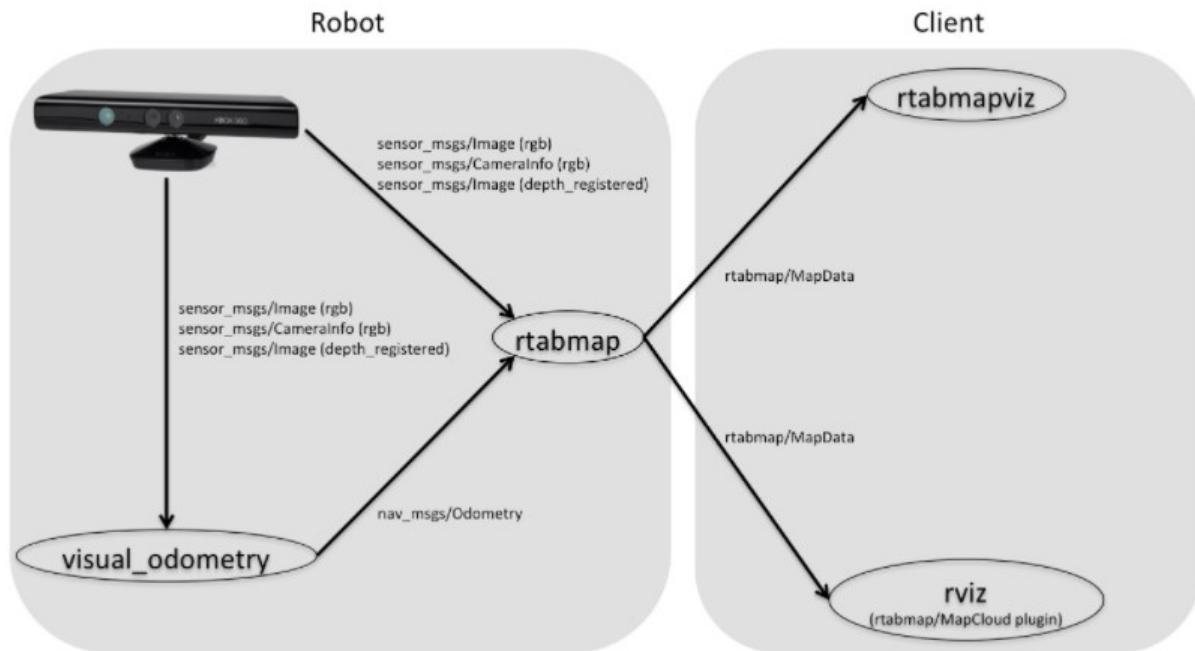


Figura 7-2: Configuración de RTAB-Map según la cantidad de sensores

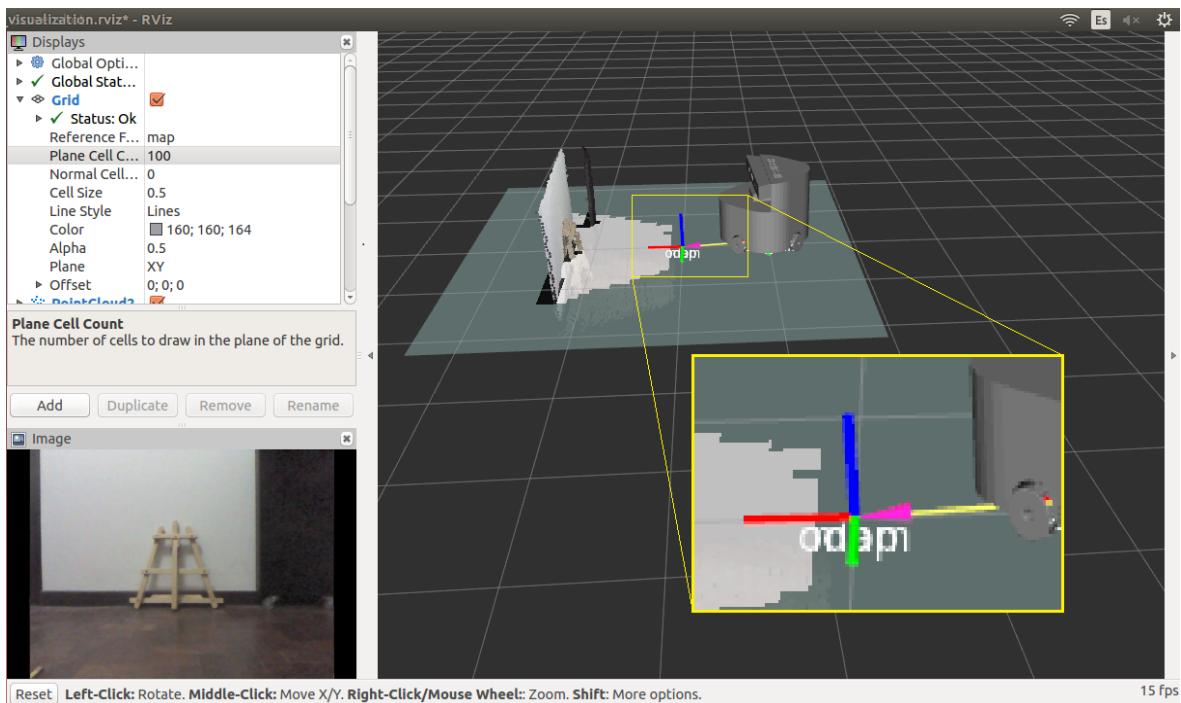
A pesar de contar con datos odométricos por el sensado de velocidad que se realiza sobre cada rueda del Hermes III, se ha decidido utilizar únicamente los datos provenientes del sensor de profundidad Kinect para proveer información al algoritmo RTAB Map, y así poder probar su rendimiento contando con los mínimos recursos posibles.

Para éste caso, la wiki oficial del algoritmo indica que deben utilizarse dos nodos del paquete RTAB Map, “*visual\_odometry*” y “*rtabmap*”. El primero será utilizado para generar datos odométricos a partir de las imágenes RGB y de profundidad capturadas y el segundo nodo se encargará de generar un mapa basándose en la información producida por el primer nodo y por la información obtenida directamente del sensor Kinect.

Existe un tercer nodo que nos brinda el paquete, llamado “*rtabmapviz*” que brinda la posibilidad de visualizar en tiempo real la posición y el mapa generado por el algoritmo.

Luego se tiene el nodo “*rviz*”, que como se ha mencionado en iteraciones anteriores, es la herramienta por excelencia en ROS para la monitorización y visualización de múltiples tópicos publicados en la red.

Entonces, para realizar las primeras pruebas de mapeado y localización, se necesitará únicamente el sensor Kinect y ejecutar en la NVIDIA Jetson el *launchfile* “C.1 rtabmap\_odometry.launch”. Una vez hecho ésto, se puede proceder a ejecutar RVIZ mediante el *launchfile* “C.3 remote\_visualization.launch”, que cabe destacar que puede ejecutarse tanto en la misma NVIDIA o en cualquier PC esclava ROS.



**Figura 7-3: Desplazamiento de Hermes III**

Puede verse en la Figura 7-3 lo mencionado anteriormente respecto al punto de referencia inicial (punto de partida), el cual se encuentra resaltado. El mismo cuenta con 3 ejes de coordenadas x, y, z para representar los 6 grados de libertad con los que cuenta el robot. [31]

### 7.3.3 AJUSTE DE PARÁMETROS DE MAPEO

De los parámetros que pueden ajustarse del algoritmo RTAB Map [32], se han definido los siguientes valores, los cuales han producido los mejores resultados en las pruebas de laboratorio, pero, dado el número limitado de pruebas, no significa que sean los óptimos.

#### 7.3.3.1 PARÁMETROS CONFIGURADOS

##### 7.3.3.1.1 ODOMETRÍA

- ❖ <param name="frame\_id" type="string" value="base\_link"/>: Se indica el nombre del tópico que publicará el nodo de odometría.
- ❖ <param name="Odom/ResetCountdown" type="string" value="1" />: La odometría se resetea automáticamente luego de perder referencia con el punto de partida.
- ❖ <param name="Odom/Strategy" value="1"/>: Se setea la estrategia de odometría en "frame to frame", con ello se logra una mejor performance de ejecución y por ende un muestreo a mayor frecuencia que evitaría posibles pérdidas de posición del robot.
- ❖ <param name="Vis/CorType" value="0"/>: Se desactiva esta característica para obtener resultados más robustos de odometría, aunque posiblemente menos *matches*. Se decidió ésto dada la baja resolución del sensor Kinect que necesita requerimientos de comparación de *frames* más estrictos para no dar tantos falsos positivos.
- ❖ <param name="GFTT/MinDistance" type="string" value="5"/>: Este valor se define acorde a la resolución de las imágenes que capture el sensor, Kinect en nuestro caso. Para bajas resoluciones, se recomienda valores bajos.

### 7.3.3.1.2 MAPEO

---

- ❖ <param name="frame\_id" type="string" value="base\_link"/>: Nombre del tópico de odometría al que estará suscripto el nodo rtabmap.
- ❖ <param name="subscribe\_scan" type="string" value="true"/>: El tópico "scan" debe estar configurado.
- ❖ <param name="subscribe\_depth" type="bool" value="true"/>: Los tópicos "depth/image" y "rgb/camera\_info" deben estar configurado.
- ❖ <param name="Reg/Force3DoF" value="true" />: Se indica al algoritmo que el robot se moverá sobre una superficie plana, por ende sólo hace falta el cálculo de su posición considerando solamente 3 grados de libertad.
- ❖ <param name="queue\_size" type="int" value="10"/>: Cantidad de tópicos de entrada que puede acumular antes de leerlos.
- ❖ <param name="RGBD/AngularUpdate" type="string" value="0.01"/>: Este valor numérico representa cuánto deberá girar el robot para realizar un nuevo muestreo de profundidad para realizar el mapa.
- ❖ <param name="RGBD/LinearUpdate" type="string" value="0.01"/>: Este valor numérico representa cuánto deberá avanzar linealmente el robot para realizar un nuevo muestreo de profundidad para realizar el mapa.
- ❖ <param name="Rtabmap/TimeThr" type="string" value="500"/>: Con el fin de hacer un uso eficiente de la memoria, aquellos datos del mapa que permanezcan inactivos por el tiempo definido en éste parámetro, serán transferidos a la memoria secundaria.
- ❖ <param name="Mem/RehearsalSimilarity" type="string" value="0.45"/>: En el caso que se obtengan dos imágenes con una similitud por encima de éste *threshold*, entonces ambos nodos serán fusionados produciendo un nodo de mayor peso. Ésto hará que sea más eficaz el proceso de intercambio de nodos entre la memoria principal y la memoria secundaria.
- ❖ <param name="RGBD/OptimizeFromGraphEnd" type="string" value="true"/>: Se activa la opción de que la posición del robot se calcule en base al último dato de mapa generado (ésto trae beneficios de rendimiento dado que los últimos datos del mapa se encuentran en memoria principal).
- ❖ <param name="Rtabmap/StartNewMapOnLoopClosure" value="true"/>: Cuando se llama al servicio de reseteo de odometría, se esperará a un "*loop closure*" para comenzar un nuevo mapa.
- ❖ <param name="Optimizer/Slam2D" value="true" />: Al trabajar solo con 3DoF, se optimiza la graficación 2D (valor X, Y y Yaw)
- ❖ <param name="cloud\_noise\_filtering\_radius" value="0.05"/>: Parámetro recomendado por la documentación oficial del algoritmo RTAB.
- ❖ <param name="cloud\_noise\_filtering\_min\_neighbors" value="2"/>: Parámetro recomendado por la documentación oficial del algoritmo RTAB.

#### 7.3.4 PRECISIÓN DEL ALGORITMO SLAM

Habiendo ajustado los parámetros, se procede a realizar pruebas que demuestran la precisión en cuanto al desplazamiento del robot que es capaz de detectar el algoritmo RTAB Map y cómo ésta varía acorde a los datos que recibe del sensor de profundidad Kinect.

Antes de realizar las pruebas, se midió en laboratorio la distancia mínima que es capaz de detectar el dispositivo Microsoft Kinect, colocando frente al mismo una pizarra donde se ha anotado la distancia a la que se encontraba del sensor (Ver Figura 7-4).

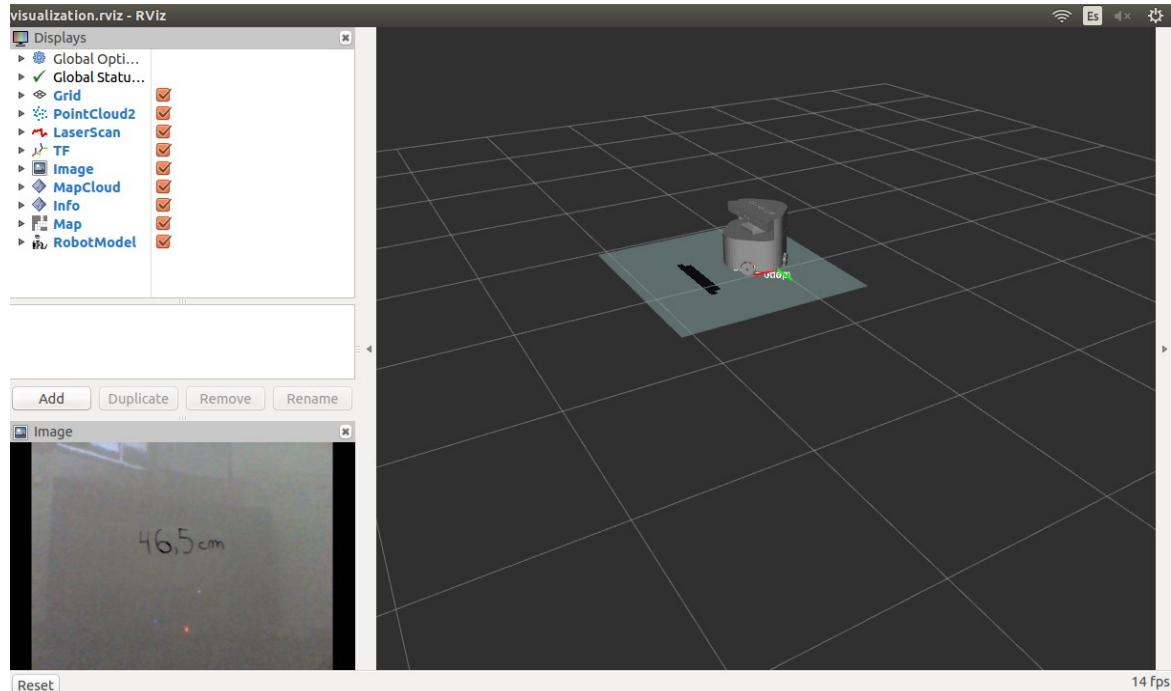


Figura 7-4: Distancia mínima 46.5 centímetros

Si un objeto se ubica frente al sensor Kinect a una distancia menor a 46.5 centímetros, no será detectado de ninguna manera por el robot.

#### 7.3.4.1 INFORMACIÓN DE PROFUNDIDAD SIMPLE

Para el primer caso de prueba se ubicará el sensor Kinect frente a una pared de color homogéneo y se alejará el robot medio metro de manera perpendicular a la misma.

Cabe aclarar que cada cuadrícula representa un área de 0.5 metros por 0.5 metros.

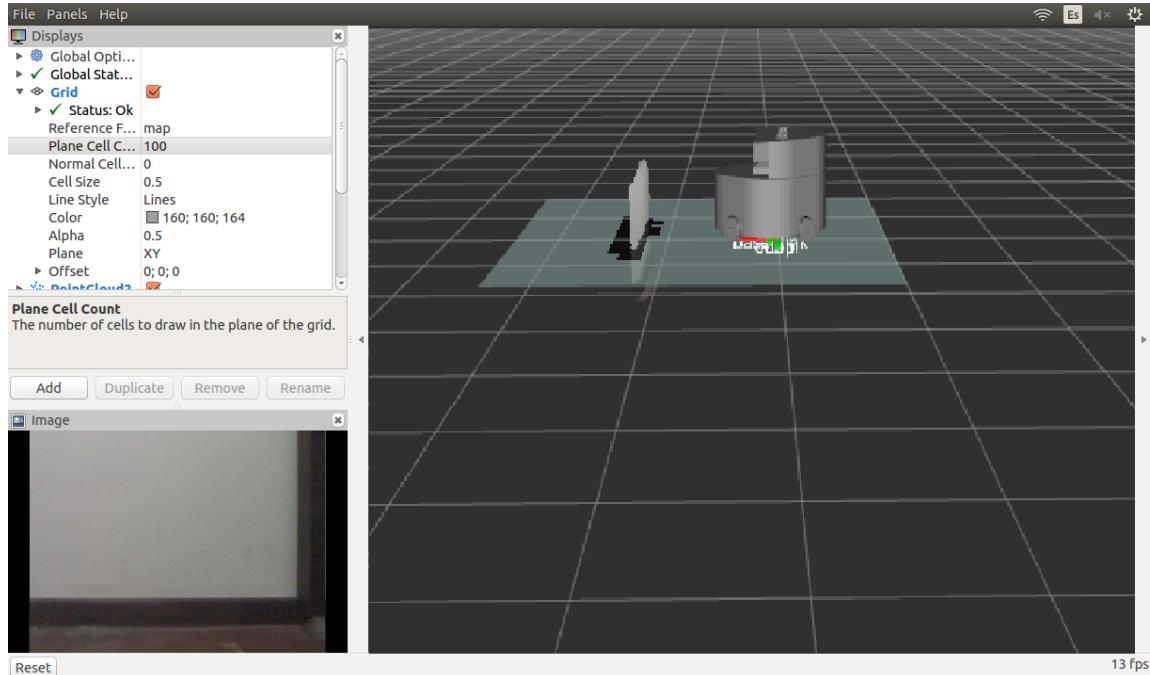


Figura 7-5: Posición inicial sin referencia (vista superior)

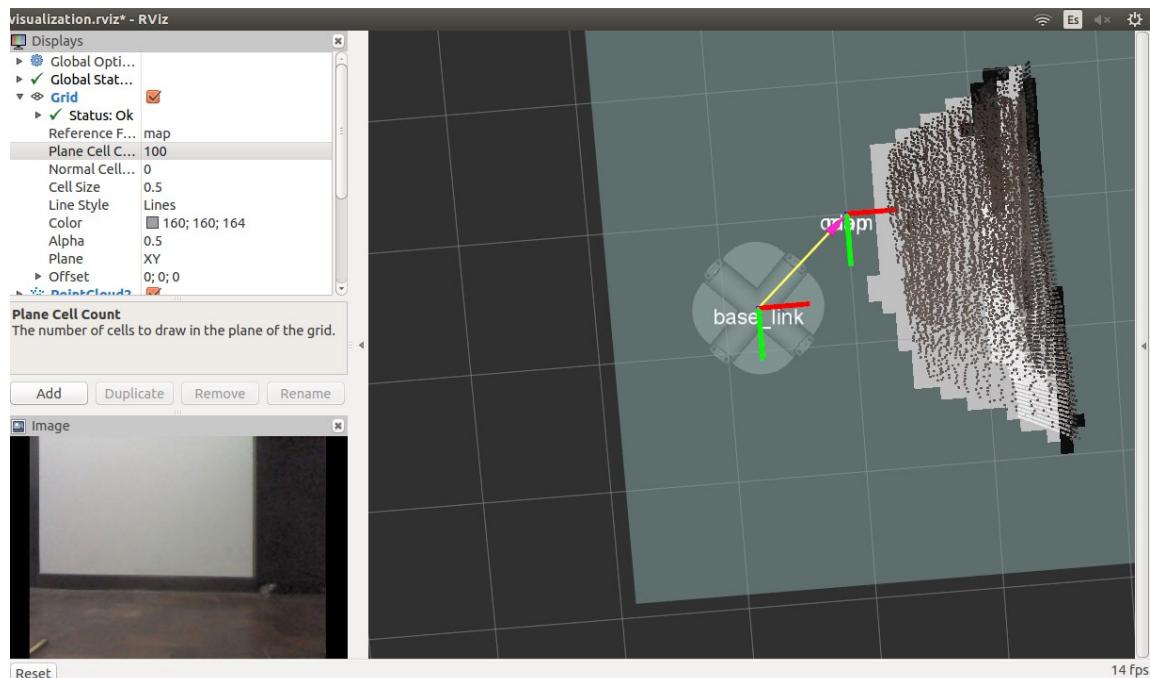
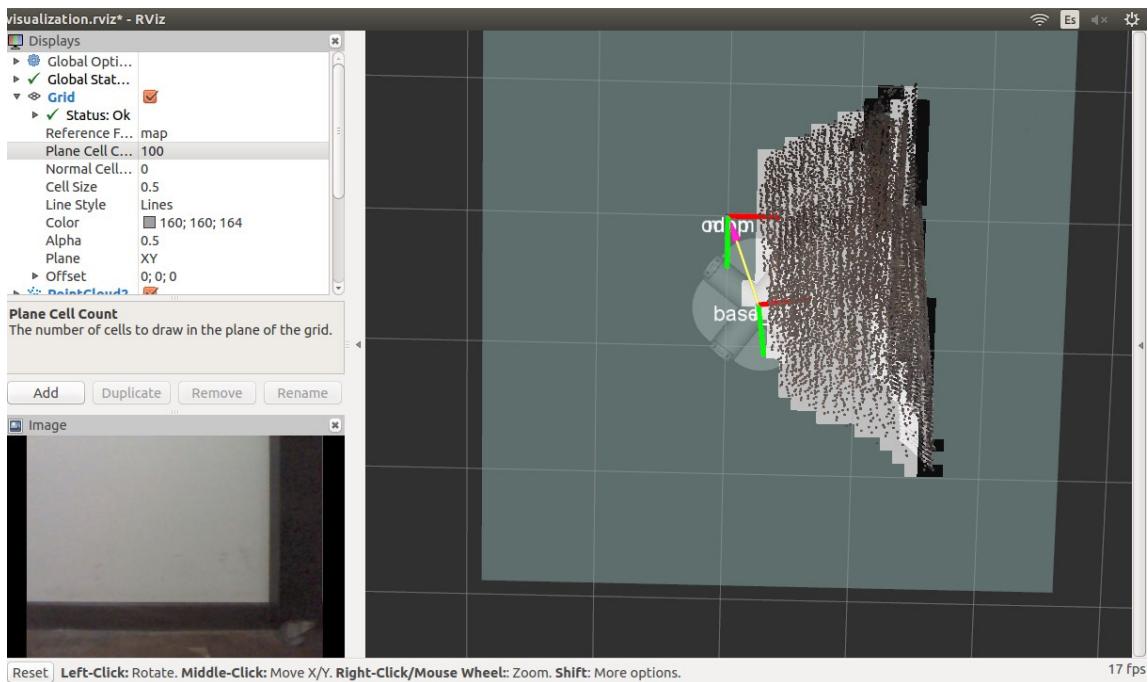


Figura 7-6: Desplazamiento de medio metro sin referencia (vista inferior)



**Figura 7-7: Regreso al punto de partida sin referencia (vista inferior)**

En la Figura 7-5, Figura 7-6 y Figura 7-7 puede apreciarse claramente como a pesar de mover al robot perfectamente en línea recta perpendicular a la pared, el algoritmo posee un gran error acumulativo que al regresar al punto de partida es de unos 30 centímetros.

#### 7.3.4.2 INFORMACIÓN DE PROFUNDIDAD COMPLEJA

Para demostrar la dependencia de la exactitud del algoritmo SLAM con el nivel de detalle que se obtiene de las imágenes tanto RGB como de profundidad del sensor Kinect, se ha realizado una segunda prueba, colocando un objeto de una estructura tridimensional muy particular sobre la pared y se procedió a realizar exactamente el mismo movimiento que en la prueba anterior (Ver Figura 7-8, Figura 7-9 y Figura 7-10).

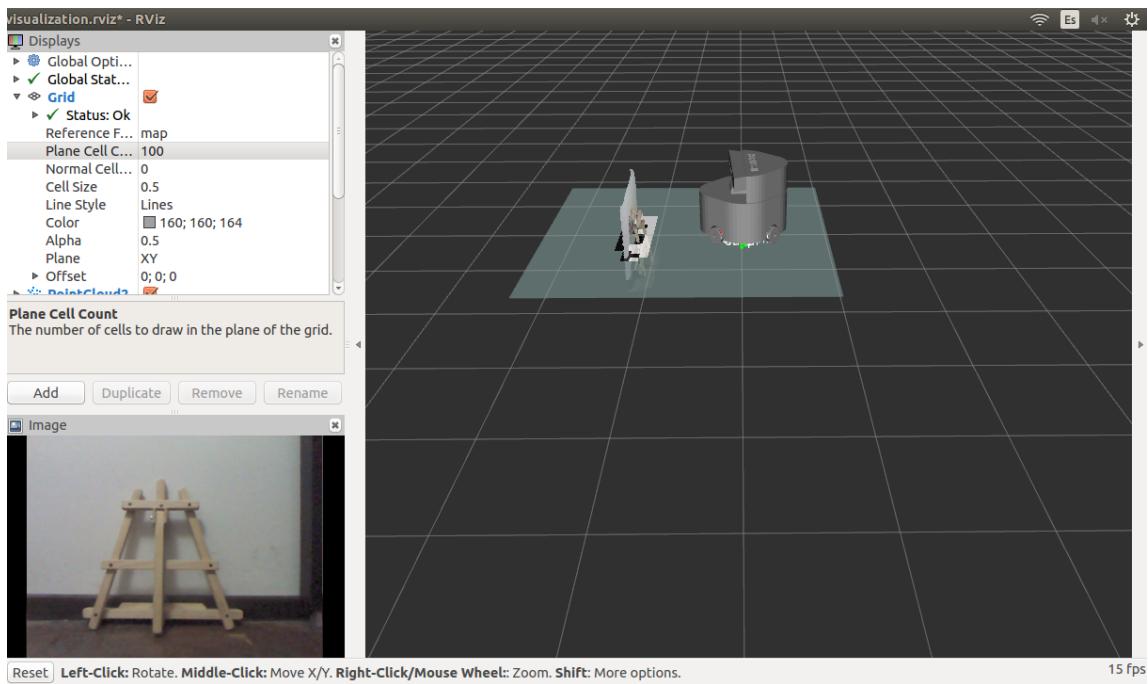


Figura 7-8: Posición inicial con referencia (vista superior)

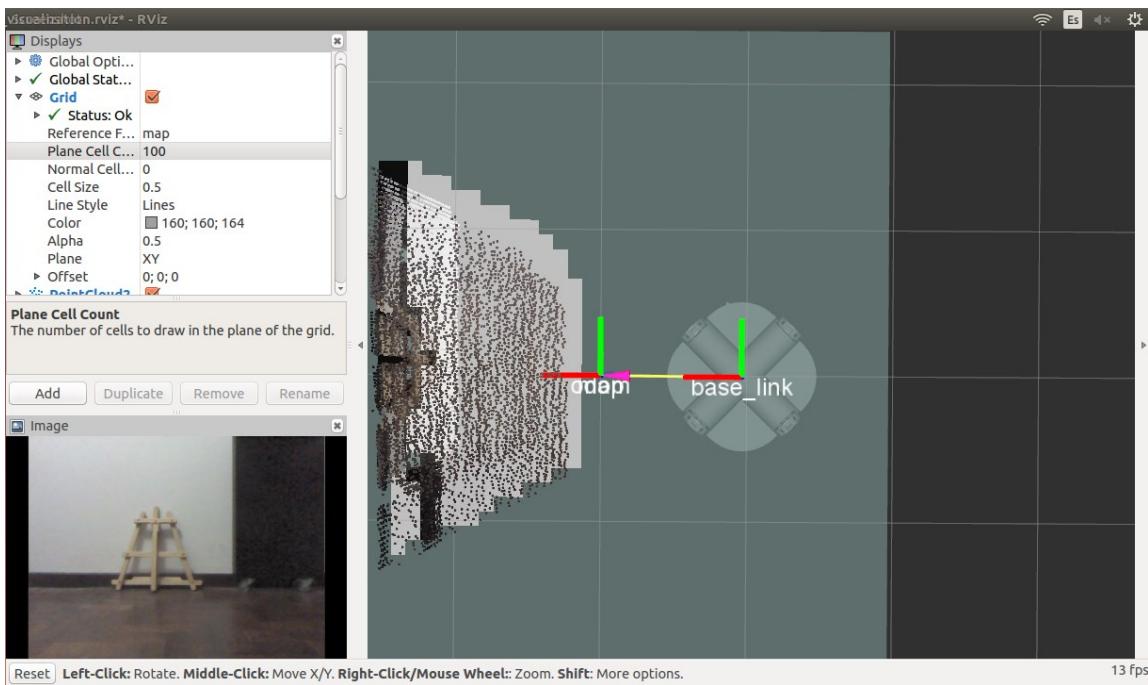


Figura 7-9: Desplazamiento de medio metro con referencia (vista inferior)

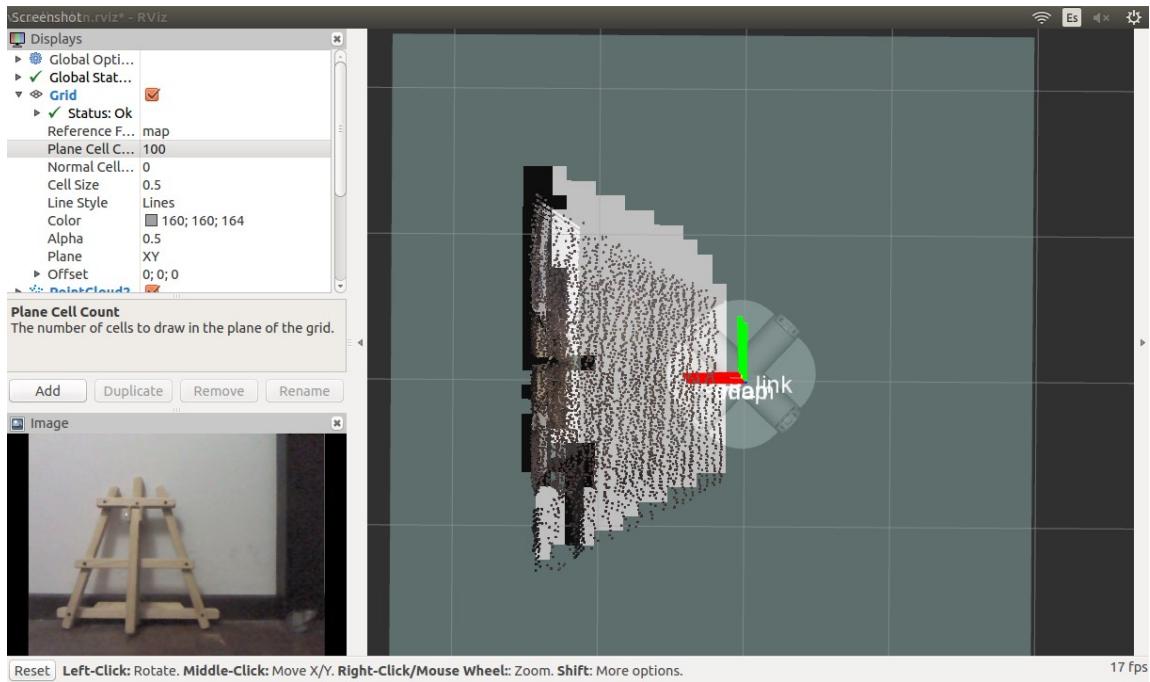


Figura 7-10: Regreso al punto de partida con referencia (vista inferior)

Es realmente sorprendente la mejora de la exactitud en las mediciones del algoritmo cuando lo hace en base a datos de profundidad heterogéneos. Ésto permite concluir que el algoritmo es más exacto cuando posee un objeto delante del sensor de profundidad el cual puede usarse como punto de referencia inequívoco y realizar en base al mismo mejores cálculos de distancia que cuando posee únicamente una pared homogénea delante.

## 7.4 PRUEBAS

Caso de prueba PRF2-2	
Objetivo	Mover el robot una distancia de 0.5 metros en línea recta en cualquier dirección, marcando en el suelo el punto de partida y de llegada que refleje dicha distancia.
Prerrequisitos	<ul style="list-style-type: none"> <li>Se deben tener los drivers del sensor Kinect correctamente instalados en la NVIDIA Jetson.</li> <li>Se debe tener el sensor Microsoft Kinect conectado a la NVIDIA Jetson.</li> <li>Se debe contar con el paquete RTAB Map instalado.</li> <li>Se debe contar con los launch files del anexo parte C.</li> </ul>
Pasos	<ol style="list-style-type: none"> <li>Correr el launchfile “slam_rtabmap.launch”</li> <li>Correr el visualizador RViz, mediante el launch file “remote_visualization.launch”</li> <li>Mover el sensor de profundidad una distancia específica con respecto a un objeto que tenga delante a una distancia superior a la mínima posible de 46.5 centímetros.</li> </ol>
Resultado esperado	Se debe visualizar en el software de monitorización del robot una relación directamente proporcional entre el desplazamiento real y el virtual.
Resultados obtenidos	Pueden apreciarse los resultados obtenidos en la sección “Precisión del algoritmo SLAM” del presente capítulo.
Códigos de prueba necesarios	Anexo C.1 / C.2 / C.3

Tabla 7-2: Prueba nro. 1 iteración 4

Caso de prueba PRF2-3	
Objetivo	Girar el robot 180º sobre su propio eje vertical.
Prerrequisitos	<ul style="list-style-type: none"> <li>Se deben tener los drivers del sensor Kinect correctamente instalados en la NVIDIA Jetson.</li> <li>Se debe tener el sensor Microsoft Kinect conectado a la NVIDIA Jetson.</li> <li>Se debe contar con el paquete RTAB Map instalado.</li> <li>Se debe contar con los launch files del anexo parte C.</li> </ul>
Pasos	<ol style="list-style-type: none"> <li>Correr el launchfile “slam_rtabmap.launch”.</li> <li>Correr el visualizador RViz, mediante el launch file “remote_visualization.launch”.</li> <li>Girar lentamente el sensor de profundidad sobre su propio eje vertical unos 180 grados.</li> </ol>

Resultado esperado	Se debe reflejar el movimiento de giro del robot de 180° en el software de monitorización.
Resultados obtenidos	Durante las pruebas de la sección “Precisión del algoritmo SLAM” se ha podido concluir lo mismo que para los movimientos traslacionales, es decir, que la precisión depende de la heterogeneidad del entorno.
Códigos de prueba necesarios	Anexo C.1 / C.2 / C.3

Tabla 7-3: Prueba nro. 2 Iteración 4

Caso de prueba PRF3	
Objetivo	Poner en funcionamiento el robot en un ambiente cerrado y que recorra la mayor superficie posible del lugar.
Prerrequisitos	<ul style="list-style-type: none"> <li>• Se deben tener los drivers del sensor Kinect correctamente instalados en la NVIDIA Jetson.</li> <li>• Se debe tener el sensor Microsoft Kinect conectado a la NVIDIA Jetson.</li> <li>• Se debe contar con el paquete RTAB Map instalado.</li> <li>• Se debe contar con los launch files del anexo parte C.</li> </ul>
Pasos	<ol style="list-style-type: none"> <li>1. Correr el launchfile “slam_rtabmap.launch”</li> <li>2. Correr el visualizador RViz, mediante el launch file “remote_visualization.launch”</li> <li>3. Mover el sensor de profundidad lentamente recorriendo toda la habitación donde se encuentre, considerando que no debe acercarse a los obstáculos a una distancia menor de 46-5 centímetros.</li> </ol>
Resultado esperado	Debe obtenerse un mapa que refleje las proporciones reales del lugar.
Resultados obtenidos	Se ha logrado realizar un mapa muy preciso del entorno físico que rodea al robot siempre y cuando el algoritmo no pierda la relación entre la imagen actual capturada y una anterior. Ésto sucede con movimientos bruscos, en zonas muy abiertas o con objetos muy cercanos al sensor. Pueden verse el mapa generado del Laboratorio de Arquitectura de computadoras en la siguiente sección “Resultados”
Códigos de prueba necesarios	Anexo C.1 / C.2 / C.3

Tabla 7-4: Prueba nro. 3 iteración 4

## 7.5 RESULTADOS

Los resultados obtenidos han sido realmente buenos, sobretodo luego de ajustar parámetros para la mejora del SLAM. El robot fue capaz de localizarse y a su vez construir un mapa preciso del Laboratorio de Arquitectura de Computadoras, como se adjunta en la siguiente imagen (Figura 7-11).

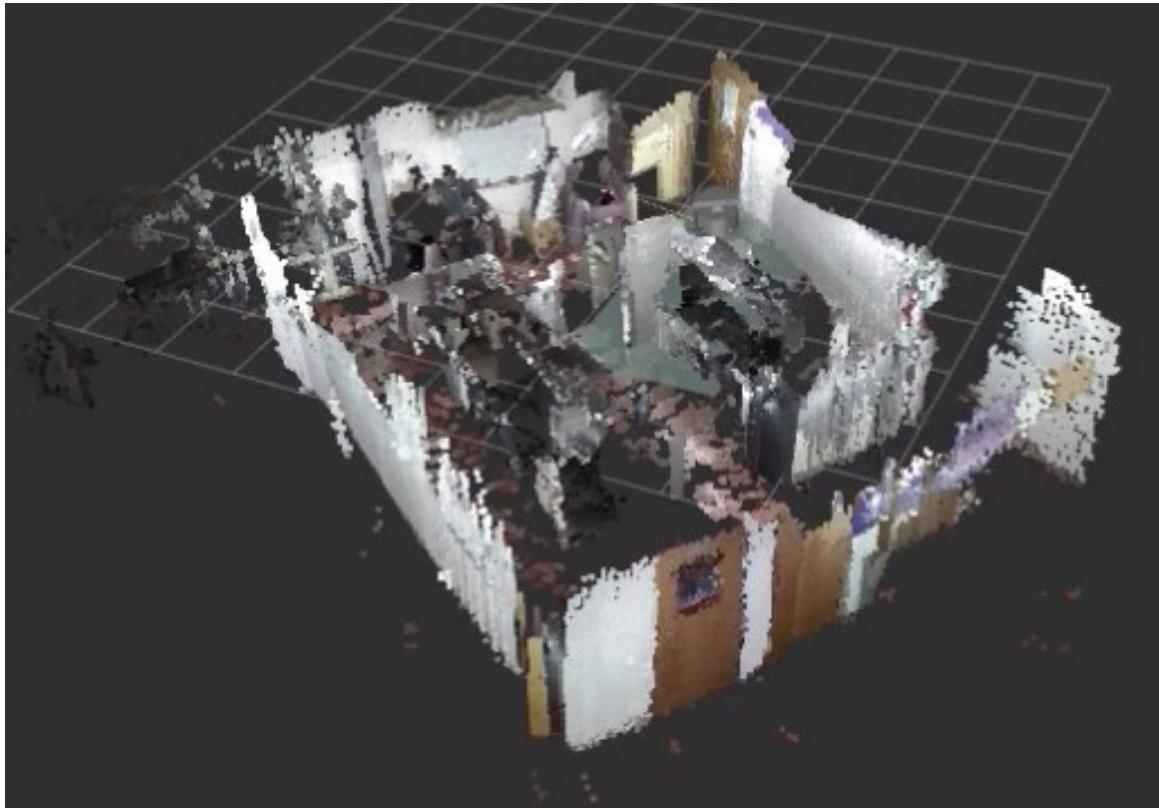


Figura 7-11: Mapa 3D del Laboratorio de Arquitectura de Computadoras

## 7.6 RIESGOS SUPERADOS

El análisis de mitigación de riesgos al finalizar la presente iteración es el siguiente:

ID	Riesgo	Probabilidad	Impacto	Exposición Final Iteración	Exposición Inicio Iteración
RI-01	Sistema Operativo inestable	4%	2	0,08	0,12
RI-02	Incompatibilidad o avería de componentes	10%	2	0,2	0,2
RI-03	Intercomunicación de componentes ineficiente o ineficaz	25%	3	0,75	0,75
RI-04	Prestaciones insuficientes de componentes	15%	3	0,45	0,6
RI-05	Modificación de los requerimientos del proyecto	20%	4	0,8	1,2

RI-06	Dificultad en conseguir determinados componentes	0%	4	0	0,2
RI-07	Excesivo tiempo para cumplir los objetivos del proyecto	30% 	5	1,5	2
RI-08	Reducción de la fuerza de trabajo	10%	2	0,2	0,2

Tabla 7-5: Riesgos mitigados en iteración 4

## 7.7 CONCLUSIONES

El robot fue capaz de reconocer su posición en tiempo real dentro del entorno donde se encontraba. Esto responde exitosamente a una de las preguntas fundamentales para que un robot fuese totalmente autónomo, la cual es “¿dónde estoy?”. En cuanto al mapeo, se han obtenido resultados satisfactorios, pero con cierta dificultad en realizarlos, dado que requiere de movimientos suaves y manteniéndose a una distancia mayor a la mínima posible para no perder referencias y detener el proceso de mapeo. Se ha logrado mejorar la estabilidad con la modificación de los parámetros, por lo que esto hace a RTAB-Map un algoritmo de SLAM flexible y totalmente modular ya que se ha acoplado al robot sin inconvenientes, mediante el sistema de comunicación por tópicos que ofrece ROS.

En cuanto al análisis de mitigación de riesgos se ha tenido en cuenta lo siguiente:

- RI-01: El algoritmo RTAB Map fue desarrollado para integrarse con el framework ROS y su utilización no ha traído mayores inconvenientes. Su puesta en funcionamiento con el sensor Kinect a sido muy rápida.
- RI-04: Dado a las pruebas realizadas en la presente iteración, se concluye que el sensor de profundidad Xbox Kinect cumple con las expectativas tanto en velocidad de frames por segundos como así también en resolución de imagen, ya que ha permitido obtener resultados muy satisfactorios.
- RI-05: A medida que se avanza en el proyecto, y se van cumpliendo los requerimientos, la probabilidad de que los requerimientos sean modificados disminuye.
- RI-07: Este riesgo disminuirá al final de cada iteración siempre que los tiempos estipulados para cumplir con los objetivos del loop se cumplan. En este caso se ha demorado lo previsto.

# CAPÍTULO 8

## 8 ITERACIÓN 5: SISTEMA DE CONTROL

### 8.1 INTRODUCCIÓN

En la presente iteración se implementará el sistema de control, el cual será utilizado para que el robot pueda desplazarse con el menor error de trayectoria posible, respondiendo correctamente a los comandos de control tanto en velocidad como en dirección, ajustando la intensidad de movimiento de cada rueda.

### 8.2 REQUERIMIENTOS

En la presente iteración se atenderán los siguientes requerimientos funcionales:

RF1	Debe tener un sistema de locomoción omnidireccional
RF4	Debe tener un sistema de control de movimiento eficaz
RNF5	Debería tener tiempos de respuesta aceptables para un buen funcionamiento del sistema de control

Tabla 8-1: Requerimientos funcionales iteración 5

### 8.3 DESARROLLO

En la Nvidia Jetson TK1, se tiene el framework de ROS instalado, el cual ya posee integrado nodos de control PID listos para ser configurados y utilizados. Ésto permitirá controlar las velocidades de las ruedas totalmente por software.

Se implementará un sistema de control PID por cada rueda del robot. Todas ellas serán independientes entre sí, lo cual simplificará la tarea de desarrollo sin sacrificar rendimiento.

#### 8.3.1 PID ROS

El paquete PID incluido en ROS, será utilizado para controlar la velocidad de cada rueda, interactuando con el evento que controla mediante los siguientes tópicos:

- Setpoint: El sistema de control de suscribe a éste tópico que lo publicará el nodo de ROS encargado de traducir los comandos de movimiento del Hermes III en velocidades traslacionales específicas para cada rueda, para que en su conjunto logren el movimiento deseado del robot.
- State: El sistema de control de suscribe a éste tópico que lo publicará la placa de desarrollo Arduino, y consiste en la velocidad traslacional en m/s a la que actualmente va una rueda.
- Control\_effort: Éste tópico será publicado por el sistema de control para indicar la magnitud del aumento o decremento del PWM que se aplica a la rueda en cuestión, para lograr que alcance la velocidad deseada. Los valores máximo y mínimo se configuran al lanzar cada nodo del sistema de control.

Una clara ventaja de utilizar este package, es que nos abstrae de la matemática y algoritmos del PID, proporcionando la capacidad de configurar los parámetros “proporcional”, “integral” y “derivativo”, entre otras opciones de ejecución detalladas claramente en la documentación del paquete ROS.

---

### 8.3.1.1 CONFIGURANDO HERMES

Ahora veamos cómo utilizamos este package mediante el launch file incluido en el anexo D.2 pid.launch, que se ejecuta en el Hermes III. Podemos ver que, se corren 4 nodos del package PID, es decir que vamos a tener 4 sistemas de control, uno por cada rueda.

Los valores configurados para cada uno de ellos son [33]:

- node\_name: Este parámetro le asigna un nombre al nodo PID indicado. Al tener 4 nodos iguales, se le coloca diferentes nombres para poder diferenciarlos.
- pid\_enable: Nombre del tópico en el cual enviando un booleano true o false, se activa o desactiva el nodo.
- Kp: Valor del coeficiente proporcional.
- Ki: Calor del coeficiente integral.
- Kd: Valor del coeficiente derivativo.
- upper\_limit: Valor límite máximo del control effort.
- lower\_limit: Valor límite mínimo del control effort.
- windup\_limit: Valor límite máximo para el error integral.
- max\_loop\_frecuency: Máxima frecuencia esperada a la cual el nodo recibe los mensajes de estado y el loop de control corre para calcular el control effort.
- min\_loop\_frecuency: Mínima frecuencia esperada a la cual el nodo recibe los mensajes de estado y el loop de control corre para calcular el control effort.

Además, se puede ver en el launchfile de nuestro pid, que se corren los nodos send\_velocity, PID\_general, y fix\_integralError. A continuación, explicaremos qué es lo que hace cada uno de estos nodos.

---

#### 8.3.1.1.1 NODO SEND\_VELOCITY

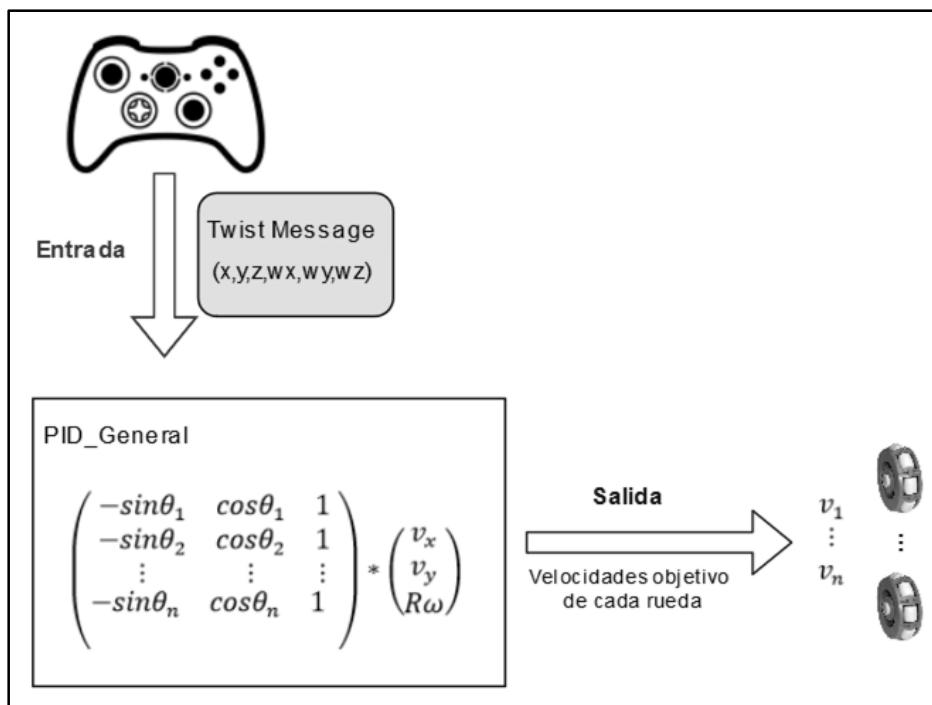
La funcionalidad de este nodo, el cual es incluido en el anexo D.3 send\_velocity.py, es comunicarle a la placa de desarrollo Arduino, que debe publicar las velocidades de sus ruedas. Este nodo es un simple “publisher”, en donde se le configura un tiempo de intervalo de publicación, en el que se le enviará un mensaje a la Arduino, y esta misma, entrará a una función especial que publicará las velocidades actuales de las ruedas al tópico state del pid correspondiente para esa rueda.

La implementación de éste nodo es la forma de manejar la frecuencia de publicación de las velocidades de las ruedas por parte de la Arduino mediante el tópico “state”. Se puede ver un diagrama de flujo en la sección anterior 2. Procesamiento, transformación y disponibilización de los datos.

---

#### 8.3.1.1.2 NODO PID\_GENERAL

Este nodo es el coordinador de los 4 controladores PID independientes que se tienen. En el anexo D.4 PID\_general.py se puede apreciar su código fuente. Este nodo se suscribe al tópico “cmd\_vel” que proviene del mando de control que se utilice (Joystick, Teclado o Smartphone). Al publicarse un mensaje en dicho tópico, el nodo PID\_general obtendrá las velocidades Euclidianas deseadas, y a través del cálculo de la matriz obtenida en el Análisis físico que se explica en el marco teórico del sistema de control, pueden obtenerse las velocidades individuales de cada rueda, enviando las mismas al nodo “fix\_integralError” el cual evaluará si se debe publicar o no dicha velocidad (Ver Figura 8-1).



**Figura 8-1: Cálculo de velocidades de cada rueda**

Un tópico “cmd\_vel” será del tipo “Twist”, es decir, que contará con una componente de velocidades lineales y otra componente de velocidades angulares, sobre los ejes x, y, z.

```
geometry_msgs/Vector3 linear
  float64 x
  float64 y
  float64 z
geometry_msgs/Vector3 angular
  float64 x
  float64 y
  float64 z
```

#### 8.3.1.1.3 NODO FIX\_INTEGRALERROR

Este nodo surge como parte de la solución al “error de inmovilización” que se detalla más adelante.

Como se puede apreciar en el código incluído en el anexo D.5 fix\_integralError.py, lo que este nodo hace es escuchar los tópicos control\_effort del PID de cada rueda, y si en una de ellas se reciben 3 valores “control\_effort” consecutivos con una diferencia menor a 0.001 entre sí y además dichos valores son menores a 1 (ajuste muy pequeño), entonces se logra desactivar el PID de dicha rueda cuando la misma se encuentra detenida. El PID se volverá a activar cuando el “set\_point” de alguna rueda es mayor al mínimo de velocidad que se es capaz de medir con los encoders, según se ha explicado en la sección Determinación del prescaler. Ésta acción se encuentra programada en el

código del anexo D.4 PID\_general.py luego de calcular las velocidades individuales deseadas de cada rueda.

### 8.3.1.2 ERROR DE INMOVILIZACIÓN

Durante las pruebas de laboratorio al implementar el sistema de control, se detectó que, al intentar detener el robot, éste lograba detenerse, pero lo hacía de manera intermitente en el tiempo. Es decir, al seguir activo los sistemas de control de las ruedas, aún cuando no se le enviaran comandos de movimiento al robot, se seguían realizando pequeños ajustes en el PWM aplicado a los motores que a la larga producían movimientos que rápidamente eran ajustados para volver a detener los motores.

Estudiando el caso detenidamente, se concluyó lo siguiente:

- El valor “control\_effort” publicado por el sistema de control nunca es un cero exacto, debido al factor integral.
- Existen velocidades muy pequeñas imposibles de alcanzar por el robot Hermes III dadas las características mecánicas de sus motores. Por ende, si el comando de velocidad publica valores deseados muy pequeños, el sistema de control entrará en una inestabilidad de aceleración y desaceleración constantes que producirán en el robot movimientos erráticos.

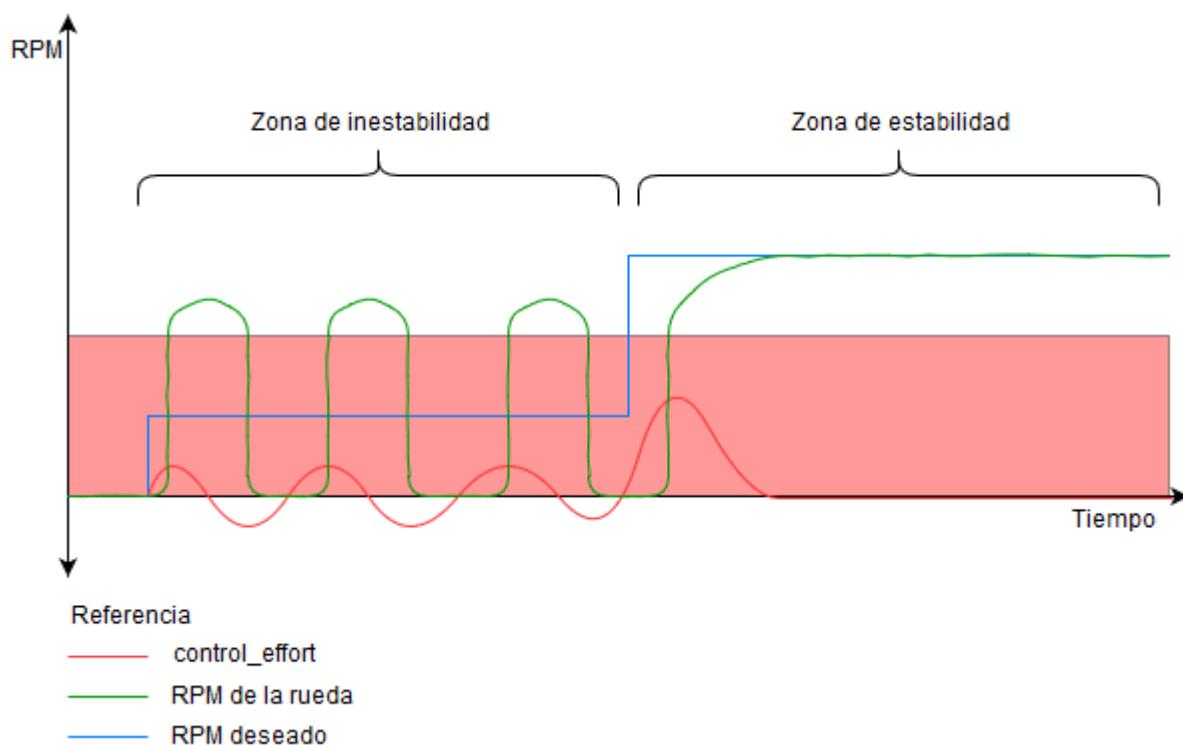


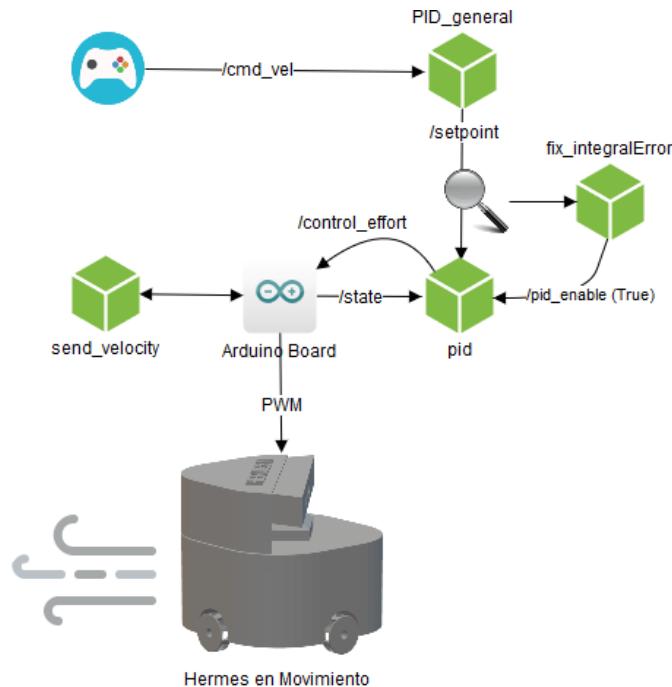
Figura 8-2: Inestabilidad a baja velocidad

El área de color rojo en la Figura 8-2, simboliza el rango de velocidades en la cual le es imposible funcionar a las ruedas del Hermes III. Lo que sucede es que el sistema de control acelerará un motor hasta romper su estado estacionario, pero una vez que ésto se logra, se excede la velocidad deseada y al desacelerarlo pierde el torque mínimo para mover al robot y por ende se vuelve a detener. Ésto sucede en la “zona de inestabilidad”.

Para solucionar el problema anterior se ha incorporado, por medio de los nodos “PID\_general” y “fix\_integralError” la capacidad de activación y desactivación del sistema de control cuando el robot se encuentra detenido.

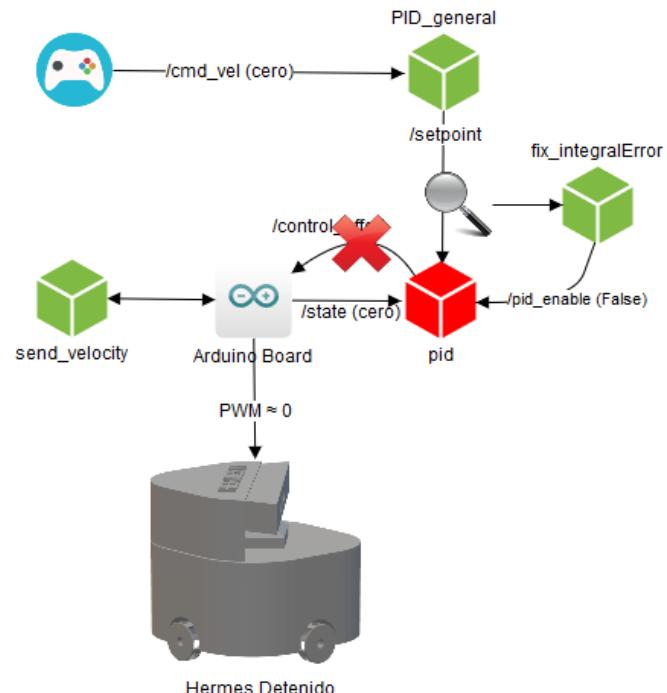
Para entender mejor la función de cada uno de los nodos mencionados anteriormente se realiza un esquema de interacción de nodos y tópicos (Figura 8-3 y Figura 8-4). Se considera el flujo de control de una sola rueda del robot Hermes para mayor simplicidad.

## 1) Rueda en movimiento



**Figura 8-3: Hermes III en movimiento**

## 2) Rueda detenida



**Figura 8-4: Hermes III detenido**

El nodo “PID\_general” volverá a activar al mismo tiempo todos los nodos “pid” en cuanto deba enviar un tema “/setpoint” que supere los márgenes mínimos de movimiento predefinidos del Hermes.

### 8.3.1.3 MEDICIÓN DE VELOCIDAD DE LAS RUEDAS DEFICIENTE

Uno de los mayores inconvenientes de la técnica de medición de velocidad de las ruedas, es que para detectar que el motor se encuentra totalmente detenido deben transcurrir 2 *overflows* consecutivos del contador del *encoder* para inferir que la rueda se encuentra detenida. Es decir, no se tiene una forma instantánea de detectar que una rueda se ha detenido.

Por esta razón, se ha decidido modificar el código que corre en la placa de desarrollo Arduino, haciendo que se ignore el dato recibido de velocidad de una rueda cuando el PWM de la misma es igual o menor al 4% (en código Arduino equivale al valor 10, de un máximo de 255 para definir el PWM) y se notifique directamente a la NVIDIA Jetson una velocidad de 0 para dichas ruedas.

### 8.3.1.4 PRECISIÓN DEL SISTEMA DE CONTROL

Para probar la calidad del control de trayectoria con el sistema de control implementado, se ha grabado en video a cámara fija, el desplazamiento en línea recta del robot, como puede apreciarse en la siguiente Figura 8-5.

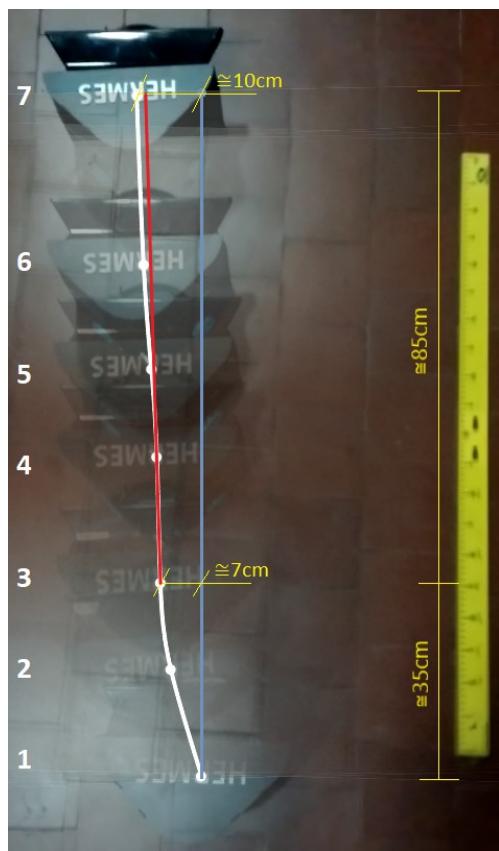
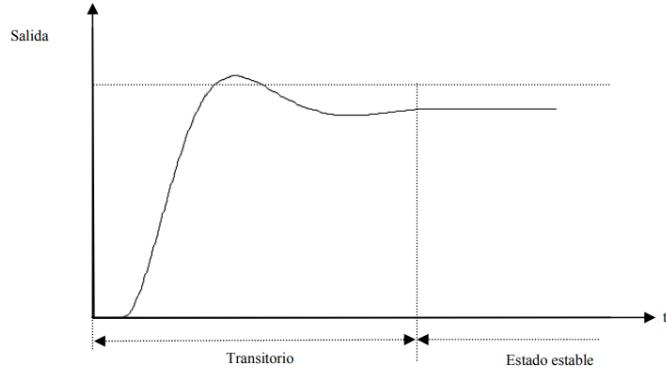


Figura 8-5: Prueba de trayectoria en línea recta del robot Hermes III

En la Figura 8-5 puede observarse un desvío mayormente pronunciado al inicio del movimiento del robot, debido a que en los primeros 35 centímetros, el sistema de control de cada rueda se encuentra en su estado transitorio (ver Figura 8-6). Una vez que se alcanza el estado estable, el robot continúa en línea recta, como puede verse a partir del tercer punto de la Figura 8-5.



**Figura 8-6: Estado transitorio y estado estable [34]**

Si se consideran ambos estados, transitorio y estable, puede decirse que el error en línea recta fue de  $10 \pm 1\text{cm}$  por metro recorrido. Si se descarta el estado transitorio (midiendo a partir del tercer punto), y extrapolando la trayectoria en línea recta en base a los puntos 3 y 4 de la Figura 8-5, se traza una línea roja ideal que dista unos  $2 \pm 1\text{cm}$  de la posición final del robot.

Lo anterior puede considerarse una prueba de integración del funcionamiento de todos los componentes del Hermes III, excepto el sensor Kinect V1, que no tiene influencia en los resultados de la presente prueba.

**Se ha logrado comprobar el correcto funcionamiento de los sistemas embebidos involucrados, Arduino Mega 2560, ATmega128 AVR Minimum Development Board y NVIDIA Jetson Tegra K1, demostrando una comunicación eficaz entre sí.**

## 8.4 PRUEBAS

Caso de prueba PRF1	
Objetivo	Enviar un comando de movimiento en una determinada dirección.
Prerrequisitos	<ul style="list-style-type: none"> <li>Tener el archivo "pid.launch" de modo que Hermes III pueda activar el funcionamiento de sus motores.</li> <li>Tener el script "arduino_ros.sh" para que se establezca comunicación entre Arduino y Nvidia Jetson</li> </ul>
Pasos	<ol style="list-style-type: none"> <li>Correr el launchfile pid.launch</li> <li>Correr el script arduino_ros.sh</li> <li>Enviar comandos de dirección rectilíneos</li> </ol>
Resultado esperado	Movimiento rectilíneo en la dirección deseada e indicada en el comando.
Resultados obtenidos	<ul style="list-style-type: none"> <li>El robot se movió en las direcciones enviadas</li> </ul>
Códigos de prueba necesarios	Anexo D

**Tabla 8-2: Prueba nro. 1 de iteración 5**

Caso de prueba PRF4-2	
Objetivo	Poner en funcionamiento el robot sobre una línea recta señalada en el suelo enviándole un comando de movimiento en la misma dirección.
Prerrequisitos	<ul style="list-style-type: none"> <li>• Tener una marca en el suelo correspondiente a una trayectoria recta</li> <li>• Se debe correr pid.launch de modo que Hermes III pueda activar el funcionamiento de sus motores.</li> <li>• Se debe correr el script “arduino_ros.sh” para que se establezca comunicación entre Arduino y Nvidia Jetson</li> </ul>
Pasos	<ol style="list-style-type: none"> <li>1. Colocar al Hermes III en el origen de la linea recta y con su dirección frontal apuntando hacia la línea</li> <li>2. Correr el launchfile pid.launch</li> <li>3. Correr el script arduino_ros.sh</li> <li>4. Enviar comandos de dirección</li> <li>5. Medir desviación respecto a la recta trazada</li> </ol>
Resultado esperado	El robot debe mantener un desplazamiento rectilíneo por encima de la línea marcada.
Resultados obtenidos	Si se consideran ambos estados, transitorio y estable, puede decirse que el error en línea recta fue de $10 \pm 1\text{cm}$ por metro recorrido. Si se descarta el estado transitorio (midiendo a partir del tercer punto), y extrapolando la trayectoria en línea recta en base a los puntos 3 y 4 de la Figura 8-5, se traza una línea roja ideal que dista unos $2 \pm 1\text{cm}$ de la posición final del robot. Para más información remitirse a la sección “Precisión del sistema de control”
Códigos de prueba necesarios	Anexo D

Tabla 8-3: Prueba nro. 2 de iteración 5

## 8.5 RESULTADOS

Se hicieron pruebas sobre los límites de velocidad y se obtuvo lo siguiente:

- Mínimo: 15.3 Hz (por desbordamiento del contador de 16 bits en placas Atmega). Esto es equivalente a 0.12 m/s según la sección “Obtención de velocidad lineal equivalente”.
- Máximo: 31.25 KHz (con lo cual se puede medir sobradamente la velocidad máxima a la que es capaz de funcionar el robot Hermes III).

Además, en base a la prueba PRF4-2, se obtuvo que la desviación del Hermes III es de aproximadamente  $\pm 0.1\text{ m}$  en un tramo de longitud de 1 m.

Por último, mientras se realizaban las pruebas, se han medido los tiempos de respuesta en los cuales el sistema de control empieza a mantenerse estable, y los resultados para 30 mediciones han sido de un promedio de 2.81 segundos.

## 8.6 RIESGOS MITIGADOS

ID	Riesgo	Probabilidad	Impacto	Exposición Final Iteración	Exposición Inicio Iteración
RI-01	Sistema Operativo inestable	1% 	2 	0,02	0,08
RI-02	Incompatibilidad o avería de componentes	10%	2	0,2	0,2
RI-03	Intercomunicación de componentes ineficiente o ineficaz	10% 	3 	0,3	0,75
RI-04	Prestaciones insuficientes de componentes	5% 	3 	0,15	0,45
RI-05	Modificación de los requerimientos del proyecto	10% 	4 	0,4	0,8
RI-06	Dificultad en conseguir determinados componentes	0%	4	0	0,2
RI-07	Excesivo tiempo para cumplir los objetivos del proyecto	10% 	5 	0,5	1,5
RI-08	Reducción de la fuerza de trabajo	10%	1 	0,1	0,2

Tabla 8-4: Riesgos mitigados en iteración 5

## 8.7 CONCLUSIONES

Se concluye que el sistema de control es una de las partes más importantes en el movimiento del robot, coordinando todas las ruedas para un robot omnidireccional fue un gran desafío. Se tuvo que programar algunos nodos para pequeños ajustes para mejorar el perfeccionamiento del mismo, pero la modularidad que ofrece ROS hace que todas estas implementaciones sean muchos más ágiles de utilizar.

En cuanto a la mitigación de riesgos se ha tenido en cuenta lo siguiente:

- RI-01: Se ha desarrollado nodos que puramente dependen del framework ROS y de sus sistema de comunicación entre nodos, y al obtener resultados positivos en las pruebas, se considera que el sistema operativo funciona correctamente para los requerimientos de esta iteración.
- RI-03: Se ha comprobado en el desarrollo de las pruebas que las comunicaciones mediante el protocolo TCP/IP entre los diferentes nodos de ROS son eficientes, logrando no tener retrasos ni perdidas de paquetes entre los diferentes canales de comunicación, por lo que este riesgo se ve disminuido.
- RI-04: Dado a las pruebas realizadas en la presente iteración, se concluye que los componentes más importantes del sistema cumplen los requerimientos hasta el momento, lo cual este riesgo se ve disminuido.
- RI-05: A medida que se avanza en el proyecto, y se van cumpliendo los requerimientos, la probabilidad de que los requerimientos sean modificados disminuye.

- RI-07: Este riesgo disminuirá al final de cada iteración siempre que los tiempos estipulados para cumplir con los objetivos del loop se cumplan. En este caso se ha demorado lo previsto.
- RI-08: En la medida que se avanza en el proyecto, menos impacta que disminuya el nivel de colaboración de un miembro del equipo, debido a que cada vez son menos las tareas pendientes.

# CAPÍTULO 9

## 9 ITERACIÓN 6: SISTEMAS DE MANDO

### 9.1 INTRODUCCIÓN

Para esta iteración se tendrán en cuenta todas las posibles formas de controlar o dirigir al Hermes III previstas en los requerimientos. Se implementará más de una forma de conectividad para demostrar la escalabilidad y modularidad de nuestro framework para robótica ROS.

### 9.2 REQUERIMIENTOS

En la presente iteración se atenderán los siguientes requerimientos funcionales:

RF5	Debe poder controlarse inalámbricamente
RNF1	Debe poderse interactuar con el robot desde múltiples plataformas (PC, Joystick, Smartphones)

Tabla 9-1: Requerimientos funcionales iteración 6

### 9.3 DESARROLLO

El desarrollo de esta iteración se dividirá en tres partes, una por cada forma de control que desea tenerse para el Hermes III. Las mismas son: teclado en PC de monitoreo, joystick inalámbrico comunicado directamente con el robot, y por último control de mando mediante un smartphone como un nodo más de la red ROS:

#### 9.3.1 MANDO A TRAVÉS DE TECLADO DE PC

Para el desarrollo del mando a través del teclado, se han reutilizado herramientas provistas por el package de “Turtlebot” (robot comercial de pruebas oficial de ROS).

Para ello se instalará el software necesario para comandar el robot turtlebot mediante el siguiente comando:

```
$ sudo apt-get install ros-indigo-turtlebot-teleop
```

Una vez obtenido el código del “*launchfile*” para el comando por teclado y dada su generalidad y flexibilidad, se ha procedido a modificarlo en base a las configuraciones del robot Hermes III, creando el launchfile que se presenta en el anexo E.3 keyboard\_teleop.launch y que al ejecutarlo puede verse en consola lo expuesto en la Figura 9-1.

Lo que hará el código será publicar un tópico del tipo “Twist” en base a la tecla predefinida que se presione en la PC de monitoreo, y éste será recibido por el sistema de control del Hermes III para comandar las ruedas del robot. El tópico definido para este fin es llamado “cmd\_vel”.

```

Terminal - /opt/ros/indigo/share/turtlebot_teleop/launch/keyboard_teleop.launch - + x
Archivo Editar Ver Terminal Pestañas Ayuda
turtlebot_teleop_keyboard (turtlebot_teleop/turtlebot_teleop_key)
ROS_MASTER_URI=http://localhost:11311

core service [/rosout] found
process[turtlebot_teleop_keyboard-1]: started with pid [30472]

Control Your Turtlebot!
-----
Moving around:
  u   i   o
  j   k   l
  m   ,   .

q/z : increase/decrease max speeds by 10%
w/x : increase/decrease only linear speed by 10%
e/c : increase/decrease only angular speed by 10%
space key, k : force stop
anything else : stop smoothly

CTRL-C to quit

currently:    speed 0.2      turn 1

```

Figura 9-1: Comando a través de teclado

### 9.3.2 MANDO A TRAVÉS DE JOYSTICK INALÁMBRICO

Para el desarrollo de esta funcionalidad, es necesario tener instalado el package “joy” de ros, para ello se puede realizar la instalación mediante el comando:

```
$ sudo apt-get install ros-indigo-joy
```

Luego, una vez instalado, se desarrolló el código para utilizar el joystick Xbox 360, tal como se muestra en el anexo E.1 xbox360.py. La función de este código es configurar la velocidad máxima de comando mediante una ponderación de los valores “axis” leídos por el controlador del joystick, y además poder enviar comandos de velocidad solo cuando el botón “Left Bumper” (Ver Figura 9-2) del joystick esté presionado. Cuando el mismo se deja de presionar, se envía un mensaje Twist publicando una velocidad de 0 en todas sus direcciones para indicar al sistema de control que se desea detener el robot.

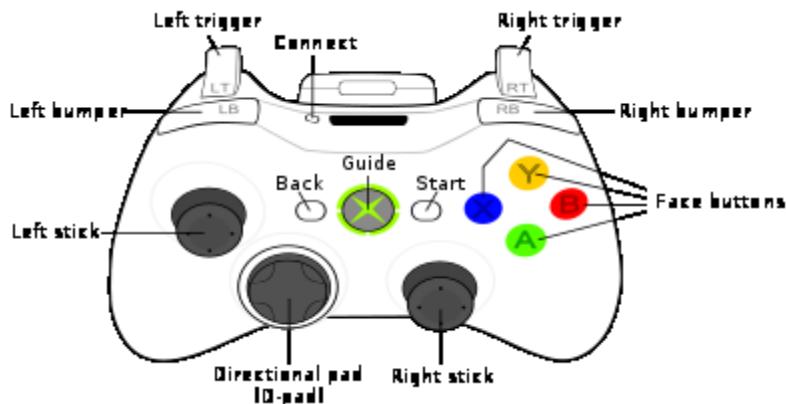


Figura 9-2: Referencia de botones de Joystick Xbox 360

Esto último se ha implementado debido a que cuando se suelta la palanca analógica “Left Stick”, ésta tiende a volver a su centro pero no logra hacerlo de manera perfecta, por ende, por su alta precisión, envía valores muy pequeños que no llegan a ser exactamente 0. Por ésta razón, sin la utilización del “Left bumper”, el robot Hermes tiende a seguir moviéndose en base éste valor muy pequeño que se envía.

Para comandar al Hermes mediante un joystick, es necesario correr el *launchfile* incluido en el anexo E.2 `xbox360_teleop.launch`. Éste último simplemente pone en funcionamiento el nodo “joy” de ROS y el nodo personalizado del Hermes III escrito en python E.1 `xbox360.py`

### 9.3.3 MANDO A TRAVÉS DE SMARTPHONE

Dada la gran magnitud del ecosistema de Android, no fue difícil encontrar desarrolladores que se dediquen a realizar aplicaciones para comandar y monitorear un robot utilizando el framework de ROS.

Gracias a la esencia modular de ROS, éstas aplicaciones pueden ser descargadas y simplemente configurando los tópicos a publicar y a suscribirse, podemos tener un smartphone totalmente integrado en nuestro sistema Hermes III. Ejemplos de estas apps son:

- ❖ ROS Control (Figura 9-3)
- ❖ ROS Teleop
- ❖ ROS Mover
- ❖ ROS Sensors

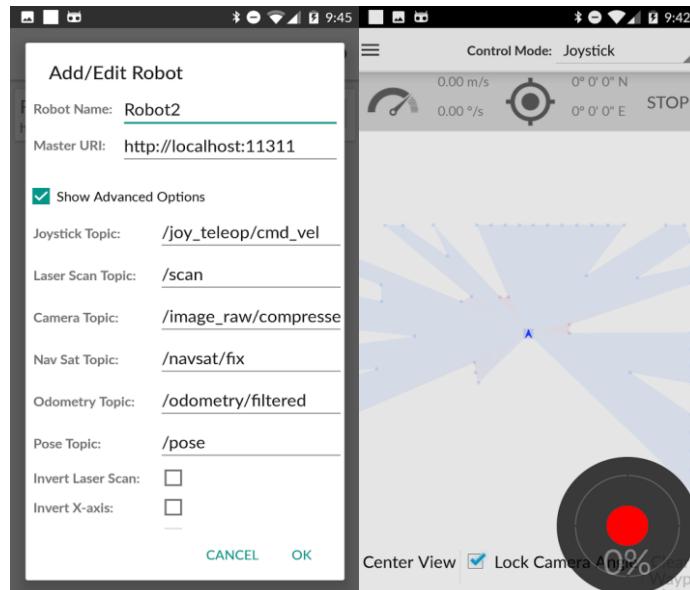


Figura 9-3: Aplicación Ros Control para Android

Mas allá de las aplicaciones Android existentes, las cuales en su mayoría son Open Source, pueden crearse otras totalmente personalizadas ya que ROS brinda la librería “*rosjava*” que permite a cualquier smartphone ser parte del ecosistema de ROS.

La gran ventaja de ésto, es que los teléfonos celulares actuales, cuentan con una gran cantidad de sensores, tales como giroscopio, barómetro, acelerómetro, GPS y cámaras (en algunos casos estereoscópicas), los cuales pueden ser aprovechados para extender las capacidades de un sistema robótico. Por ejemplo, algunas aplicaciones de control que se han probado utilizan el acelerómetro del *smartphone* para mover a un robot en base a la inclinación del dispositivo móvil.

#### 9.4 PRUEBAS

Caso de prueba PRF5	
Objetivo	Enviar comandos de movimiento al robot desde diferentes controles remotos, ya sea un joystick, un teclado de un sistema remoto, o un smartphone.
Prerrequisitos	<ul style="list-style-type: none"> <li>● Prerequisito segun mando: <ul style="list-style-type: none"> <li>○ Joystick: Se debe contar con un joystick Xbox 360 conectado y correr <code>xbox360_teleop.launch</code></li> <li>○ Teclado: Se debe correr el launchfile <code>keyboard_teleop.launch</code> desde una maquina conectada al roscore de la Nvidia Jetson</li> <li>○ Smartphone: Se debe tener una de las aplicaciones mencionadas en la sección de desarrollo conectada al roscore de la Nvidia Jetson y setear como tópico de joystick "cmd_vel"</li> </ul> </li> <li>● Se debe correr <code>pid.launch</code> de modo que Hermes III pueda activar el funcionamiento de sus motores.</li> <li>● Se debe correr el script "<code>arduino_ros.sh</code>" para que se establezca comunicación entre Arduino y Nvidia Jetson</li> </ul>
Pasos	<ol style="list-style-type: none"> <li>1. Correr el launchfile <code>pid.launch</code></li> <li>2. Correr el script <code>arduino_ros.sh</code></li> <li>3. Correr el launchfile que corresponda al mando elegido</li> <li>4. Enviar comandos de direcciones</li> </ol>
Resultado esperado	El robot debe moverse en las direcciones y sentidos indicados por el comando de control.
Resultados obtenidos	<ul style="list-style-type: none"> <li>● El robot se mueve de acuerdo a la dirección y velocidad establecida por el mando de control</li> </ul>
Códigos de prueba necesarios	Anexo E.1 / E.2 / E.3

Tabla 9-2: Prueba nro. 1 de iteración 6

## 9.5 RESULTADOS

Los resultados de acuerdo a la prueba de manejo han sido aceptables. Se ha logrado conducir al Hermes III mediante los 3 mandos que se establecieron, y ambos de manera correcta y agradable, desplazándose el mismo en las direcciones y velocidades establecidas, a tiempos de respuesta muy rápidos y sin errores.

## 9.6 RIESGOS SUPERADOS

El análisis de mitigación de riesgos al finalizar la presente iteración es el siguiente:

ID	Riesgo	Probabilidad	Impacto	Exposición Final Iteración	Exposición Inicio Iteración
RI-01	Sistema Operativo inestable	1%	2	0,02	0,04
RI-02	Incompatibilidad o avería de componentes	5% 	2 	0,1	0,2
RI-03	Intercomunicación de componentes inefficiente o ineficaz	0% 	3 	0	0,3
RI-04	Prestaciones insuficientes de componentes	0% 	3 	0	0,15
RI-05	Modificación de los requerimientos del proyecto	0% 	4 	0	0,4
RI-06	Dificultad en conseguir determinados componentes	0%	4	0	0,2
RI-07	Excesivo tiempo para cumplir los objetivos del proyecto	0% 	5 	0	0,5
RI-08	Reducción de la fuerza de trabajo	10%	0 	0	0,1

Tabla 9-3: Riesgos mitigados en iteración 6

## 9.7 CONCLUSIONES

Se ha concluido que la modularidad de ROS hace que el desarrollo de dispositivos de comando sea rápido. Se ha realizado, sin mayor dificultad, la inclusión de tres sistemas de mandos diferentes.

En cuanto a la mitigación de riesgos se ha tenido en cuenta lo siguiente:

- RI-02: Se ha visto que mediante un package llamado "joy", se puede comunicar y programar el comportamiento de cualquier tipo de Joystick. También la gran variedad de aplicaciones móviles para los smartphones que son realizadas para el entorno de ROS, hacen que el riesgo de incompatibilidad de estos elementos que interactúan con este framework se vea disminuido.
- RI-03: En la presente iteración se han incorporado 3 dispositivos nuevos al robot Hermes 3 para poder comandarlo, y no se han tenido problemas de pérdida de conexión o bajos rendimientos con ninguno de ellos. Por ésta razón el riesgo en cuestión se ve mitigado.

- RI-04: Dada la reacción casi instantánea por parte del robot Hermes a cada uno de los comandos enviados por los diferentes medios, se puede concluir que los componentes agregados al sistema cumplen los requerimientos hasta el momento, lo cual este riesgo se ve disminuido.
- RI-05: A medida que se avanza en el proyecto, y se van cumpliendo los requerimientos, la probabilidad de que los requerimientos sean modificados disminuye. Al ser ésta la última iteración y al tener una buena comunicación con el Director del proyecto Hermes III, este riesgo se ve reducido a un número muy pequeño dado que se han cumplido satisfactoriamente las expectativas.
- RI-07: Este riesgo disminuirá al final de cada iteración siempre que los tiempos estipulados para cumplir con los objetivos del loop se cumplan. En este caso se ha demorado lo previsto.
- RI-08: En la medida que se avanza en el proyecto, menos impacta que disminuya el nivel de colaboración de un miembro del equipo, debido a que cada vez son menos las tareas pendientes. En este caso disminuye a cero dado que se ha dado por terminado el proyecto.

# CAPÍTULO 10

## 10 SISTEMA FINAL

### 10.1 INTRODUCCIÓN

El desarrollo del robot Hermes III se ha dado por terminado en la iteración anterior número 6. En el presente capítulo se explicará la integración de todos sus componentes, tanto de software como de hardware, mediante un diagrama de despliegue. Además, se mostrará como es la puesta en funcionamiento del robot y se describirán casos de usos del mismo, que demostrarán cuáles son sus capacidades.

### 10.2 INTEGRACIÓN DE COMPONENTES

El diagrama de despliegue que se expondrá a continuación en la Figura 10-1, intentará explicitar en dónde se ejecutan los principales programas dentro de la arquitectura del Hermes III. Los códigos fuente de los mismos se encuentran en los Anexos del presente informe.

#### 10.2.1 DIAGRAMAS DE DESPLIEGUE

##### 10.2.1.1 HERMES 3

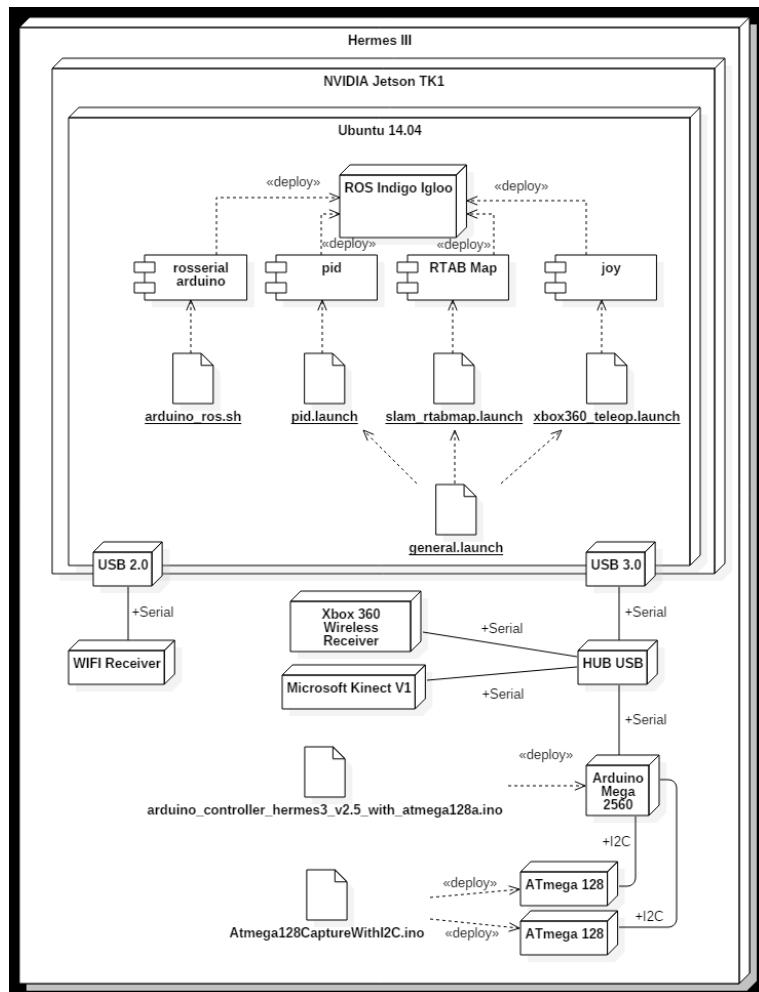


Figura 10-1: Diagrama de despliegue Hermes 3

Dentro del nodo NVIDIA Jetson TK1, los componentes “rosserial arduino”, “pid”, “RTAB Map” y “joy” son los paquetes que debe tener instalado el Framework ROS. A su vez, se especifica debajo de cada uno de estos componentes, los programas que hacen uso de los mismos y que deben ejecutarse para el funcionamiento del Hermes 3.

Se tiene el *launcher* “`general.launch`”, el cual es capaz de lanzar a ejecución los tres archivos que se especifican en el diagrama, con lo cual no es necesario ejecutar uno por uno los mismos. Por otra parte, se tiene el código “`arduino_ros.sh`”, el cual es un sencillo script Bash. Cabe destacar que éste archivo en cuestión es específico para la placa de desarrollo Arduino que se utiliza, dado que incorpora un ID de dispositivo único, para que el Framework ROS la identifique y establezca la conexión.

Además, se puede apreciar en el diagrama, los periféricos que se encuentran conectados a la placa NVIDIA mediante USB 2.0 y USB 3.0.

Los códigos de programa “`arduino_controller_hermes3_v2.5_with_atmega1238a.ino`” y “`Atmega128CaptureWithI2C.ino`” deben ser desplegados en las respectivas placas de desarrollo. Ambos programas, especifican en su código fuente los ID de conexión para la comunicación maestro-esclavo mediante el protocolo I2C, por lo que debe prestarse especial atención a la hora de conectar las placas Atmega a la Arduino, ya que si se invierten los ID’s de los esclavos Atmega, se mapearán incorrectamente las posiciones de las ruedas, provocando que los algoritmos de control y posicionamiento no funcionaran correctamente.

#### 10.2.1.2 PC DE MONITOREO

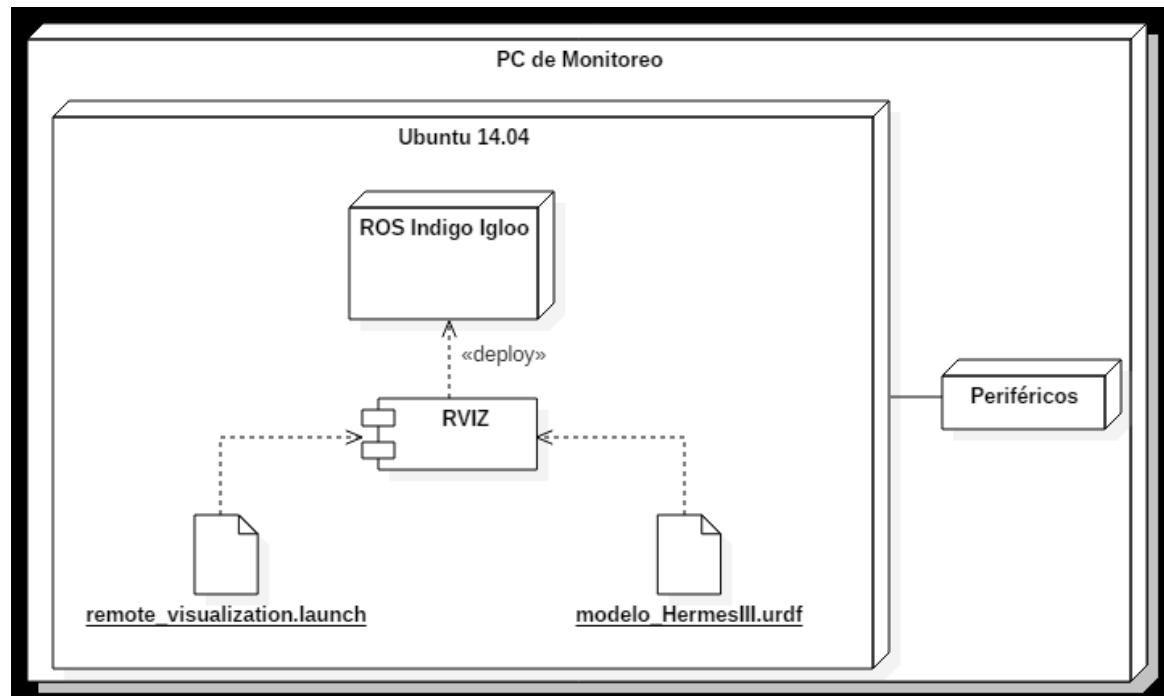


Figura 10-2: Diagrama de despliegue PC de Monitoreo

En la PC desde la cual se realizará el monitoreo del SLAM realizado por el Hermes 3, se deberá contar con una distribución de Linux Ubuntu 14.04 con el Framework ROS Indigo instalado. Luego, gracias al paquete “RVIZ”, podrá ejecutarse el *launchfile* “`remote_visualization.launch`” que configurará todo lo necesario para el monitoreo, entre ello, cargará el modelo 3D del robot Hermes III, especificado en el archivo “`modelo_HermesIII.urdf`” (Ver Figura 10-2).

## 10.3 PUESTA EN FUNCIONAMIENTO

Una vez que todos los componentes de hardware y software se encuentren correctamente configurados según el diagrama de despliegue anterior, se deberán seguir los siguientes pasos para lograr poner en funcionamiento al robot Hermes III.

### 10.3.1 CONFIGURACION PREVIA

Teniendo conectado un monitor, teclado y mouse en la NVIDIA Jetson TK1, se deberán realizar las siguientes configuraciones en el sistema operativo.

1. Activar el “inicio de sesión automático” de Ubuntu, para que al encender la placa dentro del robot Hermes III, se ingrese en el sistema operativo sin intervención humana.
2. Instalar servidor SSH para acceder a la placa NVIDIA Jetson desde la red donde estará conectada y poder ejecutar los *launchfiles* correspondientes de manera remota.
3. Configurar una o varias redes WiFi conocidas, para que la placa NVIDIA se conecte a alguna de ellas de manera automática alloguearse en el sistema operativo. La red WiFi puede ser una red Ad Hoc, es decir, una conexión inalámbrica directa entre la placa del Hermes III y la PC de monitoreo.

### 10.3.2 ENCENDER EL ROBOT

Una vez que se tenga todo configurado e instalado en el robot hasta la “configuración previa”, basta con conectar el robot Hermes III a la batería de 12V. En la mayoría de los casos, la placa NVIDIA se enciende automáticamente al conectarla, pero en otras ocasiones es necesario encenderla cortocircuitando los pines 6 y 8 de su panel frontal (Ver Figura 10-3).

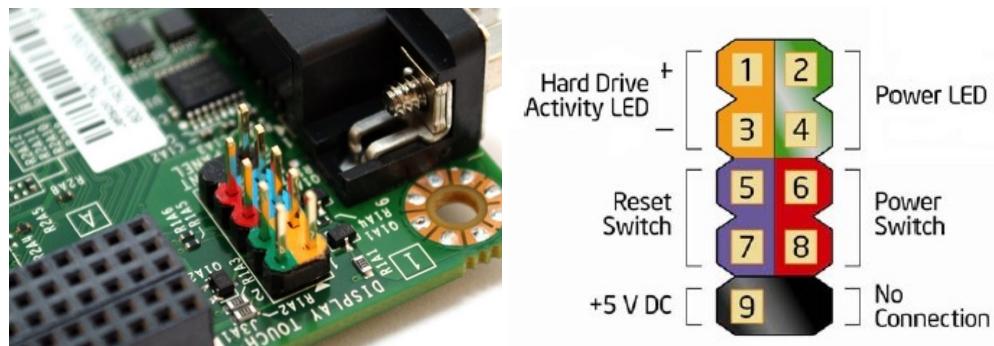


Figura 10-3: Pines panel frontal NVIDIA Jetson

### 10.3.3 VERIFICACIÓN DE CONECTIVIDAD

Una vez que se termine el proceso de arranque del sistema operativo del robot, el mismo deberá conectarse a una de las redes WiFi preconfiguradas disponibles.

Luego desde cualquier software de monitoreo de redes, podremos ver la IP que se le ha asignado al robot y realizar un ping a la misma para verificar conectividad.

### 10.3.4 CONFIGURAR IP'S MAESTRO-ESCLAVO DE ROS

#### 10.3.4.1 NVIDIA JETSON

Conociendo la dirección IP asignada al Hermes, desde la PC de monitoreo nos conectaremos por SSH al mismo y se modificará su archivo “bashrc” ejecutando los siguientes comandos.

Suponiendo que la IP del Hermes III es 192.168.1.143, la conexión al mismo muestra en consola lo expuesto en la Figura 10-4.

```
turtlebot@turtlebot-N56VB:~$ ssh ubuntu@192.168.1.143
ubuntu@192.168.1.143's password:
Welcome to Ubuntu 14.04.5 LTS (GNU/Linux 3.10.40-ga7da876 armv7l)

 * Documentation:  https://help.ubuntu.com/
 
222 packages can be updated.
54 updates are security updates.

Last login: Fri Dec 31 21:18:24 1999
ubuntu@tegra-ubuntu:~$ vim .bashrc
```

Figura 10-4: Conexión por SSH al Hermes III

Luego, en el archivo abierto, se deberán modificar las variables “ROS\_MASTER\_URI” y “ROS\_IP” o agregar las siguientes dos líneas en caso de que no existan:

```
export ROS_MASTER_URI=http://192.168.1.143:11311
export ROS_IP=192.168.1.143
```

Después de guardar el archivo, se aplican los cambios ejecutando:

```
$ source .bashrc
```

#### 10.3.4.2 PC DE MONITOREO

Localmente, en la PC de monitoreo, se realizarán los mismos pasos anteriores para la modificación del archivo “bashrc”, colocando como “ROS\_MASTER\_URI” la misma que la colocada en el archivo del robot Hermes III (dado que será el nodo maestro en la red de ROS) y como “ROS\_IP” se debe colocar la IP de la máquina local.

## 10.3.5 EJECUCIÓN DE LAUNCHFILES

### 10.3.5.1 NVIDIA JETSON

A través de dos conexiones establecidas por SSH al robot, se ejecutará en una el *launchfile* “general.launch” y en la otra el script “arduino\_ros.sh”, obteniendo en consola lo mostrado en la Figura 10-5 y Figura 10-6.

```
ubuntu@tegra-ubuntu:~$ rosrun hermesIII general.launch
WARNING: Package name "hermesIII" does not follow the naming conventions. It should start with a lower case letter and only contain lower case letters, digits and underscores.
... logging to /home/ubuntu/.ros/log/439bc7b2-5090-11e7-a2bf-7cdd90735f51/rosrun-tegra-ubuntu-2578.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

WARNING: Package name "hermesIII" does not follow the naming conventions. It should start with a lower case letter and only contain lower case letters, digits and underscores.
```

Figura 10-5: Ejecución de general.launch por SSH 1

```
ubuntu@tegra-ubuntu:~$ roscd hermesIII/script/
ubuntu@tegra-ubuntu:~/hermes3/src/hermesIII/scripts$ ./arduino_ros.sh
[INFO] [WallTime: 1497396647.113000] ROS Serial Python Node
[INFO] [WallTime: 1497396647.130385] Connecting to /dev/serial/by-id/usb-Arduino__www.arduino.cc__0042_5533834353935110A0B0-if00 at 115200 baud
[INFO] [WallTime: 1497396649.293864] Note: publish buffer size is 512 bytes
[INFO] [WallTime: 1497396649.294707] Setup publisher on /right_wheel/state [std_msgs/Float64]
[INFO] [WallTime: 1497396649.302926] Setup publisher on /back_wheel/state [std_msgs/Float64]
[INFO] [WallTime: 1497396649.313245] Setup publisher on /left_wheel/state [std_msgs/Float64]
[INFO] [WallTime: 1497396649.323746] Setup publisher on /front_wheel/state [std_msgs/Float64]
[INFO] [WallTime: 1497396649.329189] Setup publisher on /PWM_LeftValue [std_msgs/Int16]
[INFO] [WallTime: 1497396649.335547] Setup publisher on /PWM_RightValue [std_msgs/Int16]
[INFO] [WallTime: 1497396649.347482] Setup publisher on /PWM_BackValue [std_msgs/Int16]
[INFO] [WallTime: 1497396649.354381] Setup publisher on /PWM_FrontValue [std_msgs/Int16]
[INFO] [WallTime: 1497396649.364133] Note: subscribe buffer size is 512 bytes
[INFO] [WallTime: 1497396649.365208] Setup subscriber on /send_velocity [std_msgs/UInt8]
[INFO] [WallTime: 1497396649.374329] Setup subscriber on /left_wheel/control_effort [std_msgs/Float64]
[INFO] [WallTime: 1497396649.391031] Setup subscriber on /right_wheel/control_effort [std_msgs/Float64]
[INFO] [WallTime: 1497396649.400508] Setup subscriber on /front_wheel/control_effort [std_msgs/Float64]
[INFO] [WallTime: 1497396649.418665] Setup subscriber on /back_wheel/control_effort [std_msgs/Float64]
```

Figura 10-6: Ejecución de arduino\_ros.sh por SSH 2

### 10.3.5.2 PC DE MONITOREO

Se ejecutará el *launchfile* “remote\_visualization.launch”, según se muestra en la Figura 10-7.

```
turtlebot@turtlebot-N56VB:~$ rosrun hermesIII remote_visualization.launch
WARNING: Package name "hermesIII" does not follow the naming conventions. It should start with a lower case letter and only contain lower case letters, digits and underscores.
... logging to /home/turtlebot/.ros/log/439bc7b2-5090-11e7-a2bf-7cdd90735f51/rosrun-turtlebot-N56VB-3639.log
Checking log directory for disk usage. This may take awhile.
Press Ctrl-C to interrupt
Done checking log file disk usage. Usage is <1GB.

WARNING: Package name "hermesIII" does not follow the naming conventions. It should start with a lower case letter and only contain lower case letters, digits and underscores.
started rosrun server http://192.168.1.140:42966/
SUMMARY
=====
PARAMETERS
* /rostdistro: indigo
* /rosversion: 1.11.21
NODES
/
  rviz (rviz/rviz)
ROS_MASTER_URI=http://192.168.1.143:11311
core service [/rosout] found
WARNING: Package name "hermesIII" does not follow the naming conventions. It should start with a lower case letter and only contain lower case letters, digits and underscores.
process[rviz-1]: started with pid [3648]
```



Figura 10-7: Ejecución de remote\_visualization.launch

A continuación, se abrirá el software RVIZ en el que se podrá ver al robot Hermes III y el SLAM que realiza en tiempo real (Ver Figura 10-8).

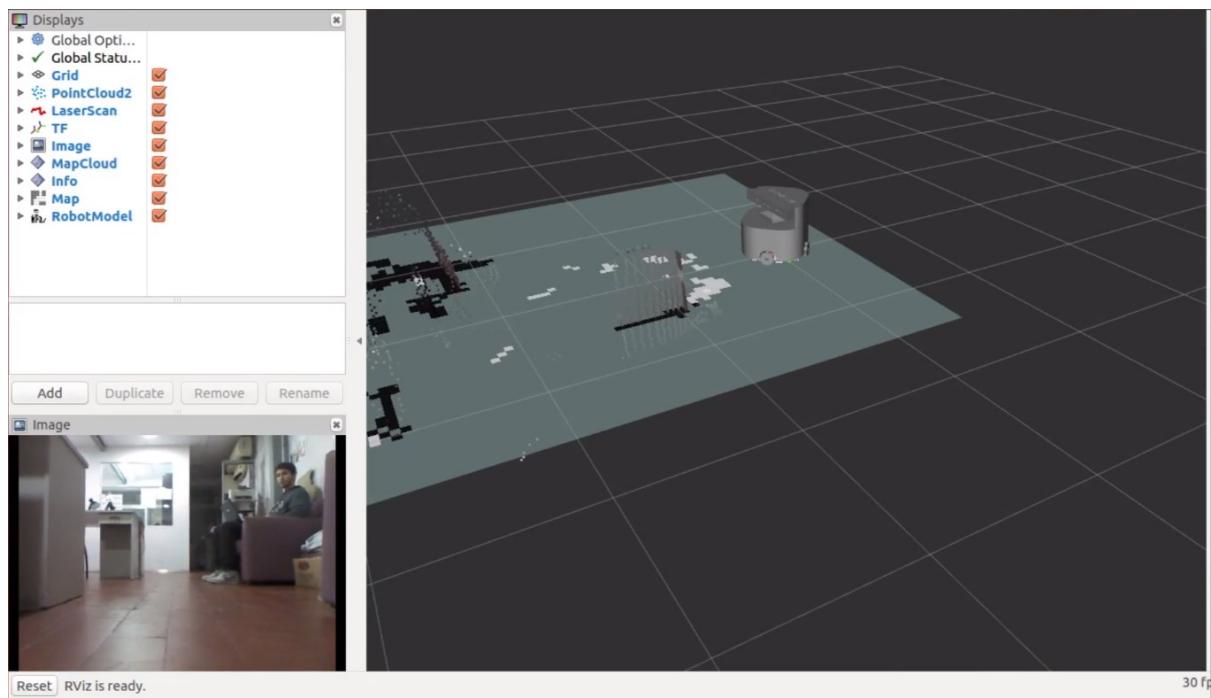


Figura 10-8: RVIZ en PC de monitoreo

#### 10.4 COMANDOS DE CONTROL

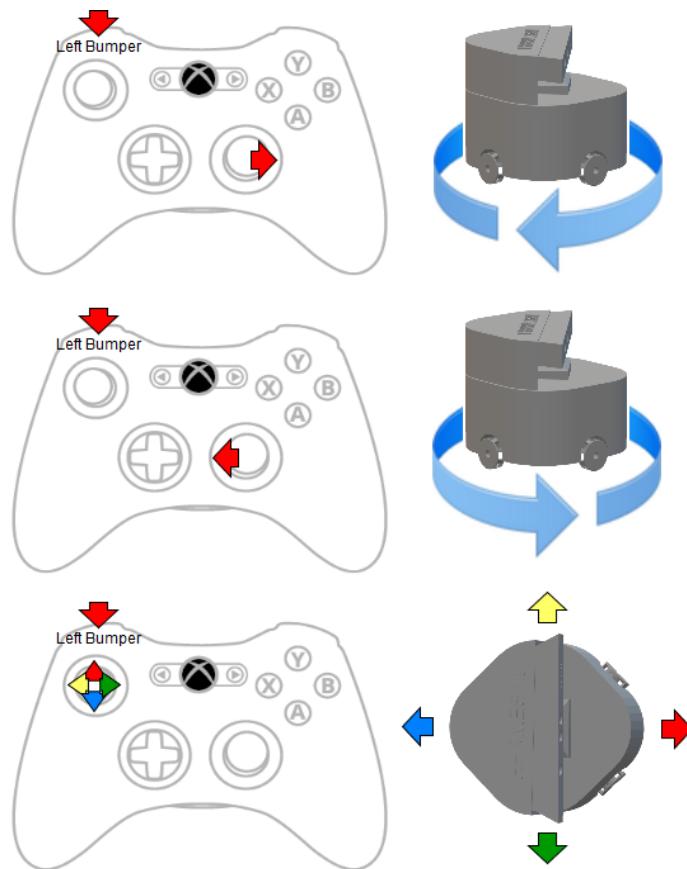


Figura 10-9: Comando del Hermes III por joystick

Recordar que todo comando de movimiento que se envíe al Hermes III mediante el joystick de Xbox 360 debe hacerse presionando el botón “Left Bumper” como se indica en la Figura 10-9. Al no presionar dicho botón, el robot ignora todo comando proveniente del joystick y permanece inmóvil.

#### 10.5 PRUEBAS DE INTEGRACIÓN

Caso de prueba PRF8-2	
Objetivo	Comparar el modelo 3D realizado con el software CAD frente al modelo que se construyó.
Prerrequisitos	<ul style="list-style-type: none"><li>Tener el archivo “Hermes3_2017.stl” o en su defecto “Hermes3_2017.dae”</li><li>Contar con un software CAD para visualizarlo o utilizar alguna herramienta de visualización web.</li></ul>
Pasos	1. Abrir el archivo en formato STL o DAE en el software CAD correspondiente
Resultado esperado	El modelo 3D y el modelo real deben tener similar aspecto y proporciones.

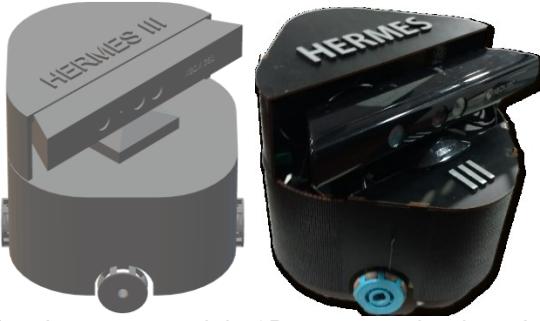
Resultados obtenidos	 <p>Se obtuvo un modelo 3D muy acorde al modelo final del Hermes III.</p>
Códigos de prueba necesarios	Anexo E.1 / E.2 / E.3

Tabla 10-1: Prueba nro. 1 Sistema Final

Caso de prueba PRF7-2	
Objetivo	Contrastar el comportamiento real del robot en funcionamiento con el comportamiento en el software de monitorización.
Prerrequisitos	<ul style="list-style-type: none"> <li>Tener el sistema implementado de la manera que se indica en los “<a href="#">Diagramas de despliegue</a>”.</li> <li>Realizar la “<a href="#">Configuración previa</a>” que se detalla en el presente capítulo.</li> </ul>
Pasos	Seguir los pasos indicados en la sección “ <a href="#">Puesta en funcionamiento</a> ”
Resultado esperado	Los datos recolectados y visualizados deben ir a la par del movimiento del robot, siendo consistentes y coherentes con la realidad.
Resultados obtenidos	Pueden apreciarse los resultados obtenidos en el video demostración del Hermes III adjunto al presente informe.
Códigos de prueba necesarios	Anexo E.1 / E.2 / E.3

Tabla 10-2: Prueba nro. 2 Sistema Final

Caso de prueba PRF9	
Objetivo	Colocar el robot en una habitación desconocida y dejarlo funcionar autónomamente (sin recibir órdenes), hasta que se logre obtener el mapa completo de la habitación.
Prerrequisitos	
Pasos	
Resultado esperado	El mapa resultante debe ser coherente y consistente con la realidad. Además, durante la monitorización del robot, no deben detectarse choques bruscos contra obstáculos.
Resultados obtenidos	Por cuestiones de tiempo no se ha llegado a cumplir con el requerimiento de autonavegación del robot, pero todas las condiciones están dadas para poder hacerlo por lo que se planteará como trabajo a futuro.

Tabla 10-3: Prueba nro.3 Sistema Final

## 10.6 ESTADO FINAL DE REQUERIMIENTOS

Se exponen las siguientes tablas con el fin de explicitar el estado final de los requerimientos planteados al inicio del presente informe.

ID	Descripción	Estado Final
RF1	Debe tener un sistema de locomoción omnidireccional	Cumplido
RF2	Debe poder auto localizarse	Cumplido
RF3	Debe poder realizar un mapa	Cumplido
RF4	Debe tener un sistema de control de movimiento eficaz	Cumplido
RF5	Debe poder controlarse inalámbricamente	Cumplido
RF6	Debería realizarse un modelo de simulación previo a la implementación	Cumplido
RF7	Debería tenerse un sistema de monitorización remoto de los sensores del robot	Cumplido
RF8	Podría realizarse un modelo tridimensional del robot	Cumplido
RF9	Quiere tener la capacidad de autonavegar	No Cumplido

Tabla 10-4: Estado final de requerimientos funcionales

Código	Descripción	Estado Final
Genéricos		
RNF1	Debe poderse interactuar con el robot desde múltiples plataformas (PC, Joystick, Smartphones)	Cumplido
RNF2	Debe contar con documentación de código y de proyecto	Cumplido
RNF3	Debe contar con pruebas y tests de funcionalidades	Cumplido
RNF4	Debería tener una carcasa con un diseño estructural robusto	Cumplido
RNF5	Debería tener tiempos de respuesta aceptables para un buen funcionamiento del sistema de control	Cumplido
Software		
RNF6	Debe utilizarse un Sistema Operativo específico para robótica	Cumplido
RNF7	Debería poderse programar en C++, Java o Python	Cumplido
RNF8	Debería utilizarse un Sistema Operativo y Framework OpenSource	Cumplido
Hardware		
RNF9	Debe tener suficientes entradas para sensores	Cumplido
RNF10	Debe tener suficientes salidas para actuadores	Cumplido
RNF11	Debe permitir el desarrollo modular	Cumplido
RNF12	Debería utilizarse componentes que cuenten con documentación	Cumplido

Tabla 10-5: Estado final de requerimientos no funcionales

## 11 CONCLUSIONES

Se logró construir un robot omnidireccional capaz de realizar un mapa en 3D del entorno que lo rodea y al mismo tiempo ubicarse dentro del mismo. Éstas capacidades sientan las bases para el desarrollo a futuro de un robot totalmente independiente del control humano, es decir, que sea capaz de auto navegar una habitación, evitando obstáculos y realizando trayectorias de barrido eficientes para lograr un mapa tridimensional en el menor tiempo posible.

Desde el punto de vista académico, el robot desarrollado puede servir como base para el estudio del estado del arte de la robótica, mediante el innovador Framework que se ha utilizado, denominado Robot Operating System, el cual tiene un enorme futuro por su naturaleza Open Source y la gran comunidad que posee, conformada tanto por investigadores de universidades de gran prestigio, como por estudiantes y aficionados.

El framework ROS se basa en el desarrollo modular, por ende, a futuro se puede trabajar en versiones modificadas del robot Hermes III, dotándolo de nuevas capacidades, en caso de ser necesario, para automatizar tareas que actualmente las realiza un ser humano.

Ésta versión del robot que se ha desarrollado íntegramente en el Laboratorio de Arquitectura de Computadoras de la Facultad de Ciencias Exactas, Físicas y Naturales ha significado un gran salto tecnológico con respecto a su predecesor, el Hermes II.

Se ha aplicado un método de desarrollo iterativo que ha permitido avanzar en el proyecto sin mayores contratiempos y logrando resultados satisfactorios. Aún así, no significa que las decisiones que se hayan tomado en determinadas ocasiones, hayan sido las mejores. La oportunidad de perfeccionamiento de las soluciones aplicadas, existe, y plantea un futuro prometedor para aquellas personas que deseen continuar con el proyecto Hermes.

Respecto a la utilización de un sistema onmidireccional heredado de su antecesor Hermes II, si bien su control trae aparejado una mayor complejidad matemática que otros sistemas de locomoción, facilita su implementación mecánica debido a que los ejes de las ruedas permanecen fijos y permite modificar la trayectoria del robot sin cambiar la orientación del mismo.

Por otro lado, con la placa de desarrollo Nvidia Jetson Tegra K1, se contaba con capacidad computacional suficiente para realizar cálculos complejos, tales como los vinculados al procesamiento de los datos provenientes del sensor Microsoft Kinect y al sistema de control que comanda los movimientos del robot.

Particularmente, se han tenido problemas dada la baja resolución del sensor de profundidad, lo cual provocaba estimaciones de posición errónea bajo determinadas condiciones, como se detalla en el capítulo correspondiente.

En cuanto a la medición de velocidad de las ruedas, se ha tomado una buena decisión debido a su simplicidad de implementación y su gran eficacia, demostrada en las pruebas finales que se realizaron. Aún así, este aspecto es una gran oportunidad de mejora, dado que en el actual proyecto no se han utilizado estos datos para hacer más robusto el algoritmo de SLAM, el cual utiliza únicamente los datos obtenidos del sensor de imágenes que posee una baja resolución.

## 12 BIBLIOGRAFÍA

- [1] DIARIO ABC, S.L., 15 03 2016. [En línea]. Available: [http://www.abc.es/ciencia/abci-principales-misiones-marte-201603142324\\_noticia.html](http://www.abc.es/ciencia/abci-principales-misiones-marte-201603142324_noticia.html).
- [2] E. B. A. Y. N. Morgan Quigley, «STAIR: Hardware and Software Architecture,» AAAI 2007 Robotics Workshop, Stanford University, 2007.
- [3] J. P. Fentanes, «Exploración y reconstrucción tridimensional de entornos mediante robots móviles,» Valladolid, 2012.
- [4] E. & L. Borenstein, *Navigating Mobile Robots: Systems and Techniques*, A K Peters Ltd, 1996.
- [5] I. Sommerville, *Ingeniería de software*, 9th edition ed., Pearson Educación, 2011.
- [6] Cyber Rovers, «Risk Mitigation, Monitoring, and Management Plan,» WMITS System Specification, 2016.
- [7] H. Y. Y. Higuera Ronald P., «Software Risk Management, Technical Report,» SEI (Software Engineering Institute), Carnegie Mellon University, Pittsburgh, Pennsylvania, 1996.
- [8] M. d. C. G. F. J. Cervantes Ojeda, «Taxonomía de los modelos y metodologías de desarrollo de software más utilizados,» Unidad Cuajimalpa, México, 2012.
- [9] Universidad de Sevilla, «Sistemas Sistemas de Locomoción de Robots Móviles,» 2016. [En línea]. Available: [http://www.esi2.us.es/~vivas/ayr2iae/Loc\\_MOV.pdf](http://www.esi2.us.es/~vivas/ayr2iae/Loc_MOV.pdf).
- [10] J. v. d. B. D. F. Mototaka Suzuki, «Omnidirectional Active Vision for Evolutionary Car Driving,» IOS Press, 2016.
- [11] P. Zanuttig, G. Marin, C. Dal Mutto, F. Dominio, L. Minto y G. M. Cortelazzo, *Time-of-Flight and Structured Light Depth Cameras*, Springer, 2016.
- [12] Instituto Nacional de Tecnología Industrial, «Vocabulario Internacional de Metrología,» 2008. [En línea]. Available: [https://www.inti.gob.ar/fisicaymetrologia/pdf/span\\_VIM.pdf](https://www.inti.gob.ar/fisicaymetrologia/pdf/span_VIM.pdf).
- [13] Revolution Robotics, Inc., «Revotics,» [En línea]. Available: [https://revotics.com/articles/lithium\\_polymer\\_lipo\\_battery\\_guide?v=5b61a1b298a0](https://revotics.com/articles/lithium_polymer_lipo_battery_guide?v=5b61a1b298a0).
- [14] M. Emir, «Top-Codig,» 01 02 2012. [En línea]. Available: <http://topcoding.blogspot.com.ar/2012/02/baterias-comparativa-litio-nimh-nicd.html>.
- [15] K. Ogata, *Ingeniería de control moderna* 5 ed., Madrid, España: Pearson, 2010.

- [16] R. Rojas y A. G. Förster, «Holonomic Control of a robot with an omnidirectional drive,» 2006.
- [17] B. G. W. D. S. Morgan Quigley, Programming Robots With ROS, O'Reilly, 2015.
- [18] «Sistemas de comunicación,» 2013. [En línea]. Available: <https://sistemascomunic.wordpress.com/sistemas-de-comunicacion/>.
- [19] C. Commons, «CCM,» 10 07 2017. [En línea]. Available: <http://es.ccm.net/contents/281-protocolo-tcp>.
- [20] SparkFun Electronics, «Sparkfun,» Niwot, Colorado, 07 08 2016. [En línea]. Available: <https://learn.sparkfun.com/tutorials/bluetooth-basics>.
- [21] CCM Benchmark Group, «CCM,» 15 10 2016. [En línea]. Available: <http://es.ccm.net/contents/789-introduccion-a-wi-fi-802-11-o-wifi>.
- [22] ROS.org, «Ubuntu ARM install of ROS Indigo,» [En línea]. Available: <http://wiki.ros.org/indigo/Installation/UbuntuARM>. [Último acceso: 2016].
- [23] NVIDIA, «NVIDIA,» [En línea]. Available: <http://www.nvidia.es/object/tegra-k1-processor-es.html>.
- [24] ROS, «Ros Wiki URDF,» 17 10 2013. [En línea]. Available: <http://wiki.ros.org/urdf/XML>.
- [25] O. R. P., «SISTEMAS O.R.P,» 20 Marzo 2013. [En línea]. Available: <https://www.sistemasorp.es/2013/03/20/calcular-el-torque-de-los-motores-para-un-robot-velocista-o-de-sumo/>. [Último acceso: 2017].
- [26] Society of Robots, «Society of Robots,» 2005. [En línea]. Available: [http://www.societyofrobots.com/mechanics\\_dynamics.shtml](http://www.societyofrobots.com/mechanics_dynamics.shtml). [Último acceso: 2017].
- [27] A. Cruz, «ElectronicLab ingeniería y diseño electrónico,» 17 Mayo 2014. [En línea]. Available: <https://electronilab.co/tutoriales/tutorial-de-uso-driver-dual-l298n-para-motores-dc-y-paso-a-paso-con-arduino/>. [Último acceso: 2017].
- [28] Hans, «MCUdude,» [En línea]. Available: <https://github.com/MCUdude/MegaCore>. [Último acceso: 2017].
- [29] ROS, «ROS Wiki Arduino,» 2016. [En línea]. Available: [http://wiki.ros.org/rosserial\\_arduino/Tutorials/Arduino%20IDE%20Setup](http://wiki.ros.org/rosserial_arduino/Tutorials/Arduino%20IDE%20Setup).
- [30] IntRoLab, «RTAB-Map,» Université de Sherbrooke, 07 2017. [En línea]. Available: <http://introlab.github.io/rtabmap/>.
- [31] ROS, «ROS Wiki RTAB Map,» 20 09 2016. [En línea]. Available: [http://wiki.ros.org/rtabmap\\_ros/Tutorials/SetupOnYourRobot](http://wiki.ros.org/rtabmap_ros/Tutorials/SetupOnYourRobot).

[32] ROS, «ROS Wiki RTAB Map Tuning,» 29 03 2017. [En línea]. Available: [http://wiki.ros.org/rtabmap\\_ros/Tutorials/Advanced%20Parameter%20Tuning](http://wiki.ros.org/rtabmap_ros/Tutorials/Advanced%20Parameter%20Tuning).

[33] ROS, «ROS Wiki PID,» 01 05 2017. [En línea]. Available: <http://wiki.ros.org/pid>.

[34] J. M. Rocha Núñez y E. L. Hernández, «Respuesta en el tiempo de un Sistema de Control».

[35] J. M. Rocha Núñez y E. L. Hernández, «Respuesta en el tiempo de un Sistema de Control».

# ANEXO A

## 13.1 A.1 MODELO\_HERMESII.URDF

```

<?xml version="1.0"?>
<robot name="HermesII">
    <!-- Creamos la base -->
    <link name="base_link">
        <visual>
            <origin xyz="0 0 0.3" rpy="0 0 0"/>
            <geometry>
                <mesh filename="package://hermesIII/model/dae/Base.dae"/>
            </geometry>
        </visual>
        <collision>
            <origin xyz="0 0 0.3" rpy="0 0 0"/>
            <geometry>
                <mesh filename="package://hermesIII/model/dae/Base.dae"/>
            </geometry>
        </collision>
        <inertial>
            <mass value="1.0"/>
            <inertia ixx="0.015" iyy="0.0375" izz="0.0375"
                     ixy="0" ixz="0" iyz="0"/>
        </inertial>
    </link>

    <!-- Creamos las ruedas -->
    <link name="wheel_front_link">
        <visual>
            <origin xyz="0 0 0" />
            <geometry>
                <mesh filename="package://hermesIII/model/dae/wheel.dae"/>
            </geometry>
        </visual>
        <collision>
            <origin xyz="0 0 0"/>
            <geometry>
                <mesh filename="package://hermesIII/model/dae/wheel.dae"/>
            </geometry>
        </collision>
        <inertial>
            <mass value="0.1"/>
            <inertia ixx="5.1458e-5" iyy="5.1458e-5" izz="6.125e-5"
                     ixy="0" ixz="0" iyz="0"/>
        </inertial>
    </link>

    ...
    ...
    ...

    <!-- Creamos cilindros rueda front -->
    <link name="cylinderN_wheel_front_link">

```

```

<visual>
    <origin xyz="0 0 0" />
    <geometry>
        <mesh filename="package://hermesIII/model/dae/Cilindro.dae"/>
    </geometry>
</visual>
<collision>
    <origin xyz="0 0 0"/>
    <geometry>
        <mesh filename="package://hermesIII/model/dae/Cilindro.dae"/>
    </geometry>
</collision>
<inertial>
    <mass value="0.0001"/>
    <inertia ixx="1.0583e-9" iyy="1.0583e-9" izz="4.5e-10"
    ixy="0" ixz="0" iyz="0"/>
</inertial>
</link>

...
...
...

<!-- Unimos la base con las ruedas -->
<joint name="base_to_front_wheel" type="continuous">
    <axis xyz="1 0 0"/> <!--Como elegimos continous, con axis decimos sobre que
eje gira-->
    <parent link="base_link"/>
    <child link="wheel_front_link"/>
    <origin xyz="0 -1.18 0.2" rpy="0 0 -1.57"/>
</joint>

...
...
...

<!-- UNIMOS LOS CILINDROS CON LA RUEDA FRONT -->
<joint name="cylinderN_to_front_wheel" type="continuous">
    <axis xyz="0 0 1"/> <!--Como elegimos continous, con axis decimos sobre que
eje gira-->
    <parent link="wheel_front_link"/>
    <child link="cylinderN_wheel_front_link"/>
    <origin xyz="0.065 0 0.235" rpy="-1.57 0 0"/>
</joint>

...
...
...

```

```

<!-- PARA GAZEBO -->
<gazebo>
    <plugin name="differential_drive_controller"
        filename="libgazebo_ros_diff_drive.so">
        <leftJoint>base_to_left_wheel</leftJoint>
        <rightJoint>base_to_right_wheel</rightJoint>
        <robotBaseFrame>base_link</robotBaseFrame>
        <wheelSeparation>0.4</wheelSeparation>
        <wheelDiameter>0.04</wheelDiameter>
        <publishWheelJointState>true</publishWheelJointState>
    </plugin>
</gazebo>
</robot>

```

Aclaración: El mismo se encuentra simplificado. Para su versión completa remitirse al repositorio [https://github.com/hmalatini/hermes3/blob/dev/src/hermesIII/model/urdf/modelo\\_HermesII.urdf](https://github.com/hmalatini/hermes3/blob/dev/src/hermesIII/model/urdf/modelo_HermesII.urdf)

## 13.2 A.2 DISPLAY.LAUNCH

```

<?xml version="1.0"?>
<launch>
    <arg name="model" default="$(find hermesIII)/model/urdf/modelo_HermesII.urdf" />
    <arg name="gui" default="True" />
    <param name="robot_description" textfile="$(arg model)" />
    <param name="use_gui" value="$(arg gui)" />
    <node name="joint_state_publisher" pkg="joint_state_publisher"
type="joint_state_publisher" />
        <node name="robot_state_publisher" pkg="robot_state_publisher" type="state_publisher"
/>
        <node name="rviz" pkg="rviz" type="rviz" args="$(find hermesIII)/rviz/default.rviz"
required="true" />
</launch>

```

## 13.3 A.3 CONFIG.RVIZ

Panels:

- Class: rviz/Displays  
Help Height: 78  
Name: Displays  
Property Tree Widget:  
Expanded:
  - /Global Options1
  - /Status1
Splitter Ratio: 0.5  
Tree Height: 463
- Class: rviz/Selection

```
Name: Selection
- Class: rviz/Tool Properties
    Expanded:
        - /2D Pose Estimate1
        - /2D Nav Goal1
        - /Publish Point1
    Name: Tool Properties
    Splitter Ratio: 0.588679
- Class: rviz/Views
    Expanded:
        - /Current View1
    Name: Views
    Splitter Ratio: 0.5
- Class: rviz/Time
    Experimental: false
    Name: Time
    SyncMode: 0
    SyncSource: ""
Visualization Manager:
    Class: ""
    Displays:
        - Alpha: 0.5
        Cell Size: 1
        Class: rviz/Grid
        Color: 160; 160; 164
        Enabled: true
        Line Style:
        Line Width: 0.03
        Value: Lines
        Name: Grid
        Normal Cell Count: 0
        Offset:
            X: 0
            Y: 0
            Z: 0
        Plane: XY
        Plane Cell Count: 10
        Reference Frame: <Fixed Frame>
        Value: true
        - Alpha: 1
        Class: rviz/RobotModel
        Collision Enabled: false
        Enabled: true
        Links:
```

```
All Links Enabled: true
Expand Joint Details: false
Expand Link Details: false
Expand Tree: false
Link Tree Style: Links in Alphabetic Order
base_link:
Alpha: 1
Show Axes: false
Show Trail: false
Value: true
Name: RobotModel
Robot Description: robot_description
TF Prefix: ""
Update Interval: 0
Value: true
Visual Enabled: true
Enabled: true
Global Options:
Background Color: 48; 48; 48
Fixed Frame: base_link
Frame Rate: 30
Name: root
Tools:
- Class: rviz/Interact
Hide Inactive Objects: true
- Class: rviz/MoveCamera
- Class: rviz>Select
- Class: rviz/FocusCamera
- Class: rviz/Measure
- Class: rviz/SetInitialPose
Topic: /initialpose
- Class: rviz/SetGoal
Topic: /move_base_simple/goal
- Class: rviz/PublishPoint
Single click: true
Topic: /clicked_point
Value: true
Views:
Current:
Class: rviz/Orbit
Distance: 10
Enable Stereo Rendering:
Stereo Eye Separation: 0.06
Stereo Focal Distance: 1
```





#### 13.4 A.4 GAZEBO.LAUNCH

```
<?xml version="1.0"?>
<launch>
    <!-- Load the Hermes3 URDF model into the parameter server -->
    <param name="robot_description" textfile="$(find
hermesIII)/model/urdf/modelo_HermesII.urdf" />
    <!-- Start Gazebo with an empty world -->

    <arg name="world_file" default="$(env TURTLEBOT_GAZEBO_WORLD_FILE)"/>

    <include file="$(find gazebo_ros)/launch/empty_world.launch">
        <arg name="use_sim_time" value="true"/>
        <arg name="debug" value="false"/>
        <arg name="gui" value="true" />
        <arg name="world_name" value="$(arg world_file)"/>
    </include>
    <!-- Spawn a TortoiseBot in Gazebo, taking the description from the
parameter server -->
    <node name="spawn_urdf" pkg="gazebo_ros" type="spawn_model"
        args="-param robot_description -urdf -model hermesII" />
</launch>
```

#### 13.5 A.5 MODELO\_HERMESIII.URDF

```
<?xml version="1.0"?>
<robot name="HermesIII">
    <!-- Creamos la base -->
    <link name="base_link">
        <visual>
            <origin xyz="0 0 0.09" rpy="0 0 2.35"/>
            <geometry>
                <mesh scale="0.2 0.2 0.2"
filename="package://hermesIII/model/dae/Hermes3_2017.dae"/>
            </geometry>
        </visual>
    </link>
</robot>
```

## 13.6 A.6 DISPLAY\_HERMESIII.LAUNCH

```
<?xml version="1.0"?>
<launch>
  <arg name="model" default="$(find hermesIII)/model/urdf/modelo_HermesIII.urdf" />
  <param name="robot_description" textfile="$(arg model)" />
  <node name="rviz" pkg="rviz" type="rviz" args="$(find hermesIII)/rviz/default.rviz"
required="true" />
</launch>
```

## ANEXO B

### 13.7 B.1 ATMEGA128CAPTUREWITHI2C.INO

```
#include <Wire.h>
#include <avr/io.h>
#include <avr/interrupt.h>

unsigned int timerCapture1 = 0;
unsigned int result1 = 0;
boolean descartar1 = false;

unsigned int timerCapture3 = 0;
unsigned int result3 = 0;
boolean descartar3 = false;

void setup() {
    //Configuramos para que trabaje con fuente de clock externa (para capturador de
    eventos)
    TIFR = 0;
    TCCR1B = 0b11000100; //Noise Canceler Capture (bit 7), Rising edge (bit 6) and
    Prescaler 256 (bits 2:0)
    TIMSK = 0b00100100; //Habilitamos Input Interrupt Capture

    TCCR1A = 0;

    // Inicializamos contador en 0
    TCNT1 = 0;

    //Configuramos para que trabaje con fuente de clock externa (para capturador de
    eventos)
    ETIFR = 0;
    TCCR3B = 0b11000100; //Noise Canceler Capture (bit 7), Rising edge (bit 6) and
    Prescaler 256 (bits 2:0)
    ETIMSK = 0b00100100; //Habilitamos Input Interrupt Capture

    TCCR3A = 0;

    // Inicializamos contador en 0
    TCNT3 = 0;

    Wire.begin(9);           // join i2c bus with address #9
    Wire.onRequest(requestEvent); // register event
```

```

pinMode(44, OUTPUT);
sei();
}

ISR(TIMER1_CAPT_vect){
if (!descartar1){
    result1 = (ICR1 - timerCapture1); //result1 DEBERA SER MULTIPLICADO POR 32
(microsegundos escalon, minima resolucion) EN LA ARDUINO MEGA PARA ESTAR EN MICROSEGUNDOS
}
else{
    descartar1 = false;
}
timerCapture1 = ICR1;
}

ISR(TIMER1_OVF_vect){
digitalWrite(44, !digitalRead(44));
if (descartar1){
    result1 = 0;
}
descartar1 = true;
}

ISR(TIMER3_CAPT_vect){
if (!descartar3){
    result3 = (ICR3 - timerCapture3); //result3 DEBERA SER MULTIPLICADO POR 32
(microsegundos escalon, minima resolucion) EN LA ARDUINO MEGA PARA ESTAR EN MICROSEGUNDOS
}
else{
    descartar3 = false;
}
timerCapture3 = ICR3;
}

ISR(TIMER3_OVF_vect){
if (descartar3){
    result3 = 0;
}
descartar3 = true;
}

void loop() {
}

```

```

// function that executes whenever data is requested by master
// this function is registered as an event, see setup()
void requestEvent() { //Mandamos 4 bytes
    byte resultados[4] = {result1>>8,result1,result3>>8,result3};
    Wire.write(resultados, 4);
}

```

### 13.8 B.2 ARDUINO\_CONTROLLER\_HERMES3\_V2.5\_WITH\_ATMEGA128A.INO

```

#include <ros.h>
#include <std_msgs/UInt8.h>
#include <std_msgs/Int16.h>
#include <std_msgs/Float64.h>
#include <geometry_msgs/Twist.h>
#include <Wire.h>

#define ledPin 13

byte I2CH;
byte I2CL;

unsigned long resultadoMotor1;
unsigned long resultadoMotor2;
unsigned long resultadoMotor3;
unsigned long resultadoMotor4;

ros::NodeHandle nh;

//Pines del motor 4
int IN3Motor4Controlador2 = 22;      // Input3 conectada al pin 5
int IN4Motor4Controlador2 = 23;      // Input4 conectada al pin 4
int ENBControlador2 = 2;      // ENBControlador2 conectada al pin 2 de Arduino

//Pines del motor 3
int IN1Motor4Controlador2 = 24;      // Input3 conectada al pin 5
int IN2Motor4Controlador2 = 25;      // Input4 conectada al pin 4
int ENAControlador2 = 3;      // ENAControlador2 conectada al pin 3 de Arduino

//Pines del motor 2
int IN3Motor4Controlador1 = 26;      // Input3 conectada al pin 5
int IN4Motor4Controlador1 = 27;      // Input4 conectada al pin 4

```

```

int ENBControlador1 = 4;      // ENBControlador1 conectada al pin 4 de Arduino

//Pines del motor 1
int IN1Motor4Controlador1 = 28;    // Input3 conectada al pin 5
int IN2Motor4Controlador1 = 29;    // Input4 conectada al pin 4
int ENAControlador1 = 5;      // ENAControlador1 conectada al pin 5 de Arduino

//Predeterminadamente deberian arrancar moviendose hacia adelante
bool haciaAdelante = true;
bool sentidoHorarioM1 = true;
bool sentidoHorarioM2 = true;
bool sentidoHorarioM3 = true;
bool sentidoHorarioM4 = true;
float divisorX = 0.5;

float PWM_LeftWheel;
float PWM_RightWheel;
float PWM_BackWheel;
float PWM_FrontWheel;

float vel1Anterior = 0;
float vel2Anterior = 0;
float vel3Anterior = 0;
float vel4Anterior = 0;

std_msgs::Float64 vel;
std_msgs::Int16 PWM;
ros::Publisher state1("/right_wheel/state", &vel);
ros::Publisher state2("/back_wheel/state", &vel);
ros::Publisher state3("/left_wheel/state", &vel);
ros::Publisher state4("/front_wheel/state", &vel);

ros::Publisher PWM_Left_Value("/PWM_LeftValue", &PWM);
ros::Publisher PWM_Right_Value("/PWM_RightValue", &PWM);
ros::Publisher PWM_Back_Value("/PWM_BackValue", &PWM);
ros::Publisher PWM_Front_Value("/PWM_FrontValue", &PWM);

void adjustLeftPWM( byte PWM_value){
    analogWrite(ENAControlador2,PWM_value); //Motor M3
}
void adjustRightPWM( byte PWM_value){
    analogWrite(ENAControlador1, PWM_value); //Motor M1
}
void adjustFrontPWM( byte PWM_value){

```

```

analogWrite(ENBControlador2,PWM_value);//Motor M4
}

void adjustBackPWM( byte PWM_value){
    analogWrite(ENBControlador1,PWM_value);//Motor M2
}

void setSentidoGiro(int cualMotor, bool sentidoHorario){
    switch(cualMotor){
        case 1: //Motor M1
            if(sentidoHorario){
                digitalWrite (IN2Motor4Controlador1, LOW) ;
                digitalWrite (IN1Motor4Controlador1, HIGH);
            } else {
                digitalWrite (IN1Motor4Controlador1, LOW);
                digitalWrite (IN2Motor4Controlador1, HIGH);
            }
            break;
        case 2: //Motor M2
            if(sentidoHorario){
                digitalWrite (IN4Motor4Controlador1, LOW);
                digitalWrite (IN3Motor4Controlador1, HIGH);
            } else {
                digitalWrite (IN3Motor4Controlador1, LOW);
                digitalWrite (IN4Motor4Controlador1, HIGH);
            }
            break;
        case 3: //Motor M3
            if(sentidoHorario){
                digitalWrite (IN2Motor4Controlador2, LOW);
                digitalWrite (IN1Motor4Controlador2, HIGH);
            } else {
                digitalWrite (IN1Motor4Controlador2, LOW);
                digitalWrite (IN2Motor4Controlador2, HIGH);
            }
            break;
        case 4: //Motor M4
            if(sentidoHorario){
                digitalWrite (IN4Motor4Controlador2, LOW);
                digitalWrite (IN3Motor4Controlador2, HIGH);
            } else {
                digitalWrite (IN3Motor4Controlador2, LOW);
                digitalWrite (IN4Motor4Controlador2, HIGH);
            }
            break;
    }
}

```

```

        }

}

void publicarVelocidades( const std_msgs::UInt8& publicarVelocidades)
{
    /*----- LEEMOS LAS VELOCIDADES DE LOS DOS PRIMEROS MOTORES -----*/
    Wire.requestFrom(8, 4);      // request 4 bytes from slave device #8
    while (Wire.available()) { // slave may send less than requested
        I2CH = Wire.read(); // receive a byte
        I2CL = Wire.read(); // receive a byte
        resultadoMotor3 = (I2CH<<8) + I2CL; //LEFT MOTOR

        I2CH = Wire.read(); // receive a byte
        I2CL = Wire.read(); // receive a byte
        resultadoMotor4 = (I2CH<<8) + I2CL; //FRONT MOTOR
    }

    /*----- LEEMOS LAS VELOCIDADES DE LOS OTROS DOS MOTORES -----*/
    Wire.requestFrom(9, 4);      // request 4 bytes from slave device #9
    while (Wire.available()) { // slave may send less than requested
        I2CH = Wire.read(); // receive a byte
        I2CL = Wire.read(); // receive a byte
        resultadoMotor1 = (I2CH<<8) + I2CL; //RIGHT MOTOR

        I2CH = Wire.read(); // receive a byte
        I2CL = Wire.read(); // receive a byte
        resultadoMotor2 = (I2CH<<8) + I2CL; //BACK MOTOR
    }

    /*----- CONVERTIMOS LAS VELOCIDADES EN METROS/SEGUNDOS -----*/
    float vel1 = 0;
    if(resultadoMotor1 != 0){
        vel1 = (0.0078*1000000)/(resultadoMotor1*32);
    }
    float vel2 = 0;
    if(resultadoMotor2 != 0){
        vel2 = (0.0078*1000000)/(resultadoMotor2*32);
    }
    float vel3 = 0;
    if(resultadoMotor3 != 0){
        vel3 = (0.0078*1000000)/(resultadoMotor3*32);
    }
}

```

```

float vel4 = 0;
if(resultadoMotor4 != 0){
    vel4 = (0.0078*1000000)/(resultadoMotor4*32);
}

/* ----- FILTRAMOS DATOS ATIPICOS (VELOCIDADES MAYORES A 0.9) -----
   */
/* ----- Y PUBLICAMOS LAS VELOCIDADES -----
   */

if (vel1 > 0.9){ //Comprobacion que se realiza por los datos erroneos que suelen
obtenerse... La rueda por cuestiones fisicas no puede ir a mas de 0.9 m/s
    vel1 = vel1Anterior;
}
else{
    vel1Anterior = vel1;
}

if (PWM_RightWheel <= 10){ //Si el PWM de ésta rueda es un valor bajo, Arduino supone
que no se mueve y envia 0 (evitando la espera de los dos overflow en las Atmega)
    vel.data = 0;
}
else if (sentidoHorarioM1){
    vel.data = vel1;
}
else {
    vel.data = vel1 * -1;
}

state1.publish( &vel );
//-----
if (vel2 > 0.9){
    vel2 = vel2Anterior;
}
else{
    vel2Anterior = vel2;
}

if (PWM_BackWheel <= 10){ //Si el PWM de ésta rueda es un valor bajo, Arduino supone
que no se mueve y envia 0 (evitando la espera de los dos overflow en las Atmega)
    vel.data = 0;
}
else if (sentidoHorarioM2){
    vel.data = vel2;
}
else{

```

```

        vel.data = vel2 * -1;
    }
    state2.publish( &vel );
//-----
    if (vel3 > 0.9){
        vel3 = vel3Anterior;
    }
    else{
        vel3Anterior = vel3;
    }

    if (PWM_LeftWheel <= 10){ //Si el PWM de ésta rueda es un valor bajo, Arduino supone
que no se mueve y envia 0 (evitando la espera de los dos overflow en las Atmega)
        vel.data = 0;
    }
    else if (sentidoHorarioM3){
        vel.data = vel3;
    }
    else{
        vel.data = vel3 * -1;
    }
    state3.publish( &vel );
//-----
    if (vel4 > 0.9){
        vel4 = vel4Anterior;
    }
    else{
        vel4Anterior = vel4;
    }

    if (PWM_FrontWheel <= 10){ //Si el PWM de ésta rueda es un valor bajo, Arduino supone
que no se mueve y envia 0 (evitando la espera de los dos overflow en las Atmega)
        vel.data = 0;
    }
    else if (sentidoHorarioM4){
        vel.data = vel4;
    }
    else{
        vel.data = vel4 * -1;
    }
    state4.publish( &vel );
}
//-----

```

```

void setLeftPWM( const std_msgs::Float64& leftPWM){ // "leftPWM" es un valor entre -1 y 1
y define la intensidad del ajuste a realizar
    float pwm = PWM_LeftWheel + leftPWM.data;
    PWM_LeftWheel = pwm;
    if(pwm < 0){

        if (sentidoHorarioM3){// Si se esta moviendo en sentido horario debemos cambiar su
        sentido de giro

            sentidoHorarioM3 = false; //Seteamos el sentido antihorario
            setSentidoGiro(3, sentidoHorarioM3); //Lo hacemos girar en sentido antihorario
        }

        if(pwm < -80){

            PWM_LeftWheel = -80;
            pwm = -80;
        }

        pwm = pwm * -1; //Hacemos que "pwm" sea positivo. Porque siempre lo debe ser
    }
    else{

        if (!sentidoHorarioM3){// Si se esta moviendo en sentido antihorario debemos cambiar
        su sentido de giro

            sentidoHorarioM3 = true; //Seteamos el sentido horario
            setSentidoGiro(3, sentidoHorarioM3); //Lo hacemos girar en sentido horario
        }

        if(pwm > 80){

            PWM_LeftWheel = 80;
            pwm = 80;
        }

    }

    adjustLeftPWM(pwm);
    //Publicamos el valor de PWM que desea enviar la placa
    PWM.data = PWM_LeftWheel;
    PWM_Left_Value.publish( &PWM );
}

void setFrontPWM( const std_msgs::Float64& frontPWM){

    float pwm = PWM_FrontWheel + frontPWM.data;
    PWM_FrontWheel = pwm;
    if(pwm < 0){

        if (sentidoHorarioM4){// Si se esta moviendo en sentido horario debemos cambiar su
        sentido de giro

            sentidoHorarioM4 = false; //Seteamos el sentido antihorario
            setSentidoGiro(4, sentidoHorarioM4); //Lo hacemos girar en sentido antihorario
        }

        if(pwm < -80){

            PWM_FrontWheel = -80;
        }

    }

}

```

```

        pwm = -80;
    }
    pwm = pwm * -1; //Hacemos que "pwm" sea positivo. Porque siempre lo debe ser
}
else{
    if (!sentidoHorarioM4){// Si se esta moviendo en sentido antihorario debemos cambiar
    su sentido de giro
        sentidoHorarioM4 = true; //Seteamos el sentido horario
        setSentidoGiro(4, sentidoHorarioM4); //Lo hacemos girar en sentido horario
    }
    if(pwm > 80){
        PWM_FrontWheel = 80;
        pwm = 80;
    }
}

adjustFrontPWM(pwm);

//Publicamos el valor de PWM que desea enviar la placa
PWM.data = PWM_FrontWheel;
PWM_Front_Value.publish( &PWM );
}

void setRightPWM( const std_msgs::Float64& rightPWM){
    float pwm = PWM_RightWheel + rightPWM.data;
    PWM_RightWheel = pwm;
    if(pwm < 0){
        if (sentidoHorarioM1){// Si se esta moviendo en sentido horario debemos cambiar su
        sentido de giro
            sentidoHorarioM1 = false; //Seteamos el sentido antihorario
            setSentidoGiro(1, sentidoHorarioM1); //Lo hacemos girar en sentido antihorario
        }
        if(pwm < -80){
            PWM_RightWheel = -80;
            pwm = -80;
        }
        pwm = pwm * -1; //Hacemos que "pwm" sea positivo. Porque siempre lo debe ser
    }
    else{
        if (!sentidoHorarioM1){// Si se esta moviendo en sentido antihorario debemos cambiar
        su sentido de giro
            sentidoHorarioM1 = true; //Seteamos el sentido horario
            setSentidoGiro(1, sentidoHorarioM1); //Lo hacemos girar en sentido horario
        }
        if(pwm > 80){
            PWM_RightWheel = 80;
        }
    }
}

```

```

        pwm = 80;
    }
}

adjustRightPWM(pwm);

//Publicamos el valor de PWM que desea enviar la placa
PWM.data = PWM_RightWheel;
PWM_Right_Value.publish( &PWM );
}

void setBackPWM( const std_msgs::Float64& backPWM){
    float pwm = PWM_BackWheel + backPWM.data;
    PWM_BackWheel = pwm;
    if(pwm < 0){

        if (sentidoHorarioM2){// Si se esta moviendo en sentido horario debemos cambiar su
        sentido de giro
            sentidoHorarioM2 = false; //Seteamos el sentido antihorario
            setSentidoGiro(2, sentidoHorarioM2); //Lo hacemos girar en sentido antihorario
        }
        if(pwm < -80){

            PWM_BackWheel = -80;
            pwm = -80;
        }
        pwm = pwm * -1; //Hacemos que "pwm" sea positivo. Porque siempre lo debe ser
    }
    else{

        if (!sentidoHorarioM2){// Si se esta moviendo en sentido antihorario debemos cambiar
        su sentido de giro
            sentidoHorarioM2 = true; //Seteamos el sentido horario
            setSentidoGiro(2, sentidoHorarioM2); //Lo hacemos girar en sentido horario
        }
        if(pwm > 80){

            PWM_BackWheel = 80;
            pwm = 80;
        }
    }

    adjustBackPWM(pwm);

    //Publicamos el valor de PWM que desea enviar la placa
    PWM.data = PWM_BackWheel;
    PWM_Back_Value.publish( &PWM );
}

```

```

ros::Subscriber<std_msgs::UInt8> subVelRequest("/send_velocity", &publicarVelocidades );

ros::Subscriber<std_msgs::Float64> subLeftControlEffort("/left_wheel/control_effort",
&setLeftPWM );
ros::Subscriber<std_msgs::Float64> subFrontControlEffort("/front_wheel/control_effort",
&setFrontPWM );
ros::Subscriber<std_msgs::Float64> subRightControlEffort("/right_wheel/control_effort",
&setRightPWM );
ros::Subscriber<std_msgs::Float64> subBackControlEffort("/back_wheel/control_effort",
&setBackPWM );

void setup()
{
    pinMode(ledPin, OUTPUT);
    Wire.begin();           // join i2c bus (address optional for master)

    //PWM pins config
    pinMode (ENBControlador2, OUTPUT);
    pinMode (IN3Motor4Controlador2, OUTPUT);
    pinMode (IN4Motor4Controlador2, OUTPUT);

    pinMode (ENAControlador2, OUTPUT);
    pinMode (IN1Motor4Controlador2, OUTPUT);
    pinMode (IN2Motor4Controlador2, OUTPUT);

    pinMode (ENBControlador1, OUTPUT);
    pinMode (IN3Motor4Controlador1, OUTPUT);
    pinMode (IN4Motor4Controlador1, OUTPUT);

    pinMode (ENAControlador1, OUTPUT);
    pinMode (IN1Motor4Controlador1, OUTPUT);
    pinMode (IN2Motor4Controlador1, OUTPUT);

//-----
//Preparamos la salida para que el robot este seteado para mover todos sus motores en
//sentido horario
//MOTORES M3 y M4 giran en sentido horario
    digitalWrite (IN3Motor4Controlador2, HIGH);
    digitalWrite (IN4Motor4Controlador2, LOW);

    digitalWrite (IN1Motor4Controlador2, HIGH);
    digitalWrite (IN2Motor4Controlador2, LOW);

//MOTORES M1 y M2 giran en sentido horario
    digitalWrite (IN3Motor4Controlador1, HIGH);

```

```
digitalWrite (IN4Motor4Controlador1, LOW);

digitalWrite (IN1Motor4Controlador1, HIGH);
digitalWrite (IN2Motor4Controlador1, LOW);

pinMode(13, OUTPUT);
nh.initNode();
nh.subscribe(subVelRequest);

nh.subscribe(subLeftControlEffort);
nh.subscribe(subRightControlEffort);
nh.subscribe(subFrontControlEffort);
nh.subscribe(subBackControlEffort);

nh.advertise(state1);
nh.advertise(state2);
nh.advertise(state3);
nh.advertise(state4);

nh.advertise(PWM_Left_Value);
nh.advertise(PWM_Right_Value);
nh.advertise(PWM_Back_Value);
nh.advertise(PWM_Front_Value);
}

void loop()
{
    nh.spinOnce();
}
```

## ANEXO C

### 13.9 C.1 RTABMAP\_ODOMETRY.LAUNCH

```
<launch>
  <group ns="rtabmap">

    <!-- Odometry -->
    <node pkg="rtabmap_ros" type="rgbd_odometry" name="rgbd_odometry" >
      <remap from="rgb/image"          to="/camera/rgb/image_rect_color"/>
      <remap from="depth/image"        to="/camera/depth_registered/image_raw"/>
      <remap from="rgb/camera_info"   to="/camera/depth_registered/camera_info"/>

      <param name="frame_id" type="string" value="base_link"/>

      <param name="Odom/ResetCountdown" type="string" value="1" />
      <!-- 0=Frame-to-Map (F2M) 1=Frame-to-Frame (F2F) -->
      <param name="Odom/Strategy" value="1"/>
      <!-- Correspondences: 0=Features Matching, 1=Optical Flow -->
      <param name="Vis/CorType" value="0"/>
      <param name="GFTT/MinDistance" type="string" value="5"/> <!-- default 5 pixels -->

      <param name="Reg/Force3DoF"     value="true" />
    </node>

    <node name="rtabmap" pkg="rtabmap_ros" type="rtabmap" args="--delete_db_on_start">
      <param name="frame_id" type="string" value="base_link"/>
      <param name="subscribe_depth" type="bool" value="true"/>

      <remap from="odom" to="odom"/>
      <remap from="rgb/image" to="/camera/rgb/image_rect_color"/>
      <remap from="depth/image" to="/camera/depth_registered/image_raw"/>
      <remap from="rgb/camera_info" to="/camera/rgb/camera_info"/>

      <param name="queue_size" type="int" value="10"/>

      <!-- RTAB-Map's parameters -->
      <param name="RGBD/AngularUpdate" type="string" value="0.01"/>
      <param name="RGBD/LinearUpdate" type="string" value="0.01"/>
      <param name="Rtabmap/TimeThr" type="string" value="500"/>
      <param name="Mem/RehearsalSimilarity" type="string" value="0.45"/>
      <param name="RGBD/OptimizeFromGraphEnd" type="string" value="true"/>

      <param name="subscribe_scan" type="string" value="true"/>
      <param name="Rtabmap/StartNewMapOnLoopClosure" value="true"/>

      <param name="Reg/Force3DoF"     value="true" />
      <param name="Optimizer/Slam2D" value="true" />

      <param name="cloud_noise_filtering_radius" value="0.05"/>
      <param name="cloud_noise_filtering_min_neighbors" value="2"/>
    </node>
  </group>
</launch>
```

### 13.10C.2 SLAM\_RTABMAP.LAUNCH

```
<launch>
  <include file="$(find freenect_launch)/launch/freenect.launch">
    <!-- use device registration -->
    <arg name="depth_registration" value="true" />
  </include>

  <node name="depthimage_to_laserscan" pkg="depthimage_to_laserscan"
type="depthimage_to_laserscan" >
    <remap from="image" to="/camera/depth_registered/image_raw"/>
    <remap from="camera_info" to="/camera/depth_registered/camera_info"/>
  </node>

  <node pkg="tf" type="static_transform_publisher" name="base_to_camera" args="0.0 0.0
0.0 0.0 0.0 0.0 /base_link /camera_link 100" />

  <include file="$(find hermesIII)/launch/rtabmap_odometry.launch"></include>
</launch>
```

### 13.11C.3 REMOTE\_VISUALIZATION.LAUNCH

```
<launch>
  <node pkg="rviz" type="rviz" name="rviz" args="-d $(find
hermesIII)/rviz/remote_visualization.rviz">
  </node>
</launch>
```

### 13.12C.4 REMOTE\_VISUALIZATION.RVIZ

```
En repositorio
https://github.com/hmalatini/hermes3/blob/dev/src/hermesIII/rviz/remote\_visualization.rvi
z
```

## ANEXO D

### 13.13D.1 ARDUINO\_ROS.SH

```
#!/bin/bash
rosrun rosserial_python serial_node.py /dev/serial/by-id/usb-
Arduino_www.arduino.cc_0042_5533834353935110A0B0-if00 _baud:=115200
```

### 13.14D.2 PID.LAUNCH

```
<launch>
  <node name="controller" pkg="pid" type="controller" ns="left_wheel" >
    <param name="node_name" value="left_wheel_pid" />
    <remap from="pid_enable" to="left_wheel_pid_activate" />
    <param name="Kp" value="40.0" />
    <param name="Ki" value="0" />
    <param name="Kd" value="5" />
    <param name="upper_limit" value="60" />
    <param name="lower_limit" value="-60" />
    <param name="windup_limit" value="60" />
    <param name="diagnostic_period" value="0.25" />
    <param name="max_loop_frequency" value="10.0" />
    <param name="min_loop_frequency" value="10.0" />
    <param name="min_loop_frequency" value="10.0" />
    <remap from="setpoint" to="/setpoint_left" />
  </node>

  <node name="controller" pkg="pid" type="controller" ns="right_wheel" >
    <param name="node_name" value="right_wheel_pid" />
    <remap from="pid_enable" to="right_wheel_pid_activate" />
    <param name="Kp" value="40.0" />
    <param name="Ki" value="0" />
    <param name="Kd" value="5" />
    <param name="upper_limit" value="60" />
    <param name="lower_limit" value="-60" />
    <param name="windup_limit" value="60" />
    <param name="diagnostic_period" value="0.25" />
    <param name="max_loop_frequency" value="10.0" />
    <param name="min_loop_frequency" value="10.0" />
    <remap from="setpoint" to="/setpoint_right" />
  </node>

  <node name="controller" pkg="pid" type="controller" ns="front_wheel" >
    <param name="node_name" value="front_wheel_pid" />
    <remap from="pid_enable" to="front_wheel_pid_activate" />
    <param name="Kp" value="40.0" />
    <param name="Ki" value="0" />
    <param name="Kd" value="5" />
    <param name="upper_limit" value="60" />
    <param name="lower_limit" value="-60" />
    <param name="windup_limit" value="60" />
    <param name="diagnostic_period" value="0.25" />
    <param name="max_loop_frequency" value="10.0" />
    <param name="min_loop_frequency" value="10.0" />
    <remap from="setpoint" to="/setpoint_front" />
  </node>
```

```

<node name="controller" pkg="pid" type="controller" ns="back_wheel"    >
  <param name="node_name" value="back_wheel_pid" />
  <remap from="pid_enable" to="back_wheel_pid_activate" />
  <param name="Kp" value="40.0" />
  <param name="Ki" value="0" />
  <param name="Kd" value="5" />
  <param name="upper_limit" value="60" />
  <param name="lower_limit" value="-60" />
  <param name="windup_limit" value="60" />
  <param name="diagnostic_period" value="0.25" />
  <param name="max_loop_frequency" value="10.0" />
  <param name="min_loop_frequency" value="10.0" />
  <remap from="setpoint" to="/setpoint_back" />
</node>

<node pkg="hermesIII" type="send_velocity.py" name="send_velocity"/>

<node pkg="hermesIII" type="PID_general.py" name="PID_general"/>

<node pkg="hermesIII" type="fix_integralError.py" name="fix_integralError"/>

<!-- rqt_plot is a resource hog, so if you're seeing high CPU usage, don't launch
rqt_plot -->
<!--<node name="rqt_plot" pkg="rqt_plot" type="rqt_plot"
args="/left_wheel/control_effort/data /left_wheel/state/data /setpoint_left/data
/right_wheel/control_effort/data /right_wheel/state/data /setpoint_right/data
/front_wheel/control_effort/data /front_wheel/state/data /setpoint_front/data
/back_wheel/control_effort/data /back_wheel/state/data /setpoint_back/data" /> -->

<!-- <node name="rqt_robot_monitor" pkg="rqt_robot_monitor" type="rqt_robot_monitor"
/> -->
<!-- <node name="rqt_reconfigure" pkg="rqt_reconfigure" type="rqt_reconfigure" /> -->
</launch>
```

### 13.15D.3 SEND\_VELOCITY.PY

```

#!/usr/bin/env python
# license removed for brevity
import rospy
from std_msgs.msg import UInt8

def talker():
    pub = rospy.Publisher('send_velocity', UInt8, queue_size=10)
    rospy.init_node('velocity_request', anonymous=True)
    rate = rospy.Rate(30) # 10hz
    while not rospy.is_shutdown():
        pub.publish(1)
        rate.sleep()

if __name__ == '__main__':
    try:
        talker()
    except rospy.ROSInterruptException:
        pass
```

#### 13.16D.4 PID\_GENERAL.PY

```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist
from std_msgs.msg import Float64
from std_msgs.msg import Bool
import math

pubLeft = rospy.Publisher('/setpoint_left', Float64, queue_size=10)
pubRigth = rospy.Publisher('/setpoint_right', Float64, queue_size=10)
pubFront = rospy.Publisher('/setpoint_front', Float64, queue_size=10)
pubBack = rospy.Publisher('/setpoint_back', Float64, queue_size=10)

pubEnableLeft = rospy.Publisher('/left_wheel/left_wheel_pid_activate', Bool,
queue_size=10)
pubEnableRigth = rospy.Publisher('/right_wheel/right_wheel_pid_activate', Bool,
queue_size=10)
pubEnableFront = rospy.Publisher('/front_wheel/front_wheel_pid_activate', Bool,
queue_size=10)
pubEnableBack = rospy.Publisher('/back_wheel/back_wheel_pid_activate', Bool,
queue_size=10)

velMinima = 0.3

def setDesiredVel(data):
    #print "Me llego mensaje Twist:\n      Lineal:\n          X: ",data.linear.x,"\\n
    Y: ",data.linear.y,"\\n          Z: ",data.linear.z,"\\n      ANGULAR:\n          X:
    ",data.angular.x,"\\n          Y: ",data.angular.y,"\\n          Z: ",data.angular.z
    #ANALIZAR EL CASO EN EL QUE EL MENSAJE EN X ES IGUAL AL Y, ENTONCES DOS RUEDAS SE
    ANULAN Y FUNCIONAN SOLO DOS...
    radioRobot = 0.45
    vectorDeseado = [[data.linear.x],[data.linear.y],[data.angular.z * radioRobot]]
    matrizMotores = [[-1*math.sin(0.785398),math.cos(0.785398),1],[-1*
    math.sin(2.35619),math.cos(2.35619),1],[-1*math.sin(3.92699),math.cos(3.92699),1],[-1*
    math.sin(5.49779),math.cos(5.49779),1]]
    velMotores=[[0],[0],[0],[0]]
    for columna in range(0,len(vectorDeseado)):
        for fila in range(0,len(matrizMotores)):
            velMotores[fila][0] += matrizMotores[fila][columna] *
    vectorDeseado[columna][0]

    menor = False

    if(not menor):
        pubEnableLeft.publish(True)
        pubEnableRigth.publish(True)
        pubEnableFront.publish(True)
        pubEnableBack.publish(True)
        pubLeft.publish(velMotores[0][0])
        pubFront.publish(velMotores[1][0])
        pubRigth.publish(velMotores[2][0])
        pubBack.publish(velMotores[3][0])
    else:
        pubLeft.publish(0)
        pubFront.publish(0)
        pubRigth.publish(0)
        pubBack.publish(0)
```

```

def listener():
    rospy.init_node('PID_General', anonymous=True)
    rospy.Subscriber("/cmd_vel", Twist, setDesiredVel)
    rospy.spin()

if __name__ == '__main__':
    try:
        listener()
    except rospy.ROSInterruptException:
        pass

```

### 13.17D.5 FIX\_INTEGRALERROR.PY

```

#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist
from std_msgs.msg import Float64
from std_msgs.msg import Bool

pubEnableLeft = rospy.Publisher('/left_wheel/left_wheel_pid_activate', Bool,
queue_size=10)
pubEnableRight = rospy.Publisher('/right_wheel/right_wheel_pid_activate', Bool,
queue_size=10)
pubEnableFront = rospy.Publisher('/front_wheel/front_wheel_pid_activate', Bool,
queue_size=10)
pubEnableBack = rospy.Publisher('/back_wheel/back_wheel_pid_activate', Bool,
queue_size=10)

contadorLeft = 0
contadorRight = 0
contadorFront = 0
contadorBack = 0

oldEffortLeft = 0
oldEffortRight = 0
oldEffortFront = 0
oldEffortBack = 0

def comprobarEffortLeft(value):
    global oldEffortLeft, contadorLeft
    if( (abs(value.data - oldEffortLeft) <= 0.001) and (abs(oldEffortLeft) < 1) ):
        contadorLeft += 1
    else:
        oldEffortLeft = value.data
        contadorLeft = 0

    if(contadorLeft >= 3):
        pubEnableLeft.publish(False)

def comprobarEffortRight(value):
    global oldEffortRight, contadorRight
    if( (abs(value.data - oldEffortRight) <= 0.001) and (abs(oldEffortRight) < 1) ):
        contadorRight += 1
    else:
        oldEffortRight = value.data
        contadorRight = 0

    if(contadorRight >= 3):

```

```

        pubEnableRight.publish(False)

def comprobarEffortFront(value):
    global oldEffortFront, contadorFront
    if( (abs(value.data - oldEffortFront) <= 0.001) and (abs(oldEffortFront) < 1) ):
        contadorFront += 1
    else:
        oldEffortFront = value.data
        contadorFront = 0

    if(contadorFront >= 3):
        pubEnableFront.publish(False)

def comprobarEffortBack(value):
    global oldEffortBack, contadorBack
    if( (abs(value.data - oldEffortBack) <= 0.001) and (abs(oldEffortBack) < 1) ):
        contadorBack += 1
    else:
        oldEffortBack = value.data
        contadorBack = 0

    if(contadorBack >= 3):
        pubEnableBack.publish(False)

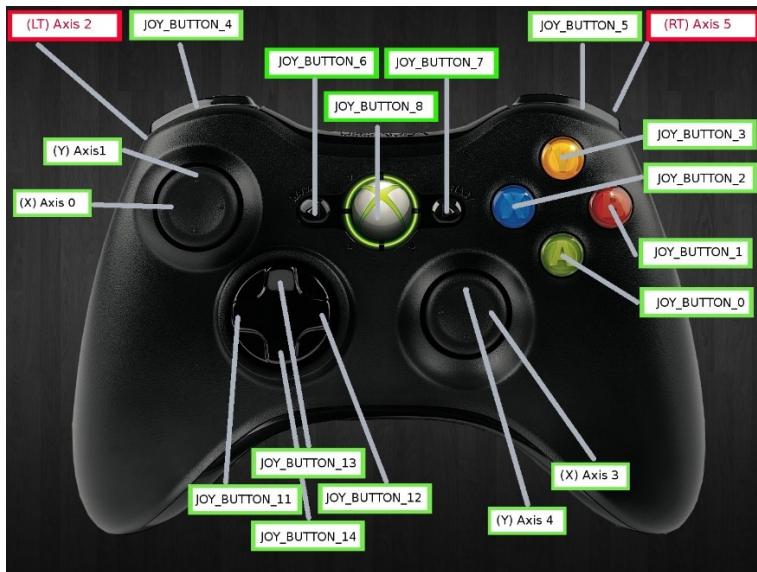
def listener():
    rospy.init_node('fix_integralError', anonymous=True)
    rospy.Subscriber("/left_wheel/control_effort", Float64, comprobarEffortLeft)
    rospy.Subscriber("/right_wheel/control_effort", Float64, comprobarEffortRight)
    rospy.Subscriber("/front_wheel/control_effort", Float64, comprobarEffortFront)
    rospy.Subscriber("/back_wheel/control_effort", Float64, comprobarEffortBack)
    rospy.spin()

if __name__ == '__main__':
    try:
        listener()
    except rospy.ROSInterruptException:
        pass

```

## ANEXO E

### 13.18E.1 XBOX360.PY



```
#!/usr/bin/env python
import rospy
from geometry_msgs.msg import Twist
from sensor_msgs.msg import Joy

toggle = False

def callback(data):
    global toggle
    twist = Twist()
    twist.linear.x = 1.5*data.axes[1]
    twist.linear.y = -1.5*data.axes[0]
    twist.angular.z = 1.5*data.axes[3]
    if(data.buttons[4] == 1):
        toggle = True
        pub.publish(twist)
    elif(toggle == True):
        twist.linear.x = 0
        twist.linear.y = 0
        twist.angular.z = 0
        pub.publish(twist)
        toggle = False

# Intializes everything
def start():
    # publishing to "turtle1/cmd_vel" to control turtle1
    global pub
    pub = rospy.Publisher('/cmd_vel', Twist, queue_size=10)
    # subscribed to joystick inputs on topic "joy"
    rospy.Subscriber("joy", Joy, callback)
    # starts the node
    rospy.init_node('Xbox360Joy')
    rospy.spin()

if __name__ == '__main__':
    start()
```

### 13.19E.2 XBOX360\_TELEOP.LAUNCH

```
<launch>
  <node pkg="joy" type="joy_node" name="joystick"/>
  <node pkg="hermesIII" type="xbox_360.py" name="xbox_360"/>
</launch>
```

### 13.20E.3 KEYBOARD\_TELEOP.LAUNCH

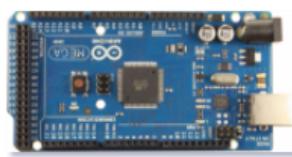
```
<launch>
  <!-- turtlebot_teleop_key already has its own built in velocity smoother -->
  <node pkg="turtlebot_teleop" type="turtlebot_teleop_key"
    name="turtlebot_teleop_keyboard" output="screen">
    <param name="scale_linear" value="0.8" type="double"/>
    <param name="scale_angular" value="1.5" type="double"/>
    <remap from="turtlebot_teleop_keyboard/cmd_vel" to="/cmd_vel"/>
  </node>
```

## ANEXO F

### 13.21F.1 GENERAL.LAUNCH

```
<launch>
  <include file="$(find hermesIII)/launch/slam_rtabmap.launch"></include>
  <include file="$(find hermesIII)/launch/pid.launch"></include>
  <include file="$(find hermesIII)/launch/xbox360_teleop.launch"></include>
  <include file="$(find hermesIII)/launch/display.launch"></include>
</launch>
```

### 13.22F.2 IMAGEN DE COMPONENTES



Arduino Mega 2560



NVIDIA Jetson TK1



Kinect V1



Receptor de Joystick  
Xbox 360



Sensores FC-03



Joystick Xbox 360



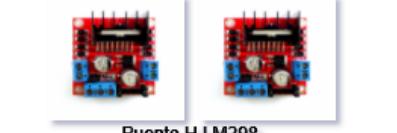
Microcontroladores  
Atmega 128a



Ruedas Omnidireccionales



Motores DC 3v-6v  
con Caja Reductora



Puente H LM298  
+ Salida Regulada 5v



Hub USB x4



Encoders



Bateria de 3 celdas  
2200mAh



Carcasa Hermes 3