



# INSTITUTO POLITÉCNICO NACIONAL

UNIDAD PROFESIONAL INTERDISCIPLINARIA EN  
INGENIERÍA Y TECNOLOGÍAS AVANZADAS

TRABAJO TERMINAL

## "COORDINACIÓN DE ROBOTS MÓVILES TERRESTRES EN CONFIGURACIÓN LÍDER-SEGUIDOR CON FORMACIÓN VARIABLE"

QUÉ PARA OBTENER EL TÍTULO DE:  
INGENIERO EN MECATRÓNICA

PRESENTAN LOS ALUMNOS:  
DIANA ESMERALDA HERNANDEZ JIMENEZ  
ENRIQUE ARTURO RODRÍGUEZ NAVA  
FRANCISCO JOAQUIN TRUJILLO OCAMPO

ASESOR:  
DR. JUAN LUIS MATA MACHUCA



JUNIO 2020

# Índice General

|   |             |
|---|-------------|
| <b>Índice de Figuras</b>                  | <b>IV</b>   |
| <b>Índice de Tablas</b>                   | <b>IX</b>   |
| <b>Lista de Abreviaturas</b>              | <b>X</b>    |
| <b>Lista de Símbolos</b>                  | <b>XI</b>   |
| <b>Resumen</b>                            | <b>XII</b>  |
| <b>Abstract</b>                           | <b>XIII</b> |
| <b>1. Introducción</b>                    | <b>1</b>    |
| 1.1. Planteamiento del Problema . . . . . | 3           |
| 1.2. Propuesta de Solución . . . . .      | 4           |
| 1.3. Justificación . . . . .              | 5           |
| 1.4. Objetivos . . . . .                  | 6           |
| 1.4.1. Objetivo General . . . . .         | 6           |
| 1.4.2. Objetivos Particulares . . . . .   | 6           |
| <b>2. Marco de Referencia</b>             | <b>7</b>    |
| 2.1. Marco Histórico . . . . .            | 8           |
| 2.1.1. Antecedentes . . . . .             | 8           |
| 2.1.2. Estado del Arte . . . . .          | 10          |
| 2.2. Marco Conceptual . . . . .           | 13          |
| 2.2.1. Robots Móviles . . . . .           | 13          |
| 2.2.2. Sistemas Multi-Robot . . . . .     | 15          |
| 2.2.3. Python . . . . .                   | 17          |

|  |            |
|--|------------|
| <b>ÍNDICE GENERAL</b>  |            |
| 2.2.4. Robot Operating System (ROS) . . . . .                  | 18         |
| 2.2.5. Gazebo . . . . .  | 23         |
| 2.3. Marco Teórico . . . . .                                   | 24         |
| 2.3.1. Representación de la Posición . . . . .                 | 24         |
| 2.3.2. Representación de la Orientación . . . . .              | 26         |
| 2.3.3. Transformaciones Homogéneas . . . . .                   | 32         |
| 2.3.4. Sistemas de Control . . . . .                           | 35         |
| 2.3.5. Controladores Clásicos . . . . .                        | 36         |
| 2.3.6. Lógica Difusa . . . . .                                 | 38         |
| 2.3.7. Modelo Cinemático del Robot Móvil Diferencial . . . . . | 44         |
| <b>3. Análisis y Diseño</b>                                    | <b>46</b>  |
| 3.1. Análisis de Especificaciones . . . . .                    | 47         |
| 3.2. División por Áreas Funcionales . . . . .                  | 49         |
| 3.3. Diseño Conceptual . . . . .                               | 52         |
| 3.3.1. Matriz Morfológica . . . . .                            | 52         |
| 3.3.2. Búsqueda Morfológica . . . . .                          | 52         |
| 3.3.3. Análisis de Ventajas y Desventajas . . . . .            | 53         |
| 3.3.4. Selección del Diseño Conceptual . . . . .               | 54         |
| 3.4. Diseño Detallado . . . . .                                | 57         |
| 3.4.1. Descripción de los Robots . . . . .                     | 57         |
| 3.4.2. Área funcional 1: Percepción . . . . .                  | 59         |
| 3.4.3. Área funcional 2: Procesamiento . . . . .               | 63         |
| 3.4.4. Área funcional 3: Movimiento . . . . .                  | 106        |
| 3.5. Integración de Áreas Funcionales . . . . .                | 108        |
| <b>4. Pruebas y Resultados</b>                                 | <b>110</b> |
| 4.1. Sistema de Localización . . . . .                         | 111        |
| 4.2. Sistema de Asignación de roles . . . . .                  | 113        |
| 4.3. Sistema de Navegación . . . . .                           | 115        |
| 4.4. Sistema de Formación . . . . .                            | 134        |
| <b>Conclusiones</b>  | <b>XIV</b> |
| <b>Referencias</b>   | <b>XVI</b> |

|  |            |
|--|------------|
| <b>Apéndices</b>                       | <b>XIX</b> |
| A. Códigos . . . . .                   | XX         |
| A.1. Sistema de Localización . . . . . | XX         |
| A.2. Sistema de Navegación . . . . .   | XXIII      |
| A.3. Sistema de Formación . . . . .    | XXXIX      |
| A.4. Funciones Auxiliares . . . . .    | LVI        |
| B. Planos . . . . .                    | LXI        |
| B.1. Escenario de pruebas . . . . .    | LXI        |

# Índice de Figuras

|  |           |
|--|-----------|
| <b>1. Introducción</b>   | <b>1</b>  |
| 1.1. Diagrama general de funcionamiento. . . . .                                     | 4         |
| <b>2. Marco de Referencia</b>  | <b>7</b>  |
| 2.1. Configuraciones de los RMR [6]. . . . .   | 14        |
| 2.2. Representación de un vector en coordenadas cartesianas en dos dimensiones [16]. | 24        |
| 2.3. Representación de un vector en coordenadas polares [16]. . . . .                | 25        |
| 2.4. Orientación de un sistema OUV respecto a otro OXY en un plano [16]. . . . .     | 26        |
| 2.5. Sistema de referencia OXYZ y solidario al objeto OUVW [16]. . . . .             | 27        |
| 2.6. Rotaciones del sistema OUVW con respecto al sistema de referencia OXYZ [16].    | 28        |
| 2.7. Sistema de control en lazo abierto [11]. . . . .                                | 35        |
| 2.8. Sistema de control en lazo cerrado [11]. . . . .                                | 35        |
| 2.9. Sistema difuso puro [21]. . . . .   | 38        |
| 2.10. Sistema difuso con fusificación y defusificación [21]. . . . .                 | 39        |
| 2.11. Unión de los conjuntos difusos A y B [22]. . . . .                             | 40        |
| 2.12. Intersección de los conjuntos difusos A y B [22]. . . . .                      | 40        |
| 2.13. Complemento del conjunto difuso A [22]. . . . .                                | 41        |
| 2.14. Función de membresía triangular [23]. . . . .                                  | 41        |
| 2.15. Función de membresía trapezoidal [23]. . . . .                                 | 42        |
| 2.16. Función de membresía gaussiana [23]. . . . .                                   | 42        |
| 2.17. Representación geométrica del robot [11]. . . . .                              | 44        |
| <b>3. Análisis y Diseño</b>  | <b>46</b> |
| 3.1. IDEF0 (Módulo A-0). . . . .   | 49        |
| 3.2. IDEF0 (Módulo A0). . . . .  | 50        |

|  |    |
|--|----|
| 3.3. IDEF0 (Módulo A2). . . . .  | 51 |
| 3.4. IDEF0 (Módulo A2.4). . . . .  | 51 |
| 3.5. Formaciones posibles con tres robots. . . . .   | 54 |
| 3.6. Trayectorias propuestas. . . . .  | 55 |
| 3.7. Diseño conceptual final. . . . .  | 56 |
| 3.8. Dimensiones TurtleBot3 Burger [26]. . . . .   | 58 |
| 3.9. Diagrama área funcional percepción. . . . .   | 59 |
| 3.10. Sensor LDS-01 [27]. . . . .  | 60 |
| 3.11. OpenCR1.0 [28]. . . . .  | 61 |
| 3.12. Diagrama área funcional Procesamiento. . . . .   | 63 |
| 3.13. Raspberry Pi 3 Model B+ [29]. . . . .  | 63 |
| 3.14. Diagrama sistema de localización. . . . .  | 65 |
| 3.15. Delimitación de posiciones iniciales de los robots. . . . .                            | 65 |
| 3.16. Diagrama de flujo del sistema de localización. . . . .                                 | 66 |
| 3.17. Diagrama de flujo de la función “scan_callback_tb”. . . . .                            | 67 |
| 3.18. Diagrama sistema de asignación de roles. . . . .                                       | 68 |
| 3.19. Diagrama de flujo del sistema de asignación de roles. . . . .                          | 69 |
| 3.20. Esquema de planeación de movimiento para un robot móvil [31]. . . . .                  | 70 |
| 3.21. Parámetros considerados para el control del seguimiento de trayectoria [30]. . . . .   | 72 |
| 3.22. Diagrama de bloques para el seguimiento de trayectorias (Control P). . . . .           | 73 |
| 3.23. Trayectoria seguida de (0,0) a (1,1) para $K_v = 1$ y $K_w = 1$ (Control P). . . . .   | 73 |
| 3.24. Error en la velocidad angular para $K_v = 1$ y $K_w = 1$ (Control P). . . . .          | 74 |
| 3.25. Error en la velocidad lineal para $K_v = 1$ y $K_w = 1$ (Control P). . . . .           | 74 |
| 3.26. Trayectoria seguida de (0,0) a (1,1) para $K_v = 5$ y $K_w = 5$ (Control P). . . . .   | 75 |
| 3.27. Error en la velocidad angular para $K_v = 5$ y $K_w = 5$ (Control P). . . . .          | 75 |
| 3.28. Error en la velocidad lineal para $K_v = 5$ y $K_w = 5$ (Control P). . . . .           | 75 |
| 3.29. Trayectoria seguida de (0,0) a (1,1) para $K_v = 10$ y $K_w = 3$ (Control P). . . . .  | 76 |
| 3.30. Error en la velocidad angular para $K_v = 10$ y $K_w = 3$ (Control P). . . . .         | 76 |
| 3.31. Error en la velocidad lineal para $K_v = 10$ y $K_w = 3$ (Control P). . . . .          | 76 |
| 3.32. Trayectoria seguida de (-1,1) a (1,1) para $K_v = 10$ y $K_w = 3$ (Control P). . . . . | 77 |
| 3.33. Diagrama de flujo del sistema de seguimiento de trayectoria (Control P). . . . .       | 78 |
| 3.34. Movimiento en dos pasos para dirigirse a un punto. . . . .                             | 78 |
| 3.35. Conjunto difuso “Error de ángulo”. . . . .   | 79 |
| 3.36. Conjunto difuso “Distancia euclídea”. . . . .  | 80 |

## ÍNDICE DE FIGURAS

|   |     |
|---|-----|
| 3.37. Conjunto difuso “Velocidad angular”. . . . .  | 80  |
| 3.38. Conjunto difuso “Velocidad lineal. . . . .  | 81  |
| 3.39. Construcción del controlador difuso en el toolbox Fuzzy Logic Designer de MATLAB. . . . .   | 82  |
| 3.40. Superficie de control de la salida velocidad angular. . . . .                               | 83  |
| 3.41. Superficie de control de la salida velocidad lineal. . . . .                                | 83  |
| 3.42. Diagrama de bloques para el seguimiento de trayectorias (Control Difuso). . . . .           | 84  |
| 3.43. Simulación del desplazamiento del robot (Control Difuso) 1a prueba. . . . .                 | 84  |
| 3.44. Gáfica del error de la velocidad (Control Difuso) 1a prueba. . . . .                        | 85  |
| 3.45. Gáfica de la posición del robot (Control Difuso) 1a prueba. . . . .                         | 85  |
| 3.46. Simulación del desplazamiento del robot (Control Difuso) 2a prueba. . . . .                 | 86  |
| 3.47. Gáfica del error de la velocidad (Control Difuso) 2a prueba. . . . .                        | 86  |
| 3.48. Gáfica de la posición del robot (Control Difuso) 2a prueba. . . . .                         | 87  |
| 3.49. Diagrama de flujo del sistema de seguimiento de trayectoria (Control Difuso). . . . .       | 88  |
| 3.50. Conjunto difuso “Norte”. . . . .  | 89  |
| 3.51. Conjunto difuso “Este”. . . . .   | 90  |
| 3.52. Conjunto difuso “Oeste”. . . . .  | 90  |
| 3.53. Conjunto difuso “Velocidad angular”. . . . .  | 91  |
| 3.54. Conjunto difuso “Velocidad lineal”. . . . .   | 91  |
| 3.55. Conjunto difuso “Formación”. . . . .  | 92  |
| 3.56. Construcción del controlador difuso en el toolbox Fuzzy Logic Designer de MATLAB. . . . .   | 94  |
| 3.57. Superficie de control de la salida velocidad angular. . . . .                               | 95  |
| 3.58. Superficie de control de la salida velocidad lineal. . . . .                                | 95  |
| 3.59. Superficie de control de la salida formación. . . . .                                       | 96  |
| 3.60. Diagrama de flujo del sistema de evasión. . . . .   | 97  |
| 3.61. Representación del cálculo del punto destino para un robot seguidor. . . . .                | 98  |
| 3.62. Representación del cálculo constante de los puntos destino durante el recorrido. . . . .    | 99  |
| 3.63. Representación del control de formación mediante un ángulo y una distancia. . . . .         | 99  |
| 3.64. Navegación de los robots a los puntos de la formación delta. . . . .                        | 101 |
| 3.65. Gráficas del error angular y de distancia de los seguidores en la formación delta. . . . .  | 102 |
| 3.66. Navegación de los robots a los puntos de la formación convoy. . . . .                       | 103 |
| 3.67. Gráficas del error angular y de distancia de los seguidores en la formación convoy. . . . . | 103 |
| 3.68. Representación del seguimiento al punto destino. . . . .                                    | 104 |
| 3.69. Diagrama de flujo del sistema de formación. . . . .   | 105 |
| 3.70. Diagrama de flujo de la función Mantener formación. . . . .                                 | 105 |

|   |            |
|---|------------|
| 3.71. Diagrama área funcional Movimiento. . . . .   | 106        |
| 3.72. Dynamixel XL430-W250-T [32]. . . . .  | 106        |
| 3.73. Integración de las áreas funcionales a nivel estructural. . . . .                   | 108        |
| 3.74. Integración de los sistemas a nivel funcional. . . . .                              | 109        |
| <b>4. Pruebas y Resultados</b>  | <b>110</b> |
| 4.1. Simulación del sistema de localización. . . . .                                      | 111        |
| 4.2. Mediciones del sistema de localización. . . . .                                      | 112        |
| 4.3. Simulación del sistema de asignación de roles. . . . .                               | 113        |
| 4.4. Resultados en consola del sistema de asignación de roles. . . . .                    | 114        |
| 4.5. Simulación del sistema de seguimiento de trayectoria (Control P) Punto 1. . . . .    | 115        |
| 4.6. Simulación del sistema de seguimiento de trayectoria (Control P) Punto 2. . . . .    | 115        |
| 4.7. Simulación del sistema de seguimiento de trayectoria (Control P) Punto 3. . . . .    | 116        |
| 4.8. Simulación del sistema de seguimiento de trayectoria (Control P) Punto 4. . . . .    | 116        |
| 4.9. Simulación del sistema de seguimiento (Control P) modificado Punto 1. . . . .        | 118        |
| 4.10. Simulación del sistema de seguimiento (Control P) modificado Punto 2. . . . .       | 118        |
| 4.11. Simulación del sistema de seguimiento (Control P) modificado Punto 3. . . . .       | 119        |
| 4.12. Simulación del sistema de seguimiento (Control Difuso) 1a prueba. . . . .           | 120        |
| 4.13. Simulación del sistema de seguimiento (Control Difuso) 1a prueba. . . . .           | 120        |
| 4.14. Simulación del sistema de seguimiento (Control Difuso) 1a prueba. . . . .           | 121        |
| 4.15. Simulación del sistema de seguimiento (Control Difuso) 1a prueba. . . . .           | 121        |
| 4.16. Simulación del sistema de seguimiento (Control Difuso) 1a prueba. . . . .           | 122        |
| 4.17. Gráfica de los errores del sistema de seguimiento (Control Difuso) 1a prueba. . . . | 122        |
| 4.18. Gráfica de la posición del sistema de seguimiento (Control Difuso) 1a prueba. . .   | 123        |
| 4.19. Simulación del sistema de seguimiento (Control Difuso) 2a prueba. . . . .           | 123        |
| 4.20. Simulación del sistema de seguimiento (Control Difuso) 2a prueba. . . . .           | 124        |
| 4.21. Simulación del sistema de seguimiento (Control Difuso) 2a prueba. . . . .           | 124        |
| 4.22. Simulación del sistema de seguimiento (Control Difuso) 2a prueba. . . . .           | 125        |
| 4.23. Simulación del sistema de seguimiento (Control Difuso) 2a prueba. . . . .           | 125        |
| 4.24. Gráfica de los errores del sistema de seguimiento (Control Difuso) 2a prueba. . .   | 126        |
| 4.25. Gráfica de la posición del sistema de seguimiento (Control Difuso) 2a prueba. . .   | 126        |
| 4.26. Simulación sistema de evasión. . . . .  | 128        |
| 4.27. Simulación sistema de evasión. . . . .  | 128        |
| 4.28. Simulación sistema de evasión. . . . .  | 129        |

## **ÍNDICE DE FIGURAS**

|   |     |
|---|-----|
| 4.29. Simulación sistema de evasión. . . . .  | 129 |
| 4.30. Simulación sistema de evasión. . . . .  | 130 |
| 4.31. Simulación sistema de evasión. . . . .  | 130 |
| 4.32. Simulación sistema de evasión. . . . .  | 131 |
| 4.33. Simulación sistema de evasión. . . . .  | 131 |
| 4.34. Simulación sistema de evasión. . . . .  | 132 |
| 4.35. Simulación sistema de evasión. . . . .  | 132 |
| 4.36. Posición inicial de los robots en el escenario para la formación en delta. . . . .  | 134 |
| 4.37. Puntos calculados de los seguidores para la formación delta. . . . .                | 135 |
| 4.38. Realización de la formación delta. . . . .  | 135 |
| 4.39. Recorrido en formación delta. . . . .   | 136 |
| 4.40. Posición inicial de los robots en el escenario para la formación en convoy. . . . . | 137 |
| 4.41. Puntos calculados de los seguidores para la formación convoy. . . . .               | 138 |
| 4.42. Realización de la formación convoy. . . . .   | 138 |
| 4.43. Recorrido en formación convoy. . . . .  | 139 |
| 4.44. Cambio de formación de delta a convoy. . . . .                                      | 140 |
| 4.45. Cambio de formación de convoy a delta. . . . .                                      | 141 |

# Índice de Tablas

|   |            |
|---|------------|
| <b>1. Introducción</b>  | <b>1</b>   |
| <b>2. Marco de Referencia</b>                                       | <b>7</b>   |
| <b>3. Análisis y Diseño</b>   | <b>46</b>  |
| 3.1. Especificaciones para el Diseño del Producto (PDS) . . . . .   | 47         |
| 3.2. Matriz morfológica. . . . .                                    | 52         |
| 3.3. Análisis de los diseños conceptuales propuestos. . . . .       | 53         |
| 3.4. Especificaciones sensor LDS-01 [27]. . . . .                   | 60         |
| 3.5. Especificaciones tarjeta OpenCR1.0 [28]. . . . .               | 61         |
| 3.6. Especificaciones Raspberry Pi 3 Model B+ [29]. . . . .         | 64         |
| 3.7. Especificaciones motor Dynamixel XL430-W250-T [32]. . . . .    | 107        |
| <b>4. Pruebas y Resultados</b>                                      | <b>110</b> |
| 4.1. Comparación de los valores del controlador de evasión. . . . . | 133        |

# **Lista de Abreviaturas**

|        |  |
|--------|--|
| GPS    | Sistema de Posicionamiento Global                    |
| IDEF0  | Definición de Integración para Modelado de Funciones |
| IPN    | Instituto Politécnico Nacional                       |
| LiDAR  | Light Detection and Ranging                          |
| MATLAB | Matrix Laboratory                                    |
| MRS    | Sistema Multi-Robot                                  |
| PDS    | Especificaciones para el Diseño del Producto         |
| Pose   | Posición y Orientación del Robot                     |
| RMR    | Robots Móviles con Ruedas                            |
| ROS    | Robot Operating System                               |
| 2D     | 2 Dimensiones  |
| 3D     | 3 Dimensiones  |

# Lista de Símbolos

|                             |   |
|-----------------------------|---|
| $O$                         | Origen de un sistema de referencia  |
| $P$                         | Punto en un sistema de referencia   |
| $d$                         | Distancia entre dos puntos [m]  |
| $\hat{i}, \hat{j}, \hat{k}$ | Vectores unitarios de los ejes coordinados de un sistema de referencia dextrógiro                   |
| $R$                         | Matriz de rotación  |
| $Q$                         | Cuaternion  |
| $T, H$                      | Matriz de transformación homogénea  |
| $V$                         | Velocidad lineal del robot [m/s]  |
| $W$                         | Velocidad angular del robot [rad/s]   |
| $v_d$                       | Velocidad lineal de la rueda derecha del robot [m/s]  |
| $v_i$                       | Velocidad lineal de la rueda izquierda del robot [m/s]  |
| $w_d$                       | Velocidad angular de la rueda derecha del robot [rad/s]   |
| $w_i$                       | Velocidad angular de la rueda izquierda del robot [rad/s]   |
| $r$                         | Radio de las ruedas del robot   |
| $L$                         | Distancia entre las ruedas del robot  |
| $K_v$                       | Ganancia de la velocidad lineal   |
| $K_w$                       | Ganancia de la velocidad angular  |
| $\theta$                    | Orientación del robot [ $^\circ$ , rad]   |
| $\theta_L$                  | Orientación del robot líder [ $^\circ$ , rad]   |
| $\theta_S$                  | Orientación del robot seguidor [ $^\circ$ , rad]  |
| $\psi$                      | Ángulo del eje horizontal a la recta que va del centro del robot al punto destino [ $^\circ$ , rad] |
| $\alpha$                    | Error del ángulo (diferencia entre $\theta$ y $\psi$ ) [ $^\circ$ , rad]                            |
| $\phi$                      | Ángulo deseado en la formación [ $^\circ$ , rad]  |

# Resumen

El presente trabajo muestra el desarrollo de un proyecto para coordinar un grupo de robots móviles terrestres en configuración líder-seguidor, capaz de navegar a través de una trayectoria definida manteniendo una formación y evadiendo obstáculos ya sea modificando la trayectoria o la formación.

Para la localización de los robots, se utilizan los parámetros internos y externos del sistema lo que permite obtener una pose (combinación de la posición y la orientación del robot) inicial para cada robot. Luego, para el seguimiento de trayectorias, se propone un controlador proporcional y uno difuso, evaluando el desempeño que tienen y su integración con la evasión de obstáculos, donde se utiliza un controlador difuso. Finalmente, para la formación y movimiento en conjunto, se implementa un controlador proporcional, con el fin de obtener una respuesta en estado transitorio que cumpla con los requerimientos establecidos para el diseño.

El sistema esta diseñado para ser implementado con tres robots TurtleBot3 Burger, por lo que, para realizar la simulación de los distintos sistemas que lo componen y evaluar su desempeño, fue necesario utilizar el modelo virtual de los robots y del escenario de pruebas propuesto haciendo uso de las herramientas que proporcionan ROS y Gazebo.

**Palabras clave:** Evasión de obstáculos, Robot móvil, Seguimiento de trayectorias, Configuración líder-seguidor.

# Abstract

This work shows the development of a project to coordinate a group of land mobile robots in leader-follower configuration, capable of navigating through a defined trajectory while maintaining a formation and avoiding obstacles either by modifying the trajectory or the formation.

For the location of the robots, the internal and external parameters of the system are used to define an initial pose for each robot. Then, for the tracking of trajectories, a proportional controller and a diffuse one, are proposed, evaluating the performance they have and their integration with obstacle avoidance, where a diffuse controller is used. Finally, for the formation and joint movement, a proportional controller is implemented so that, the response in a transitory state, will meet the requirements established for the design.

The system is designed to be implemented with three TurtleBot3 Burger robots, so, in order to simulate the different systems that make it up and evaluate their performance, it was necessary to use the virtual model of the robots and the proposed scenario using of the tools provided by ROS and Gazebo.

**Key words:** Leader-follower configuration, Mobile robot, Obstacle avoidance, Trajectory tracking.

# 1

## Introducción

En la actualidad, la robótica móvil tiene relevancia en el campo de la industria y de la investigación científica, ya que es una herramienta capaz de desarrollar nuevas tecnologías por medio de la implementación de robots no sedentarios con navegación autónoma [1].

Durante los últimos años, la investigación ha estado enfocada en la mejora de la actuación de los robots móviles en su entorno para lograr la capacidad de realizar tareas requeridas en aplicaciones del mundo real [2].

Sin embargo, gracias a los avances realizados en este campo, este tipo de robots se han vuelto sofisticados; y, en consecuencia, demasiado robustos, poco viables y de difícil acceso. Esto provoca el surgimiento de un nuevo concepto, la robótica colaborativa, en el cual se plantea la utilización de varios robots más simples, que, mediante el funcionamiento cooperativo, puedan llevar a cabo de forma más eficiente y rápida aquellas tareas que resultan difíciles y tardadas de lograr por un solo robot [2].

El problema principal que se presenta al utilizar un sistema de múltiples robots móviles, radica en la coordinación del movimiento de estos para que lleven a cabo una tarea en específico. Esto implica mantener y modificar su formación al igual que su trayectoria, dependiendo de las condiciones de su entorno.

Dicho problema ha atraído significativamente la atención de la comunidad científica, por lo que, actualmente existen distintos métodos para el control de la formación. No obstante, entre estos, el esquema líder-seguidor ha sido mayormente utilizado debido a su simplicidad, flexibilidad, confiabilidad y adaptabilidad [3].

## **Introducción**

---

En esta estrategia, uno o varios robots del sistema se designan como líderes de la formación y el resto como seguidores, a los cuales se les especifica la postura (orientación y posición) deseada en relación con el líder o los líderes [4].

Este proyecto plantea el control de un conjunto de robots en dicha configuración, capaz de seguir una trayectoria en donde se mantenga una formación inicial que pueda modificarse dependiendo del entorno, con el fin de brindar un aporte a las investigaciones en curso. Lo que, en un futuro, podría ser aplicado en actividades como exploración, transporte de objetos, reconocimiento, escolta, búsqueda y rescate, poda, cosecha, siega, vigilancia, fumigación, eliminación de malezas, misiones de patrullaje, entre otras [5].

El presente documento comienza por exponer el análisis de las principales problemáticas en el desarrollo del proyecto, así como las soluciones propuestas. Posteriormente, se presenta la justificación y los objetivos. A continuación, se muestran los trabajos que anteceden al proyecto además de los conceptos y teorías necesarios para comprender el funcionamiento del mismo. Para el diseño, se enuncian los requerimientos al igual que la división por áreas funcionales y el análisis morfológico de algunos parámetros para después explicar el diseño específico de cada uno de estos. Finalmente, en la sección de resultados, aparecen los avances obtenidos en cada uno de los subsistemas y las conclusiones alcanzadas.

## 1.1. Planteamiento del Problema

La robótica en la actualidad es sumamente relevante, tanto en la industria como en la investigación científica, debido a que ayuda a realizar procesos complejos o de alto riesgo para el ser humano.

Existen aplicaciones en donde se requiere la utilización de robots que puedan desplazarse en su entorno con la finalidad de llevar a cabo una tarea específica, lo que suscita la necesidad de conocer su posición constantemente, seguir una trayectoria definida y brindarles la capacidad de evadir obstáculos.

Al implementar un grupo de este tipo de robots en procesos secuenciales y/o repetitivos, se precisa coordinarlos, para así, poder controlar su desplazamiento mientras cumplen con su función. Esto genera distintas problemáticas como son:

- La interacción del robot con su entorno. Conocimiento de la posición y orientación de los robots con la mayor precisión y exactitud posible dentro del espacio de trabajo para así poder controlar su desplazamiento durante el seguimiento de la trayectoria.
- La interacción robot-robot. Utilización del esquema de control líder-seguidor para llevar a cabo la coordinación de los robots dentro de una formación establecida por las condiciones del entorno de trabajo, generando la necesidad de obtener las distancias entre los robots, así como los ángulos entre ellos.
- La interacción humano-robot. Obtención de datos del entorno de forma continua para una correcta toma de decisiones en la evasión de obstáculos y la modificación de la formación durante el recorrido del sistema.
- La interacción humano-sistema. Comunicación entre el humano y los robots para el envío y recepción de información.

## 1.2. Propuesta de Solución

A continuación, se presenta el diagrama general del funcionamiento del sistema de acuerdo con la solución propuesta para el desarrollo del proyecto.

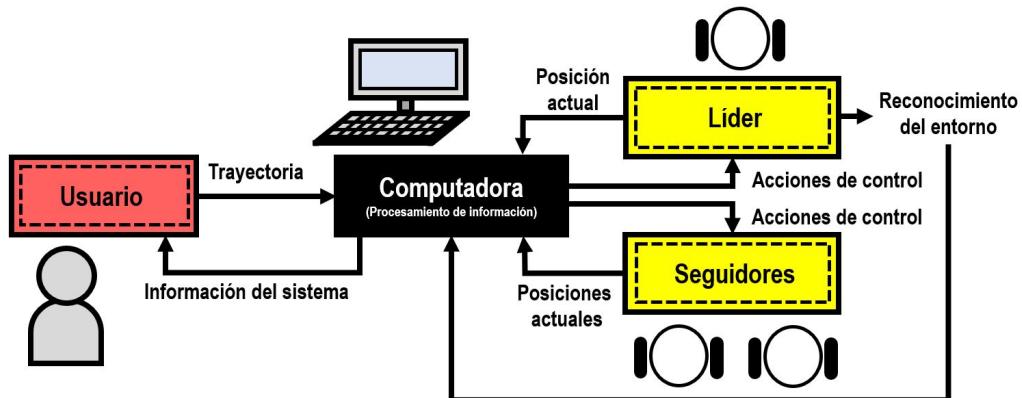


Figura 1.1: Diagrama general de funcionamiento.

Se puede observar en el diagrama de la figura 1.1 que el usuario seleccionará una trayectoria predefinida para que el conjunto de robots la siga, la computadora recibirá la trayectoria, realizará el procesamiento necesario y se la enviará al robot líder. El robot líder es el encargado del reconocimiento del entorno y de mandar constantemente a la computadora su posición actual. Por otro lado, los robots seguidores recibirán de la computadora las coordenadas del robot líder e irán siguiendo los puntos por donde ha pasado este, y de igual forma, enviarán a la computadora su posición actual. La computadora será la responsable de tomar las decisiones en cuanto a modificar la trayectoria para realizar la evasión de obstáculos detectados o reformar al conjunto de robots cuando estos no tengan el espacio suficiente para continuar con su trayectoria.

El diagrama de la figura 1.1 aplicaría de la misma manera para n-número de líderes considerando que cada uno de ellos controlaría su propio grupo de seguidores. Así, las actividades principales a realizar por cada grupo de robots para dar solución a los problemas planteados y lograr el funcionamiento descrito, son:

- Detección del entorno
- Comunicación entre el usuario y el sistema.
- Localización de los robots
- Seguimiento de trayectorias
- Evasión de obstáculos
- Formación de los robots

### 1.3. Justificación

La realización de este proyecto brindará aportaciones al campo de la investigación en la robótica cooperativa realizada en la Unidad Profesional Interdisciplinaria en Ingeniería y Tecnologías Avanzadas (UPIITA) dado que este presenta una nueva propuesta de solución al desarrollo de sistemas robóticos en configuración líder-seguidor para su futura implementación en aplicaciones de distintos campos en los cuales se requiera su utilización para resolver alguna problemática presente.

Actualmente existen distintos trabajos presentados los cuales se enfocan en una tarea en específico como el desarrollo de algoritmos más eficientes para el seguimiento de trayectorias [1], la aplicación de distintas estrategias de control para reducir el tiempo del cambio de la formación [3], la creación de métodos que ayuden a la detección del entorno [4], [10], la implementación de distintos controladores en la estrategia líder-seguidor [5], [7], [8], [9], [11], [12], etc.; pero pocos son aquellos en donde se trabajan estos objetivos en conjunto por lo cual la relevancia de este proyecto no solo radica en los aportes que se puedan realizar a la robótica colaborativa, sino en la integración de las distintas áreas de la robótica que actualmente se encuentran en auge para así diseñar un sistema más completo capaz de desplazarse dentro de un espacio de trabajo de forma coordinada siguiendo una trayectoria dada y realizando el reconocimiento de su entorno para evitar colisiones.

## 1.4. Objetivos

### 1.4.1. Objetivo General

Diseñar e implementar un sistema coordinado de robots móviles terrestres en configuración líder-seguidor capaz de seguir una trayectoria definida y realizar distintas formaciones a partir del reconocimiento de un entorno controlado.

### 1.4.2. Objetivos Particulares

#### Trabajo Terminal I:

- Desarrollar y simular el algoritmo de control a utilizar para que el robot líder lleve a cabo el seguimiento de trayectorias definidas.
- Diseñar un sistema de detección del entorno que permita a los robots evadir obstáculos y cambiar la formación.
- Diseñar un sistema de localización que permita conocer la posición de los robots dentro del entorno de trabajo para el control del sistema.
- Seleccionar y desarrollar un sistema de control para establecer y mantener la formación de los robots seguidores durante el seguimiento de la trayectoria.

#### Trabajo Terminal II:

- Implementar el algoritmo de control para el seguimiento de trayectorias definidas.
- Implementar el sistema de detección del entorno que permitirá a los robots evadir obstáculos y cambiar la formación.
- Determinar e implementar un sistema de comunicación que permita el envío y recepción de información dentro del sistema.
- Implementar el sistema de localización que permitirá conocer la posición de los robots dentro del entorno de trabajo para el control del sistema.
- Implementar el sistema de control para establecer y mantener la formación de los robots durante el seguimiento de la trayectoria.
- Diseñar y programar una interfaz gráfica que muestre al usuario información del estado del sistema para comprobar que realiza lo esperado.

# 2

## Marco de Referencia

La robótica es una ciencia que, desde su surgimiento, ha acompañado al ser humano durante su evolución; compaginándose con su vida cotidiana y brindando un desarrollo vertiginoso en sus distintos campos de aplicación [6].

Existe toda una gama de formas y aplicaciones para los robots, por lo que se pueden considerar diferentes criterios para su clasificación como son:

- Su arquitectura
- Su generación
- Su nivel de inteligencia
- El nivel de control
- El nivel de su lenguaje de programación

Dentro de la clasificación por su arquitectura se encuentran los robots móviles para los cuales, a partir de 1970, la investigación creció de manera exponencial generando una gran cantidad de trabajos que brindan significantes avances en el desarrollo de teorías y conceptos para el control de los mismos [6].

## 2.1. Marco Histórico

### 2.1.1. Antecedentes

Desde la aparición de los robots móviles, se han realizado diversos esfuerzos en la investigación por crear métodos para su posicionamiento, el seguimiento de trayectorias y la evasión de obstáculos, además de la creación de estrategias de control para el desplazamiento sincronizado de un grupo de estos como la configuración líder-seguidor. Los siguientes trabajos son algunos ejemplos.

#### Nacionales:

Un trabajo realizado en la UNAM por Bugarin y Aguilar (2013), lleva por título “Control visual para la formación de robots móviles tipo uniciclo bajo el esquema líder-seguidor”. En este se propone un controlador de robots móviles tipo uniciclo bajo el esquema líder-seguidor mediante el uso de un controlador visual con el cual se eliminan los problemas de oclusión y se facilita el establecimiento de las variables de formación deseadas, distancia y orientación. Esta propuesta se puede lograr por esquemas de control, tanto centralizados como descentralizados. En este trabajo se obtuvieron resultados satisfactorios utilizando un sistema de visión de tiempo real y alta velocidad, lo que permite tener en cuenta el uso de visión artificial como una alternativa viable para el sistema de localización del sistema [4].

#### Internacionales:

El trabajo correspondiente a Solaque, Molina y Rodríguez (2014), se denomina: “Seguimiento de trayectorias con un robot móvil de configuración diferencial” y fue desarrollado en la Universidad Militar Nueva Granada. En este trabajo se desarrolló un sistema de control para un robot móvil de configuración diferencial a partir de la cinemática inversa para el seguimiento de una trayectoria determinada. Utilizaron un robot móvil con restricciones no holonómicas programado en Arduino. Realizaron simulaciones e hicieron pruebas usando dos sistemas de control, con base a criterios de tiempo de respuesta y distancia recorrida se determinó el más eficaz. Se encontró que el control holonómico presenta mejores resultados, ya que, en las pruebas el tiempo empleado y la distancia recorrida son menores. El control holonómico consistió en dividir la trayectoria en diferentes puntos que el robot debe seguir [1].

Otro trabajo realizado en China por Sun, Yang y Chen (2010), se titula: "Leader–follower formation control of multiple non-holonomic mobile robots incorporating a receding-horizon scheme". En este artículo se propone un nuevo marco de control para la formación líder-seguidor para un grupo de robots móviles no holonómicos (robots con restricciones en los ejes de movimiento) en formación. Se utilizan dos esquemas de control donde la orientación de los robots es controlada explícitamente, lo que genera una mayor efectividad en la formación del conjunto de robots y permite el movimiento de estos en conjunto hacia atrás. Los resultados obtenidos demuestran el buen desempeño de los controles de seguimiento y formación, además de que la velocidad de convergencia de error aumento. También, se encontró que el esquema propuesto para controlar el movimiento hacia atrás de los robots es efectivo [3].

Un trabajo desarrollado por Bazoula y Maaref (2008), se llama: "Formation control of multi-robots via fuzzy logic technique". En este se presenta un controlador basado en lógica difusa para la formación de robots móviles no holónomicos, se diseñaron dos controladores; uno para la velocidad lineal, sus entradas fueron la distancia entre los robots y la derivada de la distancia, el otro controlador fue para la velocidad angular, cuyas entradas fueron la orientación de los robots y la derivada de la orientación. Los resultados obtenidos de las simulaciones realizadas fueron exitosos y se demostró la eficacia de los controladores, además de que se plantea que el uso de este controlador podría funcionar con otras formaciones [7].

## 2.1.2. Estado del Arte

Al poseer un vasto campo de aplicación, la robótica móvil continua en pleno desarrollo, por lo que actualmente existen diversos trabajos que presentan propuestas de solución a los distintos ámbitos que abarca esta disciplina. Algunos de ellos se mencionan a continuación.

### Nacionales:

En el Instituto, en 2019 se realizó en la UPIITA un Trabajo Terminal de Mecatrónica llamado “Coordinación de dos robots móviles terrestres bajo un esquema líder-seguidor”. Dicho trabajo se desarrolló utilizando robots Turtlebot3 Burger, los cuales fueron programados con el sistema operativo ROS y se utilizó un sensor LIDAR para el mapeo del entorno. El trabajo consistió en controlar a dos robots bajo el esquema líder-seguidor, el robot líder seguía una trayectoria predeterminada por el usuario y podía esquivar obstáculos fijos, mientras que el robot seguidor imitaba los movimientos del robot líder, mediante la información proporcionada por el robot líder. Todo lo anterior fue probado en un ambiente controlado. Este trabajo resulta de ayuda para el desarrollo del proyecto, debido a que se implementaron algoritmos de localización de un robot, para la detección y evasión de obstáculos, así como el uso del esquema líder-seguidor. Los resultados obtenidos de la implementación del sistema permitieron concluir que se necesita adicionar un factor para aumentar la dimensión de los obstáculos para evitar colisiones debido al tamaño de los robots. También se encontró útil la utilización de varios hilos de ejecución en la programación para aumentar la velocidad de respuesta del sistema [8].

De igual manera, dentro de la UPIITA en 2019, se desarrolló el Trabajo Terminal titulado “Control de sistema robótico multi-agente de navegación terrestre”. Este proyecto consiste en el control de cuatro robots bajo un esquema de coordinación de sistema multi-agente (o enjambre), para el seguimiento de una trayectoria definida por el usuario mediante una interfaz gráfica. En este trabajo se implementaron algoritmos de control como el método de localización de Monte Carlo con muestreo tipo KLD para el monitoreo de obstáculos. Además, se hizo utilizar un sensor de distancia por emisión láser para implementar la evasión de obstáculos. Este trabajo aún no ha sido concluido, pero lo se cita debido a que guarda relación con el objetivo que se buscan cumplir en el proyecto [9].

Otro Trabajo Terminal de Mecatrónica desarrollado en la UPIITA en 2019, nombrado “Sistema Multi-Robot descentralizado para la navegación vial autónoma en un escenario controlado”, este proyecto consiste en la implementación de un sistema compuesto por dos vehículos terrestres no tripulados que pueden desplazarse de manera autónoma por rutas establecidas, las cuales incluyen una intersección vial con dos carriles, los vehículos tienen la capacidad de trasladarse dentro del carril asignado y evitar colisiones entre sí. La adquisición de la imagen del sistema se realizó con una cámara web ubicada sobre el escenario. De igual manera este proyecto aún no llega a su término, pero resulta importante mencionarlo [10].

En el CINVESTAV, del IPN, se llevó a cabo el trabajo de Gallegos, Castro y Velasco (2018), lleva por título “Control de formación líder-seguidor robusto de un conjunto de robots móviles diferenciales”. En este artículo se presenta una estrategia de formación de dos robots móviles basadas en la metodología líder-seguidor. Desarrollaron un controlador para la velocidad lineal y la velocidad angular del seguidor basando en técnicas de regresión, modos deslizantes y pasividad. El ajuste de las ganancias de los controladores se obtuvo tras haber realizado la simulación del comportamiento de los robots. Encontraron que el controlador presenta una discontinuidad cuando la orientación del líder y el seguidor son perpendiculares. Los resultados obtenidos muestran que la utilización de un conjunto de 12 cámaras es una opción viable para este tipo de sistemas [5].

De igual forma, dentro del Instituto, en el CITEDI, se realizó un trabajo que corresponde a Villar (2016), quien realizó el: “Control inteligente aplicado a robótica colaborativa”. En esta tesis se propone una metodología para la realización del seguimiento de trayectorias de robots móviles en formación múltiple bajo el esquema líder-seguidor con la habilidad de reconfigurar la formación. Se tomó como inspiración el comportamiento de las aves para diseñar el algoritmo y lograr que los robots desempeñen tareas del tipo ir-a-meta de manera colaborativa utilizando un controlador difuso llamado síntesis difusa de Lyapunov. Para el sistema de localización se utilizó un sensor externo GPS y un sistema odométrico para compensar los errores del GPS, aunque no es conveniente para aplicaciones que requieran un alto grado de precisión. Los resultados de las simulaciones y experimentos del seguimiento de trayectoria, formación y navegación arrojan resultados satisfactorios en relación con las características cinemáticas de operación del robot. Se demostró que, a pesar de las no linealidades del sistema, es factible utilizar sistemas de inferencia difusa para sistemas de navegación [11].

**Internacionales:**

Este trabajo desarrollado en China por Han, Sun y Lang (2019), se denomina: “Leader-follower formation control of multi-robots based on bearing-only observations”. En este artículo el enfoque se aplica a la utilización de n-número de robots utilizando robots tipo automóvil; es decir, robots equipados con cuatro llantas. Dan solución a tres problemáticas principales, la primera de ellas es la localización del robot líder dentro del espacio de trabajo utilizando únicamente la observación del comportamiento de este, mediante un algoritmo de estimación conocido como “Particle Filter” (PF). La segunda es la observabilidad del sistema del robot líder utilizando rangos de criterios de observabilidad no lineales, la cual está asegurada siempre y cuando el robot líder pueda observar un mínimo de dos puntos de referencia diferentes. Finalmente, la tercera de estas es el control de la formación basado en las observaciones del comportamiento, utilizando un control con retroalimentación que permite el movimiento estable de los robots seguidores. Los resultados de las simulaciones demuestran que el enfoque propuesto puede controlar de manera eficiente la formación múltiples robots [12].

## 2.2. Marco Conceptual

### 2.2.1. Robots Móviles

Un robot móvil se define como un sistema electromecánico capaz de desplazarse de manera autónoma sin estar sujeto físicamente en un punto. Pueden ser terrestres, aéreos o acuáticos.

Los robots móviles terrestres se clasifican por el tipo de locomoción utilizado; en general, su desplazamiento es proporcionado por ruedas, patas y orugas. Si bien la locomoción por patas y orugas ha sido ampliamente estudiada, el mayor desarrollo está en los Robots Móviles con Ruedas (RMR), esto es debido a las ventajas que presentan las ruedas con respecto a las otras.

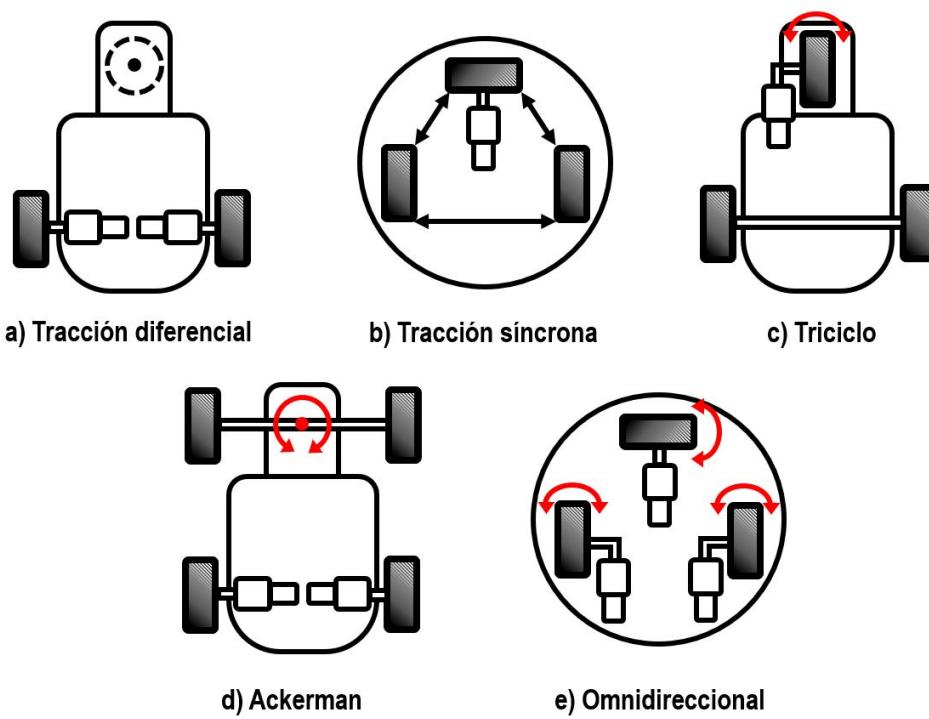
Dentro de los atributos más relevantes de los RMR, destacan su eficiencia en cuanto a energía en superficies lisas y firmes, a la vez que no causan desgaste en la superficie donde se mueven y requieren un menor número de partes, normalmente menos complejas en comparación con los robots de patas y de orugas, lo que permite que su construcción sea más sencilla [6].

Existen diferentes configuraciones cinemáticas para los RMR, éstas dependen principalmente de la aplicación hacia donde se enfocarán; no obstante, en general se tienen las siguientes configuraciones:

- a) Tracción diferencial. Cuenta con dos ruedas fijas en un eje de rotación común, y una o más ruedas locas, típicamente de menor tamaño, cuya función es mantener al robot en un balance. Las dos ruedas fijas son controladas en forma separada, mientras que la rueda loca es pasiva. Tal robot puede girar sobre su eje (i.e. sin mover el punto medio entre las ruedas), dado que las velocidades angulares de las ruedas fijas pueden ser iguales y opuestas.
- b) Tracción síncrona. Este robot tiene tres ruedas dirigibles alineadas que son manipuladas en forma síncrona por sólo dos motores mediante un acoplamiento mecánico. El primer motor controla la rotación de las ruedas sobre su eje horizontal, dando la fuerza al vehículo mientras que el segundo motor controla la rotación de las ruedas sobre su eje vertical, afectando su orientación.
- c) Triciclo. En esta configuración el robot tiene dos ruedas fijas montadas en un eje trasero y tienen una rueda dirigible en el frente. Las ruedas fijas son manipuladas por un solo motor, mientras que la rueda dirigible es manipulada por otro motor que cambia su orientación,

actuando como un dispositivo de dirección. En forma alternativa, las dos ruedas fijas traseras pueden ser pasivas y la rueda frontal puede proveer la tracción y la dirección.

- d) Ackerman. Este tipo de vehículo tiene dos ruedas fijas montadas en un eje trasero y dos ruedas dirigibles montadas sobre un eje delantero. Un motor provee la tracción (trasera o delantera) mientras el otro cambia la orientación de las ruedas delanteras.
- e) Omnidireccional. Robot con tres ruedas locas colocadas en un arreglo simétrico. La velocidad de tracción de las ruedas es manipulada en forma independiente y se puede mover en cualquier dirección cartesiana en forma casi instantánea [13].



## 2.2.2. Sistemas Multi-Robot

Una de las problemáticas de la robótica móvil que más interés ha suscitado es la navegación de agentes robóticos que trabajan cooperativamente en la solución de problemas. Es por esto que, en los últimos años, la investigación se ha dirigido hacia el control de sistemas multi-robot o MRS (por sus siglas en inglés).

La navegación en formación es una de las tareas fundamentales de los MRS ya que es la base para trasladarse dentro de un espacio, esta tarea se define como la coordinación de un grupo de robots para entrar, salir o mantener una formación durante la ejecución de una tarea como puede ser, el desplazamiento de un lugar a otro en grupo.

Por esta razón se han realizado diversos esfuerzos para crear estrategias que permitan la modificación de la formación de MRS durante la ejecución de tareas. Estas estrategias determinan las capacidades y limitaciones de los robots para colaborar entre ellos mismos. Las más comunes encontradas en la literatura son:

- Comportamientos. Los enfoques basados en comportamiento empiezan con el diseño del movimiento deseado para cada robot individual, cada uno debe realizar una tarea específica, como evasión de obstáculos, dirigirse al objetivo y mantener la formación. Los patrones de movimiento de cada robot pueden ser generados con una suma ponderada de la importancia relativa de estas primicias y la interacción de varios robots. Cada comportamiento puede tener entradas de sensores del robot o de otros comportamientos enviando los resultados a los efectores del robot o hacia otro robot.

La ventaja de este tipo de enfoque es la posible implementación en paralelo que presenta, ya que, al tener rutinas de comportamientos asignadas, menos comunicación entre los robots es necesaria. Es de gran utilidad para sistemas en espacios desconocidos o dinámicos; sin embargo, el problema principal que presenta es el complejo análisis matemático que requiere para lograr la convergencia de la formación a una determinada configuración.

- Estructuras virtuales. Este enfoque considera la formación entera de los robots con un cuerpo rígido. La ley de control para un solo robot se deriva mediante la definición de la dinámica de una estructura virtual y luego se traduce el movimiento de la estructura en el movimiento deseado a cada robot. El algoritmo básico ajusta de forma iterativa la estructura a la posición de los robots a través de la minimización de errores entre ellos,

mueve la estructura de acuerdo a un generador de trayectoria local o global y actualiza las posiciones del robot para seguir las trayectorias deseadas de la estructura.

La ventaja principal de esta metodología es que es fácil de prescribir el comportamiento coordinado para el grupo, y que la formación puede mantenerse estable durante maniobras. El inconveniente es que la formación debe ser una estructura rígida, y por lo tanto carece de la flexibilidad y capacidad de adaptación, especialmente cuando la forma de la formación tiene que ser reconfigurada con frecuencia.

- Líder-Seguidor. Con esta estrategia, algunos robots se consideran líderes mientras otros seguidores. La idea básica es que los seguidores de alguna manera conozcan la posición y orientación del líder y lo sigan, manteniendo una distancia predeterminada.

La ventaja principal de esta estrategia es que el movimiento de la formación es determinado en su totalidad por la trayectoria del líder lo que ocasiona que el controlador del sistema sea menos complejo ya que el controlador de formación se reduce a un controlador de seguimiento en donde el líder busca el objetivo del grupo, mientras que los seguidores buscan la ya calculada posición del líder con un offset de distancia prescrito. La principal desventaja es que, no existe realimentación de los seguidores hacia el líder. Además, la formación no tolera errores del líder, es decir, si por alguna razón el líder pierde su trayectoria, la formación entera pierde su referencia.

- Campos potenciales artificiales. Estos en aplicaciones de robótica colaborativa son de gran utilidad para evadir obstáculos y evitar colisiones durante la formación. Esto se logra considerando los obstáculos como una fuerza repulsiva hacia los robots evitando que se acerquen a los mismos. La meta o el punto final de la trayectoria de los robots producen una fuerza atractiva que les indica el camino a seguir. Los campos potenciales atractivos se diseñan con base a la distancia deseada entre los robots en la formación. Por otra parte, los campos potenciales repulsivos son basados en la distancia real de los robots, tienden a infinito cuando dos robots colisionan y disminuye mientras se alejan.

Una ventaja de este método es que requiere menos cálculos por lo que puede ser implementado fácilmente en tiempo real. La desventaja del método es la existencia de mínimos locales en el modelado del espacio [11].

### 2.2.3. Python

Python es un lenguaje de programación orientado a objetos de propósito general y alto nivel comúnmente utilizado para escribir scripts. Cuando se compara con C++, Python es un lenguaje interpretado que ejecuta códigos línea por línea.

Fue creado por Guido van Rossum quien comenzó a desarrollarlo desde 1989, y realizó el primer lanzamiento interno en 1990. Este es un software de código abierto administrado por la fundación sin fines de lucro Python Software.

La principal filosofía de diseño de Python es la legibilidad de código y sintaxis, la cual permite a los programadores expresar sus conceptos en mucho menos líneas de código.

La popularidad de este lenguaje es principalmente debido a lo fácil que es comenzar a aprenderlo. El código es corto, y se puede desarrollar un algoritmo de una manera más rápido en Python que en otros lenguajes. Gracias a su popularidad, existe un gran número de tutoriales en línea, comunidades activas para brindar soporte y muchas librerías para implementar en distintas aplicaciones.

Python es un lenguaje multiplataforma que es ampliamente usado en investigación, redes, gráficos, juegos, machine learning, robótica, etc. Algunas de sus ventajas con respecto a C++ son:

- Estructura estática y dinámica. Python es un lenguaje de estructura dinámica, lo que significa que no se necesita proporcionar el tipo de dato a una variable durante la programación; este lenguaje de programación toma cada variable como un objeto. En C++, se tiene que asignar un tipo de dato a una variable, y solo se puede asignar ese tipo de dato a esa variable durante todo el código.
- Sangría de código. La sangría es el espacio previo colocado en una línea de código. En C++ se debe utilizar la sangría para agrupar un bloque de código, pero no es obligatorio. En Python se debe mantener un bloque de código con la misma sangría; de lo contrario, este mostrara un error. Gracias a que la sangría es obligatoria, el código luce más ordenado y legible.
- Punto y coma. En C/C++, los punto y coma al final de cada línea es obligatoria, pero en Python no lo son. Se pueden utilizar como un separador, pero no como un terminador [14].

## 2.2.4. Robot Operating System (ROS)

ROS es un meta-sistema operativo de código abierto para robots. Brinda los mismos servicios que un sistema operativo, incluidos la abstracción de hardware, el control de dispositivos de bajo nivel, la implementación de funcionalidades de uso común, la transmisión de mensajes entre procesos y la administración de paquetes. Pero también proporciona herramientas y librerías para obtener, construir, escribir y ejecutar códigos en múltiples computadoras. para brindar soporte y muchas librerías para implementar en distintas aplicaciones.

**Meta-Sistema Operativo.** Un meta-sistema operativo describe un sistema que realiza procesos tales como planificación, carga, supervisión y manejo de errores utilizando una capa de virtualización entre aplicaciones y recursos informáticos distribuidos.

Por lo tanto, ROS no es un sistema operativo convencional sino un meta-sistema operativo que se ejecuta en el sistema operativo existente. Así, para operar ROS, primero se debe instalar un sistema operativo como Ubuntu, una de las distribuciones de Linux. Por lo que, además de las características básicas proporcionadas por el sistema operativo convencional, ROS proporciona funciones esenciales requeridas por programas destinados a aplicaciones de robots en forma de librerías.

La comunicación de datos de ROS no solo es soportada por un sistema operativo, sino por múltiples sistemas operativos, hardware y programas, haciéndolo más adecuado para el desarrollo de robots en donde existe la combinación de hardware.

En otras palabras, ROS incluye una capa de abstracción de hardware similar a los sistemas operativos. Sin embargo, a diferencia de los sistemas operativos convencionales, este puede usarse para numerosas combinaciones de implementación de hardware. Además, es una plataforma de software para robots que proporciona varios entornos de desarrollo especializados para crear programas destinados a aplicaciones de robots. Sus principales características son:

- Procesos distribuidos. Se programa en forma de unidades mínimas de procesos ejecutables, y cada proceso corre independientemente e intercambia datos sistemáticamente.
- Gestión de paquetes. Múltiples procesos con el mismo propósito se gestionan como paquete así es sencillo utilizarlo y desarrollarlo, además de conveniente para compartirlo, modificarlo y redistribuirlo.

- Repositorio público. Cada paquete se puede hacer público en el repositorio preferido por el desarrollador y especifica su licencia.
- API (librería). Cuando se desarrolla un programa que usa ROS, ROS permite simplemente llamar una API e insertarla fácilmente dentro del código utilizado.
- Soporte de varios lenguajes de programación. ROS brinda una librería cliente para soportar varios lenguajes de programación. La librería puede ser importada en lenguajes de programación que son populares en la robótica como Python, C++, y Lisp además de lenguajes como JAVA, C#, Lua, y Ruby. En otras palabras, se puede crear un programa en ROS utilizando un lenguaje de programación preferido.

ROS brinda servicios estándares de un sistema operativo como abstracción de hardware, controladores de dispositivos, implementación de características comúnmente utilizadas como sensado, reconocimiento, mapeo, planeación de movimiento, envío de mensajes entre procesos, gestión de paquetes, visualizadores y librerías para el desarrollo además de herramientas de compilación y, para poder desarrollar un robot utilizando ROS, es necesario entender sus componentes, conceptos y comandos esenciales.

**Master.** El maestro actúa como un servidor de nombres para las conexiones de nodo a nodo y la comunicación de mensajes. El comando roscore es utilizado para correr el master, y si se corre el master, se puede registrar el nombre de cada nodo y obtener información cuando sea necesario. La conexión entre nodos y la comunicación de mensajes tales como tópicos y servicios son imposibles sin el master.

**Nodo.** Un nodo se refiere a la unidad más pequeña de procesador que se ejecuta en ROS. Pensando en el cómo un programa ejecutable. ROS recomienda crear un solo nodo para cada propósito tratando de desarrollarlo para poder reutilizarlo fácilmente. Al inicio, un nodo registra información tal como nombre, tipo de mensaje, dirección URI y número de puerto del nodo. El nodo registrado puede actuar como editor, suscriptor, servidor de servicio o cliente de servicio dependiendo de la información registrada, y pueden intercambiar mensajes usando tópicos y servicios.

**Paquete.** Un paquete es la unidad básica de ROS. La aplicación ROS se desarrolla en forma de paquete, y el paquete contiene un archivo de configuración para iniciar otros paquetes o nodos. Este también contiene todos los archivos necesarios para ejecutar el paquete, incluidas las librerías de dependencias ROS para ejecutar diversos procesos, conjuntos de datos y archivos de configuración.

## Marco de Referencia

---

**Meta-paquete.** Un meta-paquete es un conjunto de paquetes que tienen un propósito común.

**Mensaje.** Un nodo envía o recibe datos entre nodos vía mensaje. Los mensajes son variables tales como enteros, flotantes y boléanos. Estructuras de mensajes anidados que contiene otros mensajes o un arreglo de mensajes también puede ser utilizados en un mensaje.

**Tópico.** El tópico es literalmente como un tema en una conversación. El nodo editor primero registra su tópico con el master y luego comienza a publicar mensajes sobre un tema. Los nodos suscriptores que quieren recibir información de un tema lo solicitan al nodo editor correspondiente al nombre del tópico registrado en el master. Basados en esta información, el nodo suscriptor se conecta directamente al nodo editor para intercambiar mensajes como un tópico. La comunicación de los tópicos es una comunicación asíncrona que está basada en un editor y un suscriptor, y es útil para transferir ciertos datos.

**Editor y publicar.** El término “publicar” representa la acción de transmitir mensajes relativos correspondientes al tópico. El nodo editor registra su propia información y tópico con el master, y envía un mensaje a los nodos suscritos que están interesados en el mismo tópico. El editor es declarado en el nodo y puede ser declarado varias veces en un nodo.

**Suscriptor y suscribir.** El término “suscribir” representa la acción de recibir mensajes relacionados correspondiente al tópico. El nodo suscriptor registra su propia información y tópico con el master, y recibe información del editor que publica tópicos relacionados desde el master. Basados en la información recibida del editor, el nodo suscriptor solicita conexión directamente al nodo editor y recibe mensajes desde el nodo editor. Un suscriptor es declarado en el nodo y puede ser declarado varias veces en un nodo.

**Servicio.** El servicio es una comunicación síncrona bidireccional entre el cliente del servicio que solicita información con respecto a una tarea en particular y el servidor del servicio que es el responsable para responder a esa solicitud. A diferencia del tópico, la comunicación del servicio es un mensaje único y cuando la solicitud y la respuesta del servicio ha sido completada, la conexión entre los dos nodos es desconectada.

**Servidor del servicio.** El “servidor del servicio” es un servidor en la comunicación de mensajes de servicio que recibe una solicitud como una entrada y transmite una respuesta como una salida. Tanto la solicitud y la respuesta están en forma de mensajes. Ante la solicitud del servicio, el servidor realiza el servicio designado y entrega el resultado al cliente como una respuesta. Este es implementado en el nodo que recibe y ejecuta una solicitud requerida.

**Cliente del servicio.** El “cliente del servicio” es un cliente en la comunicación de mensajes de servicio que solicita un servicio al servidor y recibe una respuesta como entrada. Tanto la solicitud y la respuesta son en forma de mensajes. El cliente envía una solicitud al servidor y recibe la respuesta. Este es implementado en el nodo que solicita el comando especificado y recibe resultados.

**Acción.** La acción es otro método de comunicación de mensajes utilizado para una comunicación asíncrona bidireccional. La acción es usada cuando toma mucho tiempo responder después de recibir una solicitud y se necesitan respuestas intermedias hasta que el resultado es devuelto. La estructura de un archivo de acción es similar al de un servicio; sin embargo, la sección de datos de retroalimentación para respuesta intermedia se agrega junto con la sección de datos de objetivos y resultados que se representan como solicitud y respuesta en servicio, respectivamente. Hay un cliente de la acción que establece el objetivo de la acción y un servidor de la acción que realiza la acción especificada por el objetivo y devuelve retroalimentación y resultados al cliente de acción.

**Servidor de la acción.** Este está a cargo de recibir el objetivo del cliente y responder con datos de retroalimentación y resultado. Una vez que el servidor recibe el objetivo del cliente, realiza procesos predefinidos.

**Cliente de la acción.** Este está a cargo de transmitir el objetivo al servidor y recibir el resultado o los datos de retroalimentación como entradas. El cliente envía el objetivo al servidor, luego recibe la información, y transmite instrucciones de seguimiento o una instrucción para cancelar el proceso.

**Parámetro.** El parámetro en ROS se refiere variables globales usadas en los nodos. Los valores predeterminados son colocados en el parámetro y pueden ser leídos o escritos si es necesario.

**Servidor de parámetros.** Cuando los parámetros son llamados en el paquete, son registrados con el servidor de parámetros que es cargado en el master.

**Rosrun.** Es el comando de ejecución básico de ROS. Se utiliza para ejecutar un solo nodo en un paquete.

**Roslaunch.** Este ejecuta múltiples nodos. Es un comando especializado en ejecución de nodos con funciones adicionales tales como cambio en los parámetros del paquete o nombres en los nodos, configuración del espacio de nombres de los nodos, cambio en las variables del entorno cuando se ejecutan los nodos, etc.

## Marco de Referencia

---

**Repositorio.** El repositorio es una dirección URL en internet donde los paquetes son guardados. El repositorio gestiona problemas, desarrollo, descargas y otras características utilizando sistemas de control de versiones.

**Librería cliente.** ROS brinda entornos de desarrollo para varios lenguajes de programación utilizando la librería cliente para así reducir la dependencia en el lenguaje utilizado. Con el propósito de soportar distintos lenguajes de programación se crearon librerías cliente tales como rospy, roslisp, rosjava, roslua, etc.

**URI.** Un URI (Identificador uniforme de recursos) es una única dirección que representa un recurso en internet. El URI es una de los componentes básicos que permite la interacción con internet y es usada como un identificador en el protocolo de internet [15].

## 2.2.5. Gazebo

Gazebo es un simulador 3D que provee modelos de robots, sensores y entornos para simulaciones requeridas durante el desarrollo de robots, y ofrece una simulación realista con su motor de física. Gazebo ha sido uno de los simuladores más populares de código abierto en los últimos años, y ha sido seleccionado como el simulador oficial de DARPA Robotics Challenge en los Estados Unidos.

Es un simulador muy popular en el campo de la robótica por su alto desempeño a pesar de ser de código abierto. Además, es desarrollado y distribuido por Open Robotics la cual está a cargo de ROS y su comunidad, por lo que es altamente compatible con ROS. Sus principales características son:

- Simulación dinámica. En las primeras versiones, solo soportaba el ODE (Motor de Dinámica Abierta). Sin embargo, desde la versión 3.0, se utilizan varios motores como Bullet, Simbody, y DART para satisfacer las necesidades de varios usuarios.
- Gráficos 3D. Gazebo utiliza OGRE (Motores de de renderizado de gráficos de código abierto), que es generalmente utilizado en juegos, no solo se puede dibujar realísticamente en pantalla el modelo del robot sino también las luces, sombras y texturas.
- Simulación de sensores y ruido. Soporta distintos tipos de sensores como localizadores laser (LRF) cámaras 2D y 3D, cámaras de profundidad, sensores de contacto, sensores de fuerza-torque, etc. Además, se puede aplicar ruido similar al del entorno actual a los datos del sensor.
- Complementos. Se brindan APIs para permitir a los usuarios crear robots, sensores y control de entornos como un complemento.
- Modelo de robot. PR2, Pioneer 2 DX, iRobot Create y TurtleBot ya son compatibles en forma de SDF, un archivo de modelo de Gazebo, y los usuarios pueden agregar sus propios robots con un archivo SDF.
- Herramienta de líneas de comando. Tanto la herramienta GUI y como la CUI son compatibles para verificar y controlar el estado de la simulación [15].

## 2.3. Marco Teórico

### 2.3.1. Representación de la Posición

Para localizar un cuerpo rígido en el espacio es necesario contar con una herramienta que permita la localización espacial de sus puntos. En un plano el posicionamiento tiene dos grados de libertad, y por tanto a la posición de un punto vendrá definida por dos componentes independientes.

La forma más intuitiva y utilizada de especificar la posición de un punto son coordenadas cartesianas. Existen además otros métodos, igualmente válidos y también ampliamente extendidos, como son las coordenadas polares para dos dimensiones y las cilíndricas y esféricas para espacios de tres dimensiones [16].

#### Sistema Cartesiano de Referencia

Normalmente los sistemas de referencia se definen mediante ejes perpendiculares entre sí con un origen definido. Estos se denominan sistemas cartesianos, el sistema de referencia OXY correspondiente queda definido por dos vectores coordinados OX y OY perpendiculares entre sí con un punto de intersección común O [16].

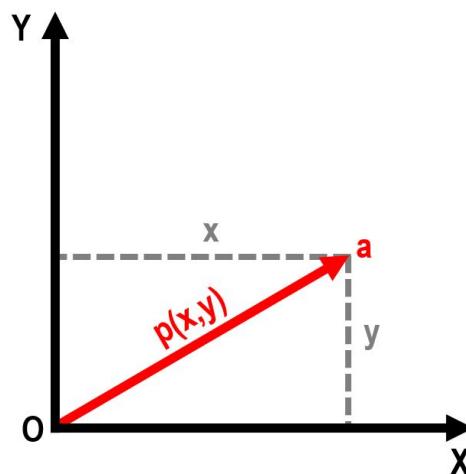


Figura 2.2: Representación de un vector en coordenadas cartesianas en dos dimensiones [16].

#### Coordenadas Cartesianas

Si se trabaja en un plano, con su sistema coordenado OXY de referencia asociado, un punto a vendrá expresado por las componentes  $(x, y)$  correspondientes a los ejes coordinados del sistema OXY. Este punto tiene asociado un vector  $p(x, y)$ , que va desde el origen O del sistema OXY hasta el punto a. Por lo tanto, la posición del extremo del vector  $\vec{p}$  está caracterizada por las dos componentes  $(x, y)$ , denominadas coordenadas cartesianas del vector y que son las proyecciones del vector p sobre los ejes OX y OY [16].

## Coordenadas Polares

También es posible caracterizar la localización de un punto o vector  $\vec{p}$  respecto a un sistema de ejes cartesianos de referencia OXY utilizando las denominadas coordenadas polares  $p(r, \theta)$ . En esta representación,  $r$  representa la distancia desde el origen O del sistema hasta el extremo del vector  $\vec{p}$ , mientras que  $\theta$  es el ángulo que forma el vector  $\vec{p}$  con el eje OX [16].

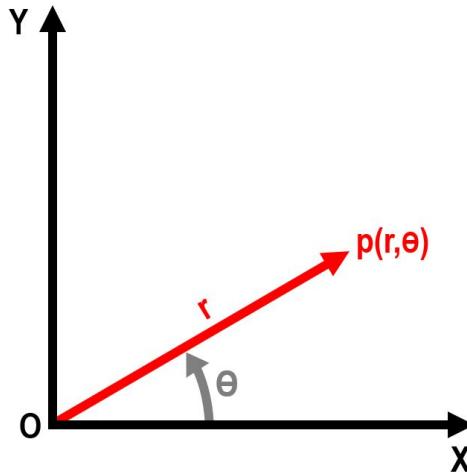


Figura 2.3: Representación de un vector en coordenadas polares [16].

## Distancia Euclíadiana

La distancia euclíadiana hace referencia a la distancia que hay entre dos puntos  $(x_1, y_1)$  y  $(x_2, y_2)$ , esta es calculada a través de la siguiente fórmula:

$$d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2} \quad (2.1)$$

De la misma forma, es posible obtener esta distancia en coordenadas polares, teniendo en cuenta que para ello hay que hacer el respectivo cambio de coordenadas de cartesianas a polares:

$$x = r \cos \theta \quad (2.2)$$

$$y = r \sin \theta \quad (2.3)$$

$$r = \sqrt{x^2 + y^2} \quad (2.4)$$

$$\theta = \tan^{-1} \left( \frac{y}{x} \right) \quad (2.5)$$

Realizando el cambio de coordenadas se obtiene que [17]:

$$d = \sqrt{r_1^2 + r_2^2 - 2r_1r_2 \cos(\theta_2 - \theta_1)} \quad (2.6)$$

### 2.3.2. Representación de la Orientación

Un punto queda totalmente definido en el espacio a través de los datos de su posición. Sin embargo, para el caso de un sólido, es necesario además definir cuál es su orientación con respecto a un sistema de referencia. En el caso de un robot, no es suficiente con especificar cuál debe su posición, sino que en general, es también necesario indicar su orientación.

Para poder describir de forma sencilla la orientación de un objeto respecto a un sistema de referencia, es habitual asignar solidariamente al objeto un nuevo sistema, y después estudiar la relación espacial que existe entre los dos sistemas. De forma general, esta relación vendrá dada por la posición y orientación del sistema asociado al objeto respecto al de referencia. Para el análisis de los distintos métodos de representar orientaciones se supondrá que ambos sistemas coinciden en el origen, y que por tanto no existe cambio alguno de posición entre ellos [16].

#### Matrices de Rotación

Las matrices de rotación son el método más extendido para la descripción de orientaciones, debido principalmente a la comunidad que proporciona el uso de álgebra matricial.

Supóngase que se tiene en el plano dos sistemas de referencia OXY y OUV con un mismo origen O, siendo el sistema OXY el de referencia fijo y el sistema OUV el móvil solidario al objeto (Figura 2.4a). Los vectores unitarios de los ejes coordenados del sistema OXY son  $i_x, j_y$ , mientras que los del sistema OUV son  $i_u, j_v$  (Figura 2.4b).

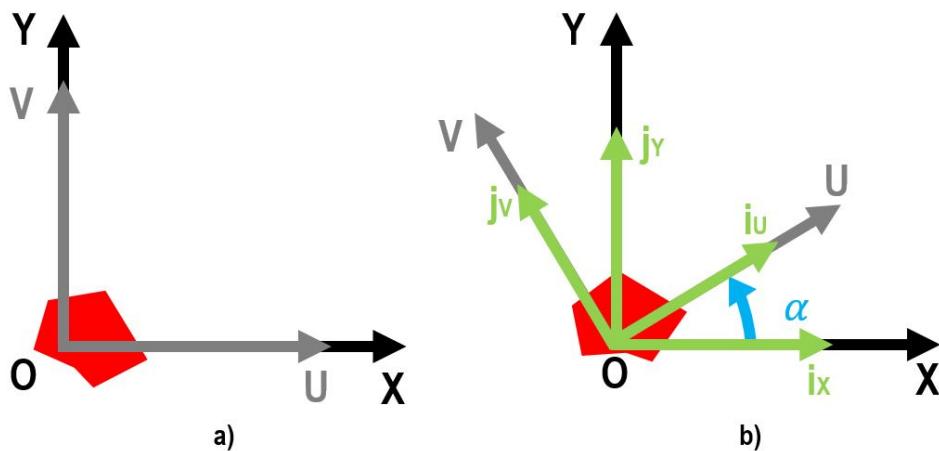


Figura 2.4: Orientación de un sistema OUV respecto a otro OXY en un plano [16].

Un vector  $\vec{p}$  del plano se puede representar en ambos sistemas como:

$$p_{xy} = [p_x, p_y]^T = p_x \cdot i_x + p_y \cdot j_y \quad (2.7)$$

$$p_{uv} = [p_u, p_v]^T = p_u \cdot i_u + p_v \cdot j_v \quad (2.8)$$

Realizando una sencilla serie de transformaciones se puede llegar a la siguiente equivalencia:

$$\begin{bmatrix} p_x \\ p_y \end{bmatrix} = R \begin{bmatrix} p_u \\ p_v \end{bmatrix} \quad (2.9)$$

Donde:

$$R = \begin{bmatrix} i_x i_u & i_x j_v \\ j_y i_u & j_y j_v \end{bmatrix} \quad (2.10)$$

Es la llamada matriz de rotación, que define la orientación del sistema OUV con respecto al sistema OXY, y qué sirve para transformar las coordenadas de un vector en un sistema a las del otro. También recibe el nombre de matriz de cosenos directores. Es fácil de comprobar que se trate de una matriz ortonormal, tal que  $R^{-1} = R^T$ .

En el caso de dos dimensiones, la orientación viene definida por un único parámetro independiente. Si se considera la posición relativa del sistema OUV girando un ángulo  $\alpha$  sobre el OXY, tras realizar los correspondientes productos escalares, la matriz  $R$  será de la forma:

$$R = \begin{bmatrix} \cos(\alpha) & -\sin(\alpha) \\ \sin(\alpha) & \cos(\alpha) \end{bmatrix} \quad (2.11)$$

Para el caso en que  $\alpha = 0$ , es decir, los ejes coordinados de ambos sistemas coinciden, la matriz  $R$  corresponderá a la matriz identidad.

En un espacio tridimensional, el razonamiento a seguir es similar. Supónganse los sistemas OXYZ y OUVW, coincidentes en el origen, siendo el OXYZ el sistema de referencia fijo, y el OUVW el solidario al objeto cuya orientación se desea definir.

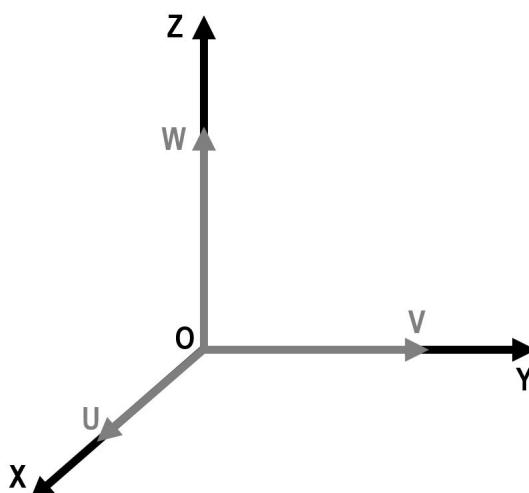


Figura 2.5: Sistema de referencia OXYZ y solidario al objeto OUVW [16].

## Marco de Referencia

Los vectores unitarios del sistema OXYZ serán  $i_x, j_y, k_z$ , mientras que los del OUVW serán  $i_u, j_v, k_w$ . Un vector  $\vec{p}$  del espacio podrá ser referido a cualquiera de los sistemas de la siguiente manera:

$$p_{xy} = [p_x, p_y, p_z]^T = p_x \cdot i_x + p_y \cdot j_y + p_z \cdot k_z \quad (2.12)$$

$$p_{uv} = [p_u, p_v, p_w]^T = p_u \cdot i_u + p_v \cdot j_v + p_w \cdot k_w \quad (2.13)$$

Y al igual que en dos dimensiones, se puede obtener la siguiente equivalencia:

$$\begin{bmatrix} p_x \\ p_y \\ p_z \end{bmatrix} = R \begin{bmatrix} p_u \\ p_v \\ p_w \end{bmatrix} \quad (2.14)$$

Donde:

$$R = \begin{bmatrix} i_x i_u & i_x j_v & i_x k_w \\ j_y i_u & j_y j_v & j_y k_w \\ k_z i_u & k_w j_v & k_z k_w \end{bmatrix} \quad (2.15)$$

Es la matriz de rotación que define la orientación del sistema OUVW con respecto al sistema OXYZ. Al igual que en dos dimensiones, también recibe el nombre de matriz de cosenos directores y se trata de una matriz ortonormal, tal que la inversa de la matriz  $R$  es igual a su transpuesta:  $R^{-1} = R^T$ .

La principal utilidad de esta matriz de rotación corresponde a la representación de la orientación de sistemas girados únicamente sobre uno de sus ejes principales del sistema de referencia [16].

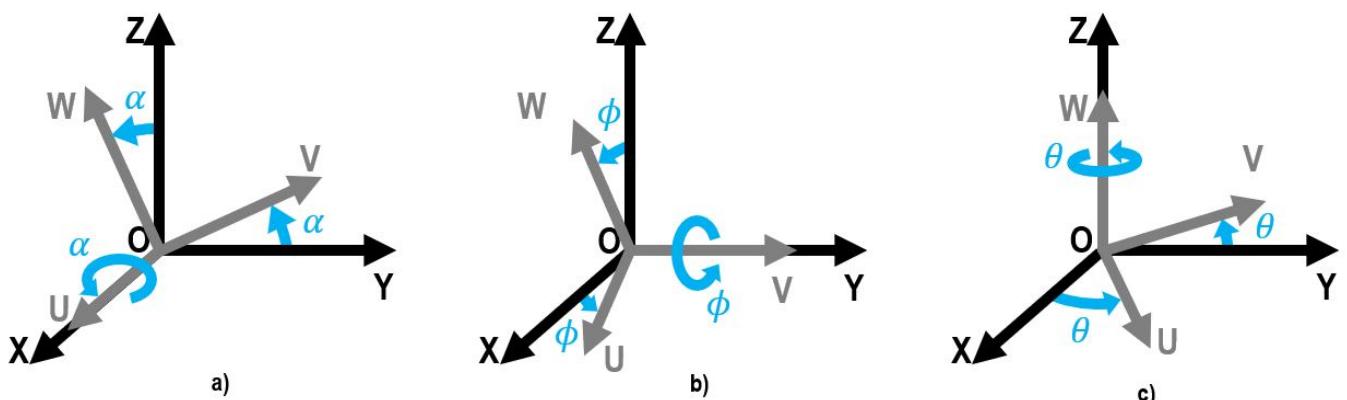


Figura 2.6: Rotaciones del sistema OUVW con respecto al sistema de referencia OXYZ [16].

La orientación del sistema OUVW, con el eje OU coincidente con el eje OX (Figura 2.6a), vendrá representada mediante la matriz:

$$R(x, \alpha) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) \\ 0 & \sin(\alpha) & \cos(\alpha) \end{bmatrix} \quad (2.16)$$

La orientación del sistema OUVW, con eje OV coincidente con el eje OY (Figura 2.6b), vendrá representada mediante la matriz:

$$R(y, \phi) = \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) \\ 0 & 1 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) \end{bmatrix} \quad (2.17)$$

Finalmente, la orientación del sistema OUVQ, con el eje OW coincidente con el eje OZ (Figura 2.6c), vendrá representada mediante la matriz:

$$R(z, \theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 \\ \sin(\theta) & \cos(\theta) & 0 \\ 0 & 0 & 1 \end{bmatrix} \quad (2.18)$$

## Ángulos de Euler

Para la representación de orientación en un espacio tridimensional mediante una matriz de rotación es necesario definir nueve elementos. Aunque la utilización de matrices de rotación presente múltiples ventajas, existen otros métodos de definición de orientación que hacen únicamente uso de tres componentes para su descripción. Este es el caso de los llamados ángulos de Euler [16].

Todo sistema OUVW solidario al cuerpo cuya orientación se quiere describir, puede definirse con respecto al sistema OXYZ mediante tres ángulos:  $\phi, \theta, \psi$ , denominados ángulos de Euler. Girando sucesivamente el sistema OXYZ sobre unos ejes determinados de un triedro ortonormal los valores de  $\phi, \theta, \psi$ , se obtendrá el sistema OUVW. Es necesario, por tanto, conocer además de los valores de los ángulos, cuáles son los ejes sobre los que se realizan los giros. Existen diversas posibilidades (24 formalmente definidas), siendo las 3 más usuales:

**Ángulos de Euler ZXZ.** Es una de las representaciones más habituales entre las que realizan los giros sobre los ejes previamente girados. Se le suele asociar con los movimientos básicos de un giroscopio. Si se parte de los sistemas OXYZ y OUVW, inicialmente coincidentes, se puede colocar al sistema OUVW en cualquier orientación siguiendo los siguientes pasos.

- 1. Girar el sistema OUVW un ángulo  $\phi$  con respecto al eje OZ, convirtiéndose así en el OU'V'W'.
- 2. Girar el sistema OU'V'W' un ángulo  $\theta$  con respecto al eje OU', convirtiéndose así en el OU"V"W".
- 3. Girar el sistema OU"V"W" un ángulo  $\psi$  con respecto al eje OW" convirtiéndose finalmente en el OU""V""W"".

Es importante que estas operaciones se realizan en la secuencia especificada, pues las operaciones de giros consecutivos sobre ejes no son conmutativas [16].

**Ángulos de Euler ZYZ.** Es otra de las representaciones más habituales entre las que se realizan los giros sobre los ejes previamente girados. Solo se diferencia del anterior en la elección del eje sobre el que se realiza el segundo giro. Si se parte de los sistemas OXYZ y OUVW, inicialmente coincidentes, se puede colocar al sistema OUVW en cualquier orientación siguiendo los siguientes pasos.

- 1. Girar el sistema OUVW un ángulo  $\phi$  con respecto al eje OZ, convirtiéndose así en el OU'V'W'.
- 2. Girar el sistema OU'V'W' un ángulo  $\theta$  con respecto al eje OV', convirtiéndose así en el OU"V"W".
- 3. Girar el sistema OU"V"W" un ángulo  $\psi$  con respecto al eje OW" convirtiéndose finalmente en el OU""V""W"".

Como antes, es preciso considerar que el orden de los giros no es conmutativo [16].

**Roll, Pitch y Yaw (Alabeo, Cabeceo y Guiñada).** Se trata de la representación más habitual de entre las que se aplican a los giros sobre los ejes del sistema fijo. Si se parte de los sistemas OXY y OUVW, al igual que en el caso anterior, se puede colocar al sistema OUVW en cualquier orientación siguiendo los siguientes pasos.

- 1. Girar el sistema OUVW un ángulo  $\psi$  con respecto al eje OX. Es el denominado Yaw o guiñada.
- 2. Girar el sistema OUVW un ángulo  $\theta$  con respecto al eje OY. Es el denominado Pitch o cabeceo.
- 3. Girar el sistema OUVW un ángulo  $\phi$  con respecto al eje OZ. Es el denominado Roll o alabeo.

Al igual que en los casos anteriores, y en general, casos en donde se concatenan varios giros seguidos, es necesario considerar que no se trata de una transformación conmutativa, debiéndose seguir una secuencia determinada de aplicación de los mismos [16].

## Cuaterniones

Los cuaternios, pueden ser utilizados como herramienta matemática de gran versatilidad computacional para trabajar con giros y orientaciones. Un cuaternion Q está constituido por cuatro componentes  $(q_0, q_1, q_2, q_3)$  que representan las coordenadas del cuaternion en una base  $e, i, j, k$ . Es frecuente denominar parte escalar del cuaternion a la componente en  $e : q_0$ , y parte vectorial al resto de componentes. De modo que un cuaternion se puede representar como:

$$Q = [q_0, q_1, q_2, q_3] = [s, v] \quad (2.19)$$

Donde  $s$  representa la parte escalar, y  $v$  la parte vectorial [16].

Para la utilización de los cuaternios como metodología de representación de orientaciones se asocia el giro de un ángulo  $\theta$  sobre el vector  $\vec{k}$  al cuaternion definido por:

$$Q = Rot(k, \theta) = \left( \cos\left(\frac{\theta}{2}\right), \vec{k} \sin\left(\frac{\theta}{2}\right) \right) \quad (2.20)$$

### 2.3.3. Transformaciones Homogéneas

La representación mediante coordenadas homogéneas de la localización de sólidos en un espacio n-dimensional se realiza a través de coordenadas de un espacio (n+1)-dimensional. Es decir, un espacio n-dimensional se encuentra representado en coordenadas homogéneas por (n+1) dimensiones, de tal forma que un vector  $p(x, y, z)$  vendrá representado por  $p(wx, wy, wz, w)$ , donde  $w$  tiene un valor y representa un factor de escala. De forma general, un vector  $p = a\hat{i} + b\hat{j} + c\hat{k}$ , donde  $\hat{i}, \hat{j}, \hat{k}$  son los vectores unitarios de los ejes OX, OY y OZ del sistema de referencia OXYZ, se representa en coordenadas homogéneas mediante el vector columna:

$$p = \begin{bmatrix} x \\ y \\ z \\ w \end{bmatrix} = \begin{bmatrix} aw \\ bw \\ cw \\ w \end{bmatrix} = \begin{bmatrix} a \\ b \\ c \\ 1 \end{bmatrix} \quad (2.21)$$

A partir de la definición de las coordenadas homogéneas surge inmediatamente el concepto de matriz de transformación homogénea. Se define como matriz de transformación homogénea  $T$  a una matriz de dimensión 4x4 qué representa la transformación de un vector de coordenadas homogéneas de un sistema de coordenadas a otro.

$$T = \begin{bmatrix} R_{3x3} & P_{3x1} \\ f_{1x3} & w_{1x1} \end{bmatrix} = \begin{bmatrix} \text{Rotación} & \text{Traslación} \\ \text{Perspectiva} & \text{Escalado} \end{bmatrix} \quad (2.22)$$

Se puede considerar que una matriz homogénea se haya compuesta por cuatro submatrices de distinto tamaño: una submatriz  $R_{3x3}$  que corresponde una matriz de rotación; una submatriz  $P_{3x1}$  corresponde al vector de traslación; una submatriz  $f_{1x3}$  qué representa una transformación de perspectiva, y una submatriz  $w_{1x1}$  qué representa un escalado global. En robótica generalmente sólo interesará conocer el valor de  $R_{3x3}$  y  $P_{3x1}$ , considerando la transformación de perspectiva nula y el escalado global unitario. En consecuencia, la matriz homogénea  $T$  resultará ser de la siguiente forma:

$$T = \begin{bmatrix} R_{3x3} & P_{3x1} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} \text{Rotación} & \text{Traslación} \\ 0 & 1 \end{bmatrix} \quad (2.23)$$

Una matriz de transformación homogénea se puede aplicar para:

1. Representar la posición y orientación de un sistema girado y trasladado O'UVW con respecto a un sistema fijo de referencia OXYZ, que es lo mismo que representar una rotación y traslación realizada sobre un sistema de referencia.

2. Transformar un vector  $\vec{r}$  o punto expresado en coordenadas con respecto a un sistema O'UVW, a su expresión en coordenadas del sistema de referencia OXYZ.

$$\begin{bmatrix} r_x \\ r_y \\ r_z \\ 1 \end{bmatrix} = T \begin{bmatrix} r_u \\ r_v \\ r_w \\ 1 \end{bmatrix} \quad (2.24)$$

3. Rotar y trasladar un vector o punto con respecto a un sistema de referencia fijo OXYZ [16].

**Traslación pura.** Supóngase que el sistema O'UVW únicamente se encuentra trasladado un vector  $p = p_x i + p_y j + p_z k$  con respecto al sistema OXYZ. La matriz  $T$  entonces corresponderá a una matriz homogénea de traslación:

$$T(p) = \begin{bmatrix} 1 & 0 & 0 & p_x \\ 0 & 1 & 0 & p_y \\ 0 & 0 & 1 & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.25)$$

Que es la denominada matriz básica de traslación [16].

**Rotación pura.** Supóngase ahora que el sistema O'UVW sólo se encuentra rotado con respecto al sistema OXYZ. La submatriz de rotación  $R_{3x3}$  será la que defina la rotación. Se pueden definir tres matrices homogéneas básicas de rotación partiendo de las matrices básicas de cada eje coordenado del sistema de referencia OXYZ.

$$T(x, \alpha) = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & -\sin(\alpha) & 0 \\ 0 & \sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.26)$$

$$T(y, \phi) = \begin{bmatrix} \cos(\phi) & 0 & \sin(\phi) & 0 \\ 0 & 1 & 0 & 0 \\ -\sin(\phi) & 0 & \cos(\phi) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.27)$$

$$T(z, \theta) = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & 0 & 0 \\ \sin(\theta) & \cos(\theta) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (2.28)$$

La traslación y la rotación son transformaciones que se realizan en relación a un sistema de referencia. Por lo tanto, si se requiere expresar la posición y orientación de un sistema O'UVW, originalmente coincidente con el de referencia y que ha sido trasladado y rotado según éste, habrá que tener en cuenta si primero se ha realizado la rotación y después la traslación o viceversa, pues se trata de transformaciones espaciales no commutativas. Por lo tanto, se tendrán matrices homogéneas distintas según se realice una traslación seguida de rotación o una rotación seguida de traslación [16].

**Traslación seguida de Rotación.** Para el primer caso, las matrices homogéneas resultantes de forma general serían:

$$T(P, R) = \begin{bmatrix} I_{3x3} & P_{3x1} \\ 0 & 1 \end{bmatrix} \begin{bmatrix} R_{3x3} & 0 \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_{3x3} & P_{3x1} \\ 0 & 1 \end{bmatrix} \quad (2.29)$$

**Rotación seguida de Traslación.** Para el segundo caso, las matrices homogéneas resultantes de forma general serían:

$$T(R, P) = \begin{bmatrix} R_{3x3} & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} I_{3x3} & P_{3x1} \\ 0 & 1 \end{bmatrix} = \begin{bmatrix} R_{3x3} & R_{3x3} * P_{3x1} \\ 0 & 1 \end{bmatrix} \quad (2.30)$$

### 2.3.4. Sistemas de Control

Un sistema de control es un arreglo de componentes diseñados con el fin de manipular, dirigir o regular un sistema a través de una acción de control. Para lograrlo, se requiere medir la variable que se desea controlar y aplicar diversas operaciones para corregir las desviaciones que presente la variable medida hasta mantenerla en el valor deseado [11].

Los sistemas de control varían de acuerdo a la aplicación y pueden ser de lazo abierto o cerrado:

- Lazo abierto. Son aquellos donde no se mide la salida y no se compara con la entrada para ejecutar una acción de control.

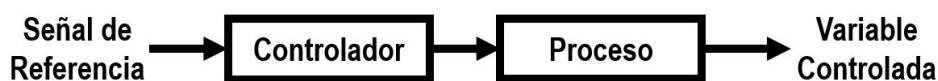


Figura 2.7: Sistema de control en lazo abierto [11].

- Lazo cerrado. Estos sistemas miden la salida del proceso, específicamente la variable que se desea controlar y en base a una operación, tiende a reducir la diferencia o el error entre la salida medida y la entrada de referencia.

En ocasiones no es suficiente aplicar solo una señal de referencia para reducir el error, para ello se requiere un bloque de control que, entre otras cosas, realiza operaciones más sofisticadas para disminuir el error de manera más eficiente.

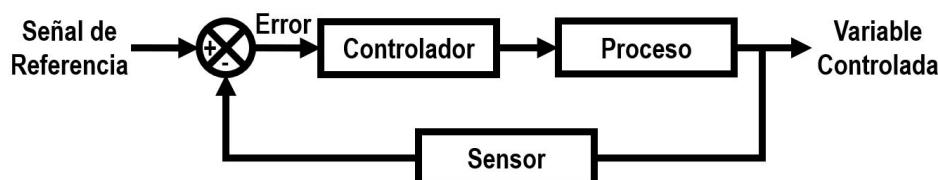


Figura 2.8: Sistema de control en lazo cerrado [11].

El controlador constituye el elemento fundamental en un sistema de control, pues determina el comportamiento del proceso ya que condiciona la acción del elemento actuador en función del error obtenido y la forma en que este genera una señal de control se le denomina acción de control [11].

### 2.3.5. Controladores Clásicos

Dentro de estos, se conocen tres acciones básicas: la proporcional (P), la integral (I) y la derivativa (D). Sin embargo, existen otras que se presentan como combinaciones de estas acciones y dependiendo de la aplicación, se puede elegir la más adecuada para un óptimo desempeño del sistema [18].

La acción de control proporcional P, nos da una salida del controlador proporcional al error . Si la señal de error es grande, el valor de la variable regulada es grande y si la señal de error del sistema es pequeña, el valor de la variable regulada es pequeño igual. Este es el más simple de todos los tipos de control y consiste simplemente en amplificar la señal de error antes de aplicarla a la planta o proceso. Por si solo es limitado por su desempeño y error en régimen permanente.

$$C_p = K_p \quad (2.31)$$

$$u(t) = C_p e(t) \quad (2.32)$$

Para la acción de control integral I, la señal de control es proporcional a la integral de la señal del error. Esto implica que mientras que en la señal de control proporcional el tiempo no influía, en este controlador la acción varía según la desviación de la salida y el tiempo durante el que esta desviación se mantiene.

$$C_i = \frac{K_i}{s} \quad (2.33)$$

$$u(t) = C_i e(t) \quad (2.34)$$

La acción de control proporcional-integral PI, nos permite mejorar el error de estado estacionario en un orden. Es decir que ante un error pequeño positivo, la parte integral realizará una acción de control creciente, y si es negativo, una acción de control decreciente, por lo que el error en régimen permanente será siempre cero.

$$u(t) = C_p e(t) + C_i e(t) \quad (2.35)$$

La acción de control proporcional-derivativa PD, nos permite medir la pendiente instantánea del error. Con esto, se puede predecir el sobrepaso que ocurrirá y realiza una acción correctiva antes que este ocurra.

$$C_d = s K_p \quad (2.36)$$

$$u(t) = C_p e(t) + C_d e(t) \quad (2.37)$$

Finalmente, la acción de control proporcional-integral-derivativa PID, combina las ventajas de cada una de las acciones de control. De forma que si la señal de error varía lentamente en el tiempo, predomina la acción proporcional e integral y, si la señal de error varía rápidamente, predomina la acción derivativa. Tiene la ventaja de tener una respuesta más rápida y una inmediata compensación de la señal de error en el caso de cambios o perturbaciones [19].

$$u(t) = C_p e(t) + C_i e(t) + C_d e(t) \quad (2.38)$$

### 2.3.6. Lógica Difusa

Es un método de razonamiento aproximado no probabilista, que puede definirse como una extensión de la lógica multivaluada que facilita el modelado de información cualitativa de forma aproximada. Permite establecer un mapeo dado un conjunto de variables de entrada obtener un valor adecuado de variables de salida, atendiendo a criterios de significado y no de precisión. La lógica difusa emplea valores continuos entre 0 (que representa hechos totalmente falsos) y 1 (totalmente ciertos).

La aplicación principal de la lógica difusa son los sistemas de control que utilizan expresiones difusas para formular las reglas que controlarán dichos sistemas. Como la lógica difusa sugiere un cierto grado de pertenencia para un dato que se presente dentro de los conjuntos difusos, permite a un controlador difuso tomar diferentes grados de acción en un sistema. Los sistemas difusos son muy recomendables en aquellos problemas complejos donde no existe un modelo matemático simple asociado. De igual forma en procesos que obedecen a un comportamiento no lineal [20].

#### Sistemas Difusos

Un sistema difuso puede definirse como el proceso de obtener valores de salida a partir de las variables de entrada empleando la inferencia difusa que a su vez se apoya en la base del conocimiento [21]. Existen diferentes dos tipos de sistemas difusos:

- Sistemas difusos puros

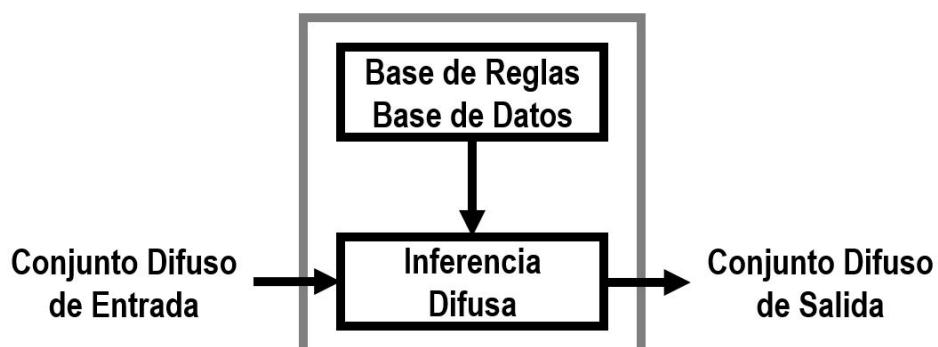


Figura 2.9: Sistema difuso puro [21].

- Sistemas difusos con fusificación y defusificación

- Takagi-Sugeno-Kang (TSK)
- Mandami

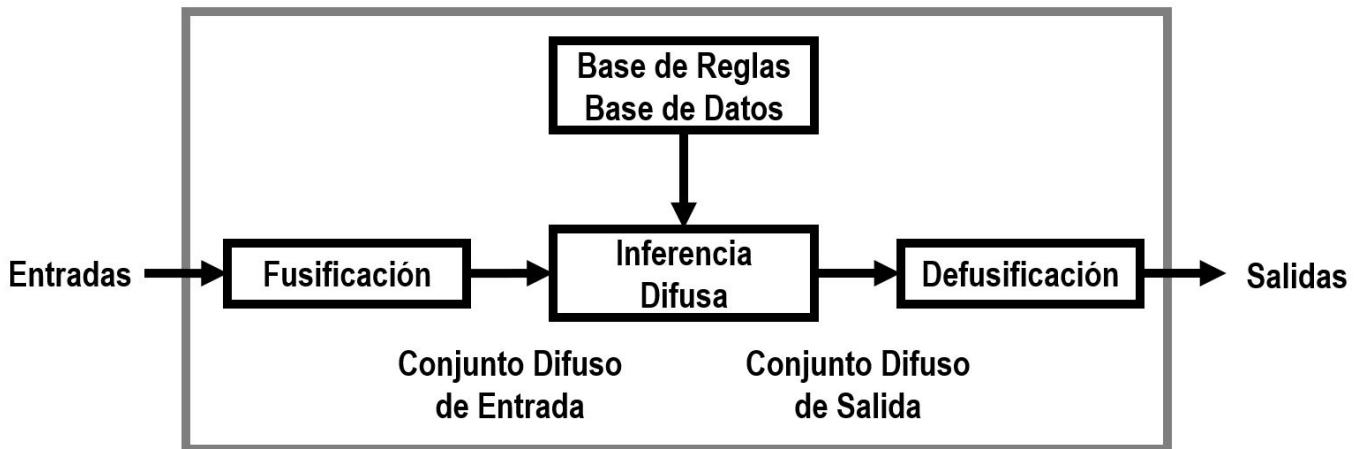


Figura 2.10: Sistema difuso con fusificación y defusificación [21].

El modelo de Mamdani se realiza en cuatro pasos:

1. Fusificación de las variables de entrada. Consiste en tomar los valores de las entradas y determinar el grado de pertenencia de estas entradas a los conjuntos difusos asociados. Cada entrada se fusifica sobre todas las funciones de pertenencia utilizadas en las reglas difusas.
2. Evaluación de las reglas. En esta se toman las entradas anteriores y se aplican a los antecedentes de las reglas difusas. El resultado del antecedente se aplica al consecuente, mediante un recorte o escalado según el valor de verdad del antecedente.
3. Agregación de las salidas de las reglas. Es el proceso de unificación de las salidas de todas las reglas, para obtener un único conjunto difuso por cada variable de salida.
4. Defusificación. En la defusificación se toma como entrada el conjunto difuso anteriormente obtenido para dar un valor de salida, existen varios métodos de defusificación, entre ellos se encuentra el del centroide, el cual calcula el punto donde una línea vertical divide el conjunto en dos áreas con igual masa [20].

$$\text{Centroide} = \frac{\sum_{x=a}^b \mu_A(x)x}{\sum_{x=a}^b \mu_A(x)} \quad (2.39)$$

La principal diferencia es que en los sistemas difusos puros las entradas y salidas son conjuntos difusos, mientras que en los otros se tienen entradas y salidas que tienen variables con valores reales. Siendo los segundos los más utilizados por la ventaja que presentan.

## Conjuntos Difusos

Puede definirse como una clase en la que hay una progresión gradual desde la pertenencia al conjunto hasta la no pertenencia; visto de otra manera, en la que un objeto puede tener un grado de pertenencia definido entre la pertenencia o no pertenencia. El universo del discurso se define como todos los posibles valores que puede tomar una determinada variable [20].

## Operaciones de Conjuntos Difusos

Las operaciones más utilizadas entre los conjuntos difusos son la unión, intersección y el complemento [22]. Definiendo dos conjuntos difusos A y B en el universo X, las operaciones se definen como:

- Unión

$$\mu_{A \cup B}(x) = \mu_A(x) \vee \mu_B(x) \quad (2.40)$$

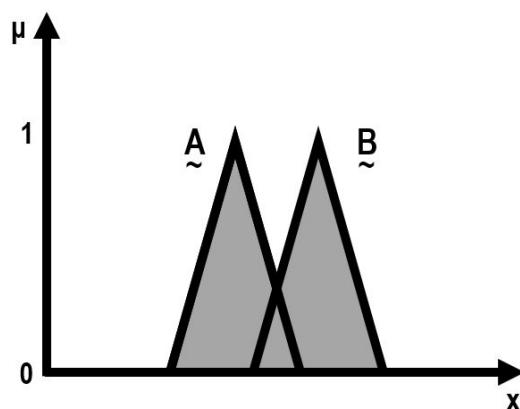


Figura 2.11: Unión de los conjuntos difusos A y B [22].

- Intersección

$$\mu_{A \cap B}(x) = \mu_A(x) \wedge \mu_B(x) \quad (2.41)$$

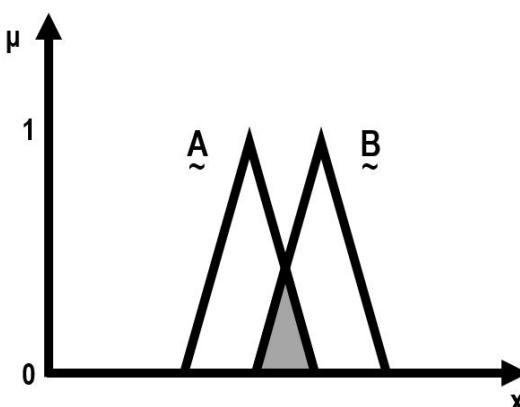


Figura 2.12: Intersección de los conjuntos difusos A y B [22].

- Complemento

$$\mu_{\bar{A}}(x) = 1 - \mu_A(x) \quad (2.42)$$

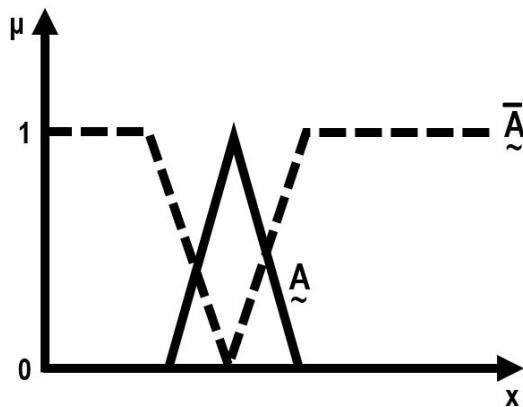


Figura 2.13: Complemento del conjunto difuso A [22].

## Funciones de Membresía

Se le llama a la función que caracteriza el grado de pertenencia de un conjunto difuso, no existen reglas para definir una función de membresía ya que estas varían de acuerdo con el problema a resolver, las funciones más utilizadas son la triangular, trapezoidal, gaussiana y campana [11].

**Función de Membresía Triangular.** Una función de membresía triangular es definida por los parámetros  $a, b, c$  como:

$$\mu_F(x) = \begin{cases} 0, & x < a \\ \frac{x-a}{b-a}, & a \leq x < b \\ \frac{c-x}{c-b}, & b \leq x \leq c \\ 0, & x > c \end{cases} \quad (2.43)$$

Dichos parámetros determinan las coordenadas en  $x$  de los tres puntos del triángulo que forma la función. Estas pueden ser simétricas o antisimétricas [11].

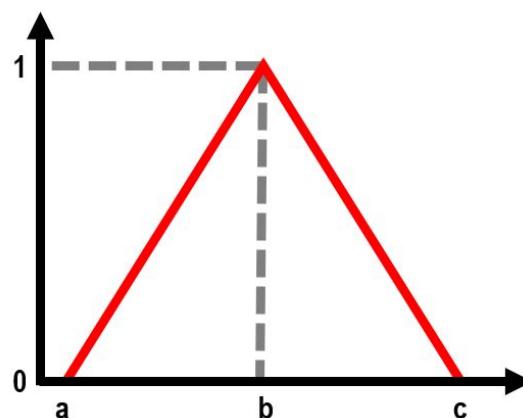


Figura 2.14: Función de membresía triangular [23].

**Función de Membresía Trapezoidal.** Una función de membresía trapezoidal es especificada por los parámetros  $a, b, c, d$  y es definida como:

$$\mu_F(x) = \begin{cases} 0, & x < a \\ \frac{x-a}{b-a}, & a \leq x < b \\ 1, & b \leq x < c \\ \frac{d-x}{d-c}, & c \leq x \leq d \\ 0, & x > d \end{cases} \quad (2.44)$$

Dichos parámetros determinan las coordenadas en  $x$  de los cuatro puntos del trapecio formado por la función. De igual forma, estas pueden ser o no simétricas dependiendo del problema. Debido a la eficiencia de las funciones trapezoidales y triangulares son actualmente las más utilizadas para realizar sistemas de inferencia difusa [11].

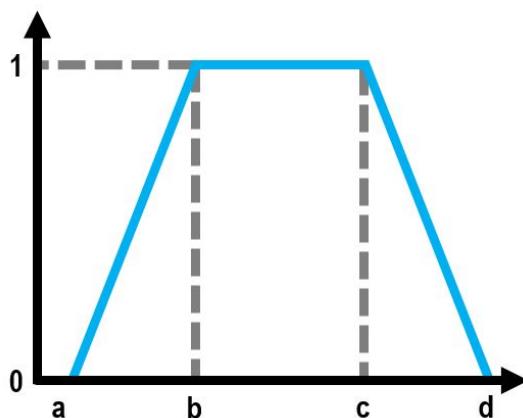


Figura 2.15: Función de membresía trapezoidal [23].

**Función de Membresía Gaussiana.** Una función de membresía Gaussiana es especificada por los parámetros  $C_F, w$  y es definida como:

$$\mu_F(x) = e^{-\frac{(x-C_F)^2}{w^2}} \quad (2.45)$$

En donde  $C_F$  y  $w$  representan la media y la desviación estándar respectivamente [11].

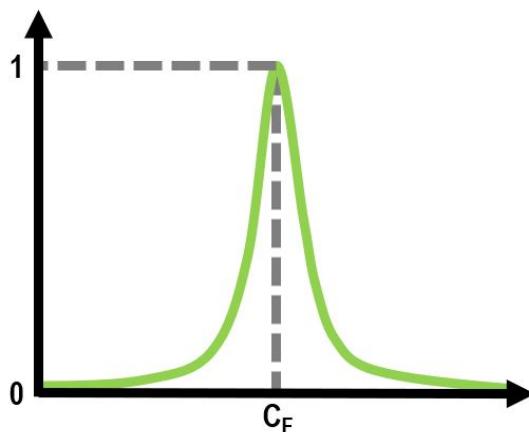


Figura 2.16: Función de membresía gaussiana [23].

## Variables Lingüísticas

Una variable lingüística es aquella cuyos valores son palabras o sentencias en un lenguaje natural o artificial, sirve para representar cualquier elemento que sea demasiado complejo, o del cual no tenemos una definición concreta, es decir, lo que no podemos describir en términos numéricos [20].

## Base de Conocimiento

En un sistema difuso la base del conocimiento se forma de una base de datos y una base de reglas del comportamiento del sistema. Las bases de datos contienen la definición lingüística de las variables de entrada y salida. Las bases de reglas los conjuntos difusos de entrada (antecedentes) y les asocia un conjunto difuso de salida (consecuente), estas se expresan simbólicamente como:

**SI** premisa (antecedente), **ENTONCES** conclusión (consecuente)

Los conjuntos del antecedente pueden asociarse mediante conjuntivas lógicas tipo AND y OR. La base de datos y base de reglas generalmente se obtiene mediante la experiencia y los conocimientos del sistema a través de la observación y el modelado del proceso [11].

## Inferencia Difusa

En lógica difusa el razonamiento no es preciso, sino aproximado, lo cual quiere decir que se puede inferir de una regla, una conclusión, aunque el antecedente de la regla no se cumpla plenamente. Para llevar a cabo la inferencia, se utiliza la base de reglas difusas que modelan al problema que se quiere resolver [11].

### 2.3.7. Modelo Cinemático del Robot Móvil Diferencial

Para realizar el modelado del robot se deben tener en cuenta ciertas hipótesis que generalizan el comportamiento de éste. Se asume que el robot se desplaza en una superficie plana idealmente sin rozamiento, también se toman los ejes de las ruedas como perpendiculares al suelo por donde se desplaza y, por último, el robot se debe mover únicamente debido a las fuerzas ejercidas por el movimiento rotacional de las ruedas.

De igual forma, se deben tener en cuenta las restricciones no holonómicas del mismo. Esto significa que se debe considerar que, un robot en configuración diferencial que opera en un plano bidimensional (como el suelo), solo puede desplazarse hacia delante o hacia atrás a lo largo de su eje longitudinal y rotar alrededor de su eje vertical por lo que no requiere más que del componente lineal x y del componente angular z. En consecuencia, para realizar un desplazamiento netamente lateral, se deben realizar un movimiento en partes [1].

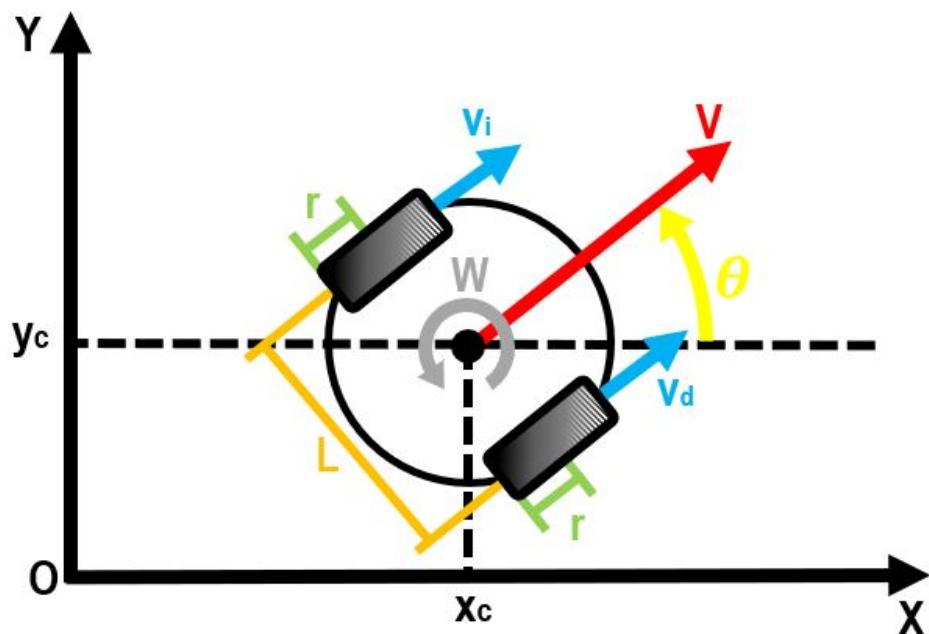


Figura 2.17: Representación geométrica del robot [11].

Se puede observar que, para que el robot se desplace en línea recta la velocidad de sus ruedas deben ser iguales, por lo que se puede definir la velocidad lineal del robot como el promedio de las velocidades lineales de las ruedas:

$$V = \frac{v_d + v_i}{2} \quad (2.46)$$

Y, para hacer rotar al robot sobre su centro de masa, las velocidades lineales de sus llantas deben tener la misma magnitud, pero con signo contrario. Por ende, se puede definir la velocidad angular del robot como la diferencia de las velocidades lineales de sus ruedas sobre la longitud que hay entre ellas:

$$W = \frac{v_d - v_i}{L} \quad (2.47)$$

En donde las velocidades lineales de las llantas se relacionan directamente con sus velocidades angulares y el radio de estas:

$$v_d = r\omega_d \quad (2.48)$$

$$v_i = r\omega_i \quad (2.49)$$

Con esto, se pueden obtener las ecuaciones que definen el movimiento del robot en cada eje:

$$\dot{x} = V \cos (\theta) \quad (2.50)$$

$$\dot{y} = V \sin (\theta) \quad (2.51)$$

$$\dot{\theta} = W \quad (2.52)$$

Finalmente, a partir de (2.50), (2.51) y (2.52), se obtiene el modelo cinemático del robot móvil en configuración diferencial:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos (\theta) & 0 \\ \sin (\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} V \\ W \end{bmatrix} \quad (2.53)$$

Cabe recalcar que el modelo cinemático no considera la dinámica del robot en su movimiento. Por lo tanto, las propiedades dinámicas del mismo tales como la fricción, inercia y la masa, no forman parte de las ecuaciones [11].

# 3

## Análisis y Diseño

Diseñar es el proceso de idear un sistema, componente o proceso para satisfacer las necesidades deseadas. Es un proceso de toma de decisiones, en el cual las ciencias básicas, matemáticas y ciencias de ingeniería son aplicadas para alcanzar un objetivo establecido [24].

Así, la metodología de diseño es la planificación de la actividad total en el desarrollo de un proyecto tomando en cuenta los distintos enfoques que lo integran. En otras palabras, esta describe un proceso de transformación en el cual la información de un problema es transformada en información acerca de una o más soluciones.

En las metodologías de diseño actuales, se propone dividir la planeación y el proceso de diseño en fases o pasos, donde en cada paso se destaca un trabajo intelectual tal que da entrada al siguiente paso. El resultado de estos pasos de trabajo provee las bases de un proceso secuencial para la búsqueda de soluciones de diseño.

- Análisis de especificaciones. Esta actividad es una guía para la formulación de una lista de requerimientos que se deberán cumplir durante el proceso de diseño.
- Diseño conceptual. En esta fase se determinan los principios de solución. Esta se lleva a cabo con la abstracción los problemas esenciales, el establecimiento de estructuras funcionales y la búsqueda de los principios de trabajo adecuados.
- Diseño detallado. El diseño detallado es la materialización del diseño conceptual por medio de la ampliación y extensión de las características generales del diseño que fueron planteadas. Ésta es la fase del proceso de diseño en donde los arreglos, formas, dimensiones y propiedades de todos los elementos son finalmente determinados [25].

### 3.1. Análisis de Especificaciones

Las especificaciones del diseño del proyecto establecen a detalle las condiciones que se tienen que cumplir para su correcta implementación. El análisis de estas, se lleva a cabo por medio de la herramienta PDS (Especificaciones para el Diseño del Producto), en donde se describen las demandas y deseos que regirán el proceso de diseño en términos de requerimientos técnicos de ingeniería.

Tabla 3.1: Especificaciones para el Diseño del Producto (PDS).

| Especificación   | Interpretación   | Requerimiento  | Demandas (D) o deseos (d) |
|--|--|--|---------------------------|
|  | Usar un software/<br>sistema operativo<br>compatible con el robot  | Sistemas operativos:<br>LINUX-Ubuntu 16.04 LTS<br>ROS-Kinetic Kame | D                         |
|  |  | Tipo de robot:<br>Terrestre  | D                         |
|  |  | Configuración del robot:<br>Diferencial                            | D                         |
| Utilizar robots<br>TurtleBot3 Burger   | Diseñar el sistema<br>de acuerdo a las<br>características y<br>componentes del robot   | Tipo de locomoción:<br>Ruedas                                      | D                         |
|  |  | Componentes:<br>Sensor LDS-01                                      |                           |
|  |  | Tarjeta Raspberry Pi 3   | D                         |
|  |  | Motor XL430-W250-T   |                           |
|  |  | Tarjeta OpenCR1.0  |                           |
| Coordinar el<br>desplazamiento de<br>un grupo de robots<br>en donde solo<br>uno conozca la<br>trayectoria a seguir | Mantener a un grupo<br>de robots en<br>formación mientras<br>se desplazan siguiendo<br>los puntos recorridos<br>por uno de ellos | Utilizar la estrategia de<br>control líder-seguidor                | D                         |
| Evitar colisiones  | Evadir los obstáculos<br>que puedan estar<br>dentro de la trayectoria  | Obstáculos que puedan<br>cabrer en un área no<br>mayor a 15cmx15cm | D                         |

| Especificación   | Interpretación  | Requerimiento   | Demanda (D)<br>o deseo (d) |
|--|---|---|----------------------------|
| Permitir al usuario seleccionar las trayectorias y visualizar la información del sistema | Realizar el seguimiento de trayectorias predefinidas                  | Número mínimo de trayectorias: 2                            | D                          |
| Variar la formación dependiendo del espacio de trabajo                                   | Proponer un medio de interacción entre el usuario y la computadora    | Programar una interfaz gráfica                              | D                          |
| Mantener las condiciones de trabajo adecuadas para un funcionamiento correcto            | Implementar distintas formaciones                                     | Número mínimo de formaciones: 2                             | D                          |
| Posición inicial de los robots aleatoria   | Número de robots suficiente para realizar distintas formaciones       | Número de robots mínimo: 3                                  | D                          |
| Formar a los robots lo más rápido posible  | Trabajar en un entorno controlado                                     | Obtáculos fijos   | D                          |
|  | Definir un escenario de pruebas con ciertas características           | Lugar cerrado   | D                          |
|  | Asignación de roles automáticamente                                   | Área máxima: 9m <sup>2</sup>                                | d                          |
|  |   | Material: Madera  | d                          |
|  |   | Superficie lisa y plana                                     | D                          |
| Tener un rango de error mínimo   | Establecer un rango de tiempo aceptable para realizar las formaciones | Definir un robot como líder y los demás como seguidores     | d                          |
|  |   | Tiempo máximo para la formación inicial: 15s                | d                          |
|  |   | Tiempo máximo para cambiar la formación: 15s                | d                          |
|  |   | Error para el seguimiento de la trayectoria: 5cm            | d                          |
|  |   | Error en las distancias y ángulos de la formación: 5cm, 10° | d                          |

### 3.2. División por Áreas Funcionales

Para realizar la división por áreas funcionales, se parte de una función principal que engloba el sistema total y se describe con actividades esenciales para que esta pueda llevarse a cabo. Por lo tanto, cada área funcional resolverá una función particular pero sin perder de vista las relaciones que tendrá con el resto [25].

El modelado del funcionamiento general del sistema se realiza a través de un IDEF0, ya que esta herramienta permite representar de forma estructurada y jerárquica las actividades que conforman el sistema así como la manera en la que interactúan estas entre si.

El bloque inicial presenta la tarea principal a desarrollar, los datos o información que utilizará el sistema, los controles que regularán las actividades, los recursos necesarios y lo que se obtendrá al finalizar (ver la figura 3.1).

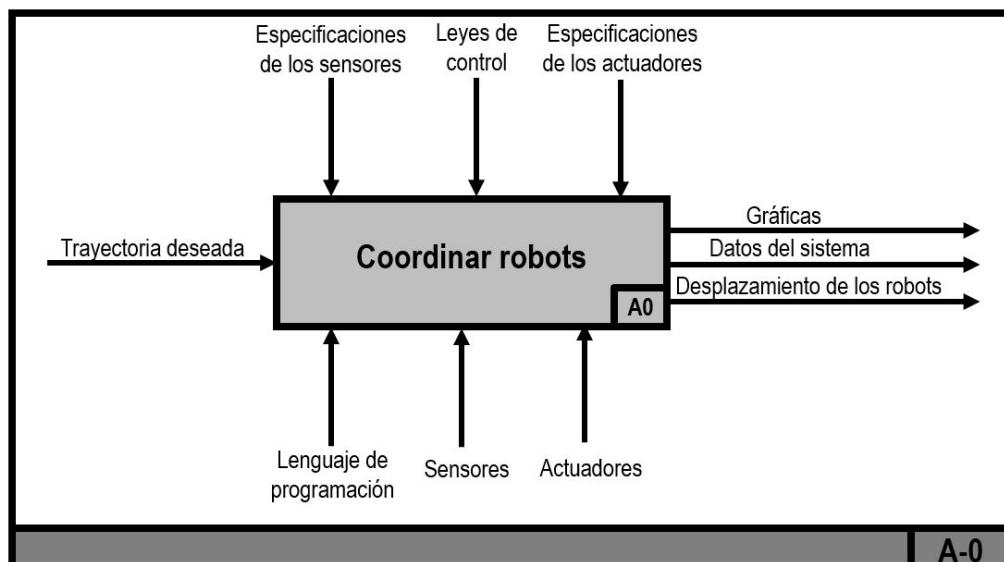


Figura 3.1: IDEF0 (Módulo A-0).

Una vez que la función principal se ha definido, se divide en las actividades esenciales para que esta pueda llevarse a cabo como se muestra en la figura 3.2. Las tres áreas funcionales del sistema serán:

- 1. Percepción.** Su función principal es brindar información del entorno y del sistema al área de procesamiento.
- 2. Procesamiento.** Llevará a cabo el control del sistema basándose en la información del sistema y del entorno. Además, será el medio de comunicación entre el usuario y el sistema.
- 3. Movimiento.** Realizará el desplazamiento de los robots conforme a las señales de control dadas por el sistema de procesamiento.

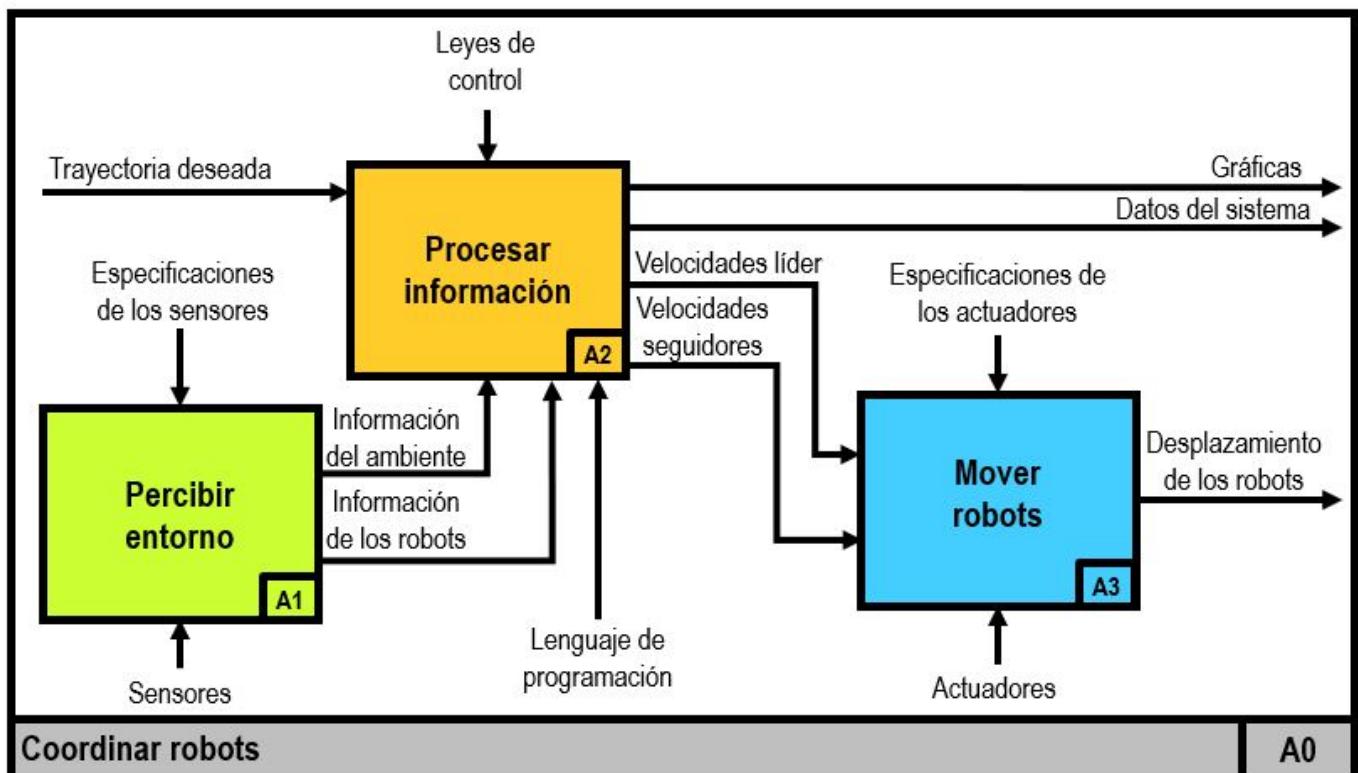


Figura 3.2: IDEF0 (Módulo A0).

A su vez, el área de procesamiento estará constituida por varios subsistemas (figura 3.3):

1. **Sistema de comunicación entre el usuario y el sistema.** Consistirá en una interfaz gráfica que permitirá al operador seleccionar una trayectoria y visualizar información relevante del funcionamiento del sistema.
2. **Sistema de localización.** Definirá el marco de referencia fijo del sistema así como las posiciones y orientaciones iniciales de los robots.
3. **Sistema de asignación de roles.** Definirá el rol de cada robot en base a las distancias entre cada uno de ellos y el punto de inicio de la trayectoria.
4. **Sistema de navegación.** Controlará el movimiento del robot líder durante el seguimiento de la trayectoria y la evasión de obstáculos.
5. **Sistema de formación.** Controlará el movimiento de los robots para así realizar, modificar y mantener la formación.

Finalmente, el sistema de navegación se divide en dos procesos (figura 3.4):

- 1. Seguimiento de trayectoria.** Llevará al robot líder a cada uno de los puntos de la trayectoria deseada.
- 2. Evasión de obstáculos.** Modificará la trayectoria para evadir obstáculos presentes e indicará cuando sea necesario modificar la formación.

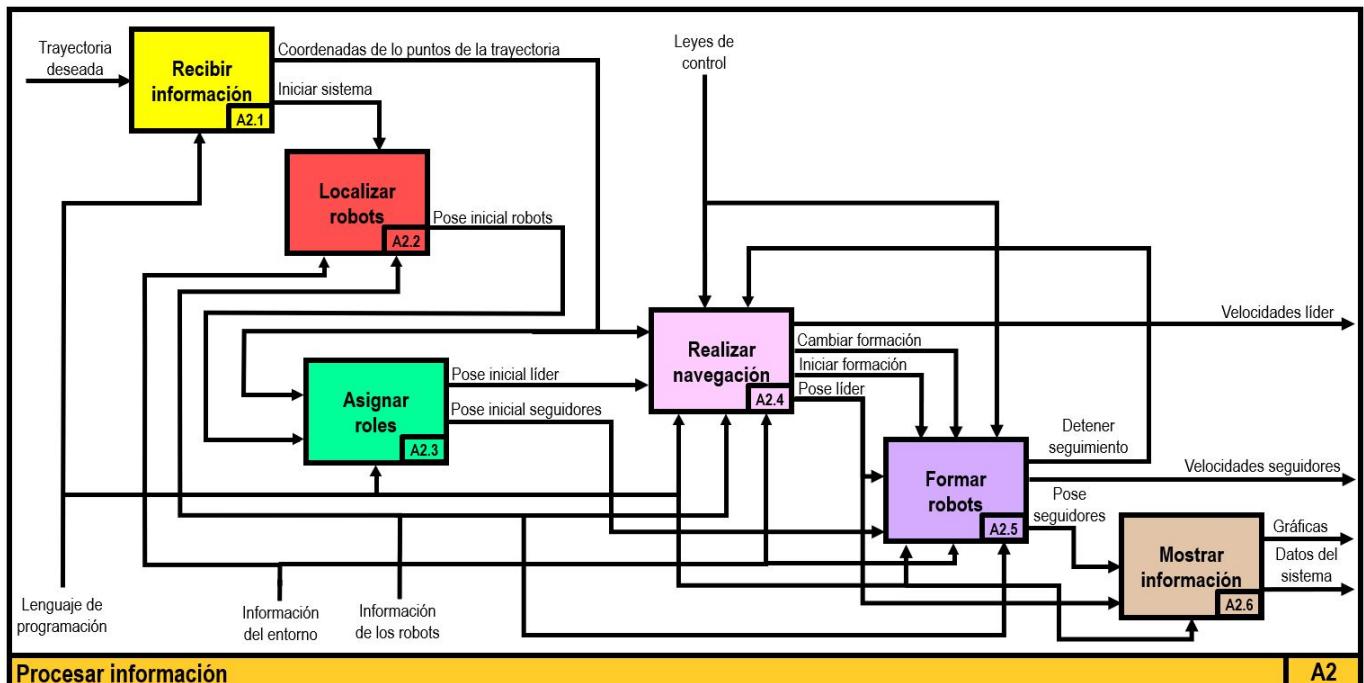


Figura 3.3: IDEF0 (Módulo A2).

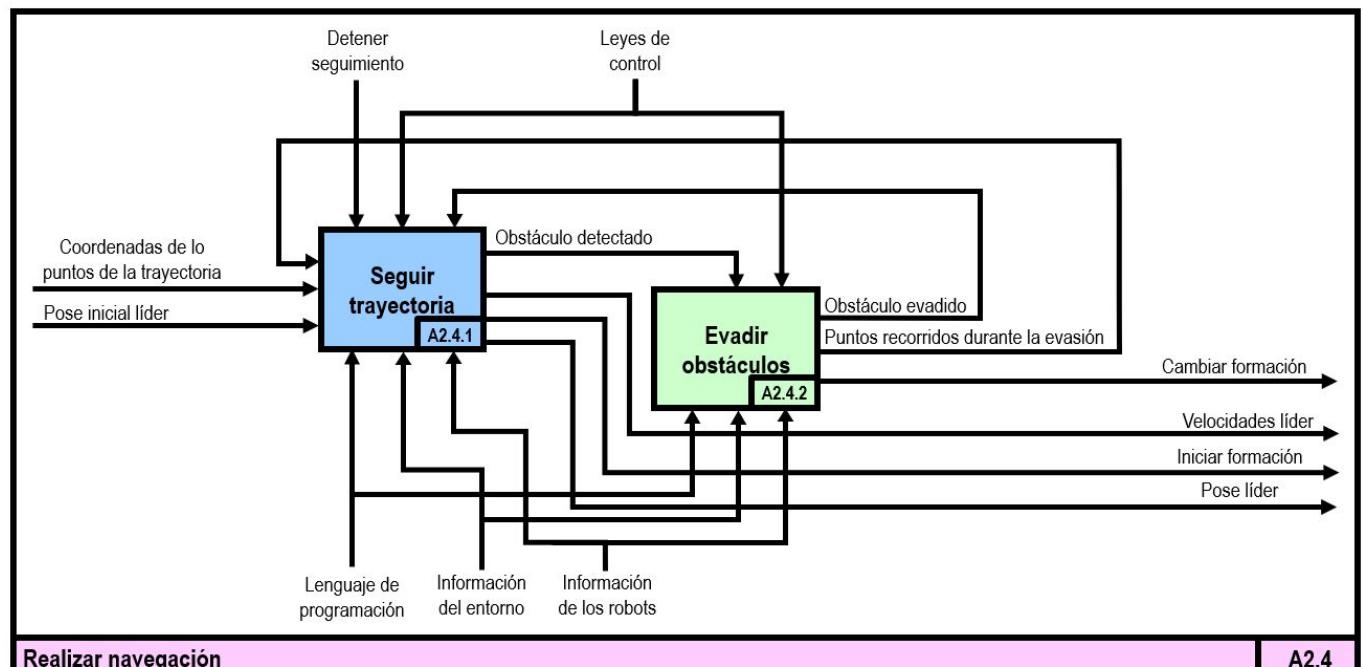


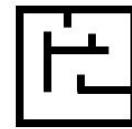
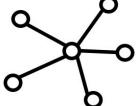
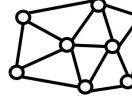
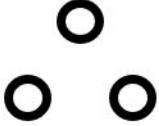
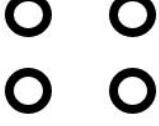
Figura 3.4: IDEF0 (Módulo A2.4).

### 3.3. Diseño Conceptual

En esta sección, a partir de las áreas funcionales, se plantean propuestas de solución para resolver cada una de las tareas y finalmente, obtener un diseño conceptual del sistema.

#### 3.3.1. Matriz Morfológica

Tabla 3.2: Matriz morfológica.

| Aspecto                                | Opciones  |  |   |
|--|---|--|---|
|  | 1   | 2  | 3   |
| A) Forma del escenario de pruebas      |    |    |    |
|  | Rectangular   | Circular   | Cuadrado  |
| B) Obstáculos del escenario de pruebas |  |  |   |
|  | Perpendiculares   | Paralelos  | Laberinto   |
| C) Arquitectura de software            |  |  |  |
|  | Centralizado  | Descentralizado  | Distribuido   |
| D) Número de robots                    |  |  |  |
|  | 3   | 4  | 5   |

#### 3.3.2. Búsqueda Morfológica

Ruta 1: A1-B2-C1-D1

Ruta 2: A3-B1-C2-D2

Ruta 3: A3-B3-C1-D3

Ruta 4: A2-B2-C3-D1

### 3.3.3. Análisis de Ventajas y Desventajas

Tabla 3.3: Análisis de los diseños conceptuales propuestos.

| Ruta | Ventajas   | Desventajas   |
|------|--|---|
| 1    | <ul style="list-style-type: none"> <li>• Facilidad para elaborar el escenario</li> <li>• El escenario de pruebas permite probar el cambio de formación</li> <li>• Arquitectura de software fácil de implementar y con un solo servidor</li> <li>• Formaciones más sencillas por el número de robots</li> </ul> | <ul style="list-style-type: none"> <li>• El control dependerá solo de un dispositivo</li> <li>• Menos opciones de formación</li> </ul>  |
| 2    | <ul style="list-style-type: none"> <li>• Debido a la arquitectura de software, no importa si algún nodo falla</li> <li>• Formaciones más sencillas por el número de robots</li> <li>• Facilidad para elaborar el escenario</li> </ul>  | <ul style="list-style-type: none"> <li>• La arquitectura de software es compleja de implementar</li> </ul>  |
| 3    | <ul style="list-style-type: none"> <li>• Arquitectura de software fácil de implementar y con un solo servidor</li> <li>• Se pueden implementar múltiples formaciones con el número de robots usados</li> </ul>   | <ul style="list-style-type: none"> <li>• Dificultad para elaborar el escenario</li> <li>• No se pueden elegir trayectorias debido a los obstáculos del escenario</li> <li>• Dificultad para reconocer todo el entorno de trabajo</li> </ul> |
| 4    | <ul style="list-style-type: none"> <li>• Ahorro por usar el número mínimo de robots</li> <li>• El poder de cómputo está distribuido</li> </ul>   | <ul style="list-style-type: none"> <li>• Dificultad para elaborar el escenario</li> <li>• Menos opciones de formación</li> </ul>  |

### 3.3.4. Selección del Diseño Conceptual

De las cuatro opciones presentadas, la ruta 3 fue descartada debido a la dificultad para elaborar el escenario de pruebas y por presentar mayores desventajas que ventajas, la ruta 4 también fue descartada por la dificultad de construcción del escenario, la ruta 2 se descartó por la complejidad para implementar la arquitectura de software propuesta. Por lo tanto, se opta entonces por un diseño basado en la ruta 1.

Ahora que ya se ha definido el número de robots, se puede elegir el tipo de formación a implementar, a continuación, se muestran las opciones que se tienen.

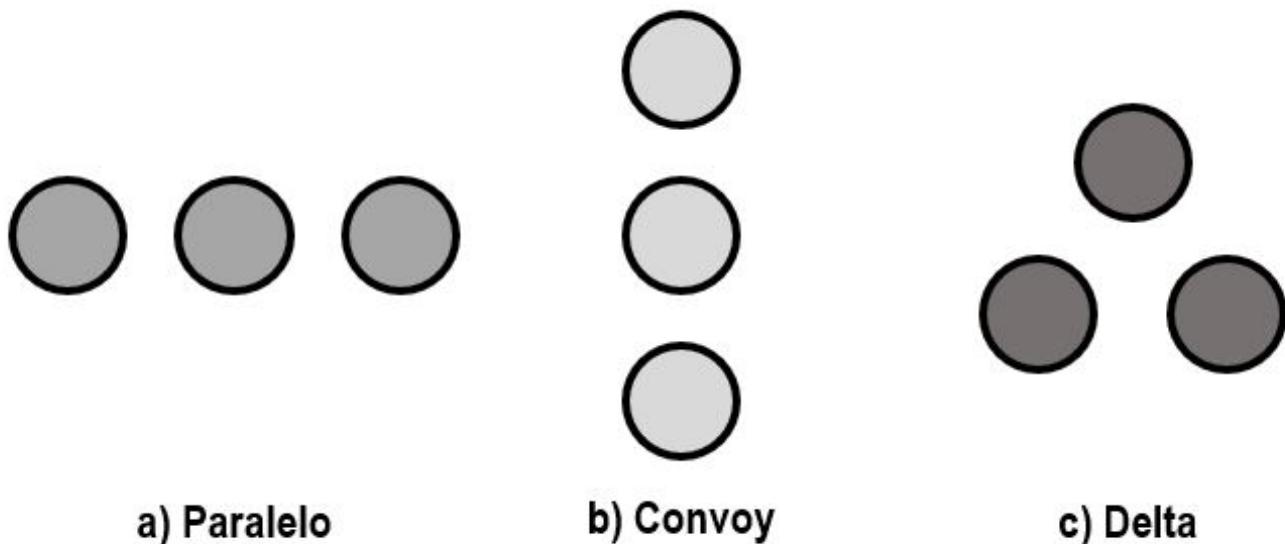


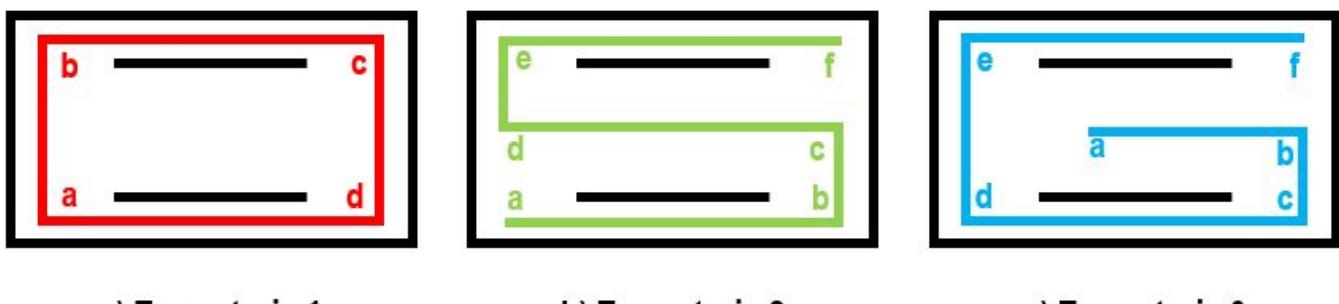
Figura 3.5: Formaciones posibles con tres robots.

Para la implementación de las tres formaciones, la lógica del algoritmo funcionaría de manera similar. Sin embargo, para las formaciones en convoy y en delta, el tamaño es reducido por lo que pueden trabajarse en un escenario de pruebas con dimensiones que cumplan con los requerimientos establecidos, además, existe más documentación de la implementación de ambas en distintos trabajos.

A diferencia de las otras, la formación en paralelo tiene dimensiones mayores, por lo que requeriría un escenario de pruebas amplio. Asimismo, en esta formación los robots seguidores interfieren con las mediciones del robot líder por lo que se requeriría utilizar los datos del entorno adquiridos por los robots seguidores durante el movimiento y en consecuencia, se tendría más información que procesar para realizar la toma de decisiones.

Se busca que la toma de decisiones del sistema dependa completamente del robot líder durante el desplazamiento de los robots, por lo tanto, la formación en paralelo no es una opción viable y se decide trabajar con las formaciones en convoy y en paralelo.

Una vez que se ha seleccionado el escenario de pruebas, es posible definir las trayectorias a trabajar. Es importante mencionar que las opciones son reducidas debido a que los obstáculos colocados reducen el área por donde los robots pueden circular. Sin embargo, a pesar de no ser trayectorias complejas, estas permiten poner a prueba el cambio de formación y visualizar el funcionamiento del sistema.



a) Trayectoria 1

b) Trayectoria 2

c) Trayectoria 3

Figura 3.6: Trayectorias propuestas.

Así, para cada trayectoria, el cambio de formación se llevará a cabo de la siguiente forma:

- Trayectoria 1.
  - Segmentos ab y cd, formación delta.
  - Segmentos bc y da, formación convoy.
- Trayectoria 2.
  - Segmentos ab y ef, formación convoy.
  - Segmentos bc, cd y de, formación delta.
- Trayectoria 3.
  - Segmentos ab, bc y de, formación delta.
  - Segmentos cd y ef, formación convoy.

El diseño conceptual final define las siguientes características:

- Número de robots: 3
- Escenario de pruebas: Rectangular con obstáculos en paralelo
- Arquitectura de software: Centralizado
- Formaciones: Convoy y delta
- Número de trayectorias: 3

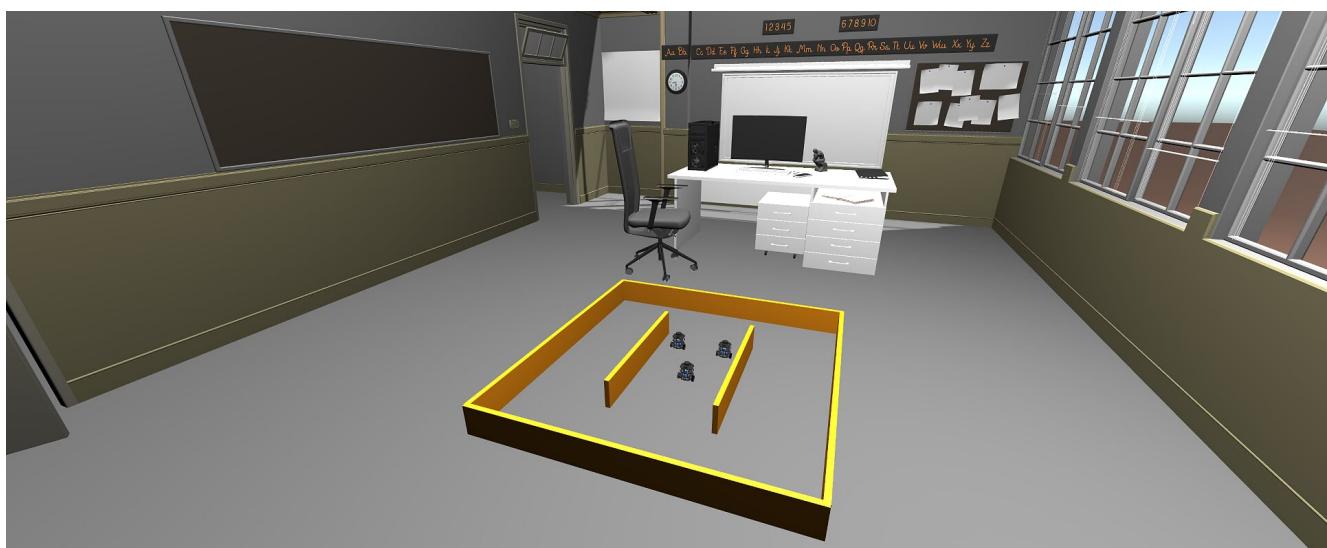


Figura 3.7: Diseño conceptual final.

## 3.4. Diseño Detallado

En esta sección, se analiza a profundidad cada una de las áreas funcionales antes mencionadas junto con las características y datos técnicos de cada uno de los elementos que las conforman. De igual manera, se presenta el diseño de cada uno de los sistemas que intervienen en su funcionamiento.

### 3.4.1. Descripción de los Robots

TurtleBot3 Burger es un robot móvil pequeño con llantas tipo diferencial basado en ROS para su uso en educación, investigación y creación de prototipos. El robot se puede personalizar de varias formas dependiendo de cómo se acomoden sus piezas mecánicas, admite varias estructuras y hardware [26].

Los componentes básicos del robot son:

- Actuadores tipo servomotor: XL430-W250
- Placa de desarrollo: Raspberry Pi 3 Model B and B+
- Tarjeta integrada: OpenCR 32-bit ARM Cortex®-M7 (incluye giroscopio, acelerómetro y magnetómetro)
- Sensor tipo LiDAR: 360 Laser Distance Sensor LDS-01
- Batería tipo Li-po de 3 celdas: Lithium polymer 11.1V 1800mAh / 19.98Wh 5C

Entre las especificaciones técnicas más importantes se encuentran:

- Velocidad lineal máxima: 0.22 m/s
- Velocidad angular máxima: 2.84 rad/s (162.72 deg/s)
- Carga máxima: 15 kg
- Dimensiones: 138 mm x 178 mm x 192 mm (figura 3.8)
- Peso: 1 kg
- Duración de la batería: 150 min
- Tiempo de carga de la batería: 150 min

## Análisis y Diseño

- Pines de propósito general entrada-salida: 18 pines
- Conexión a PC: USB
- Conexiones de alimentación: 3.3V/800mA, 5V/4A, 12V/1A

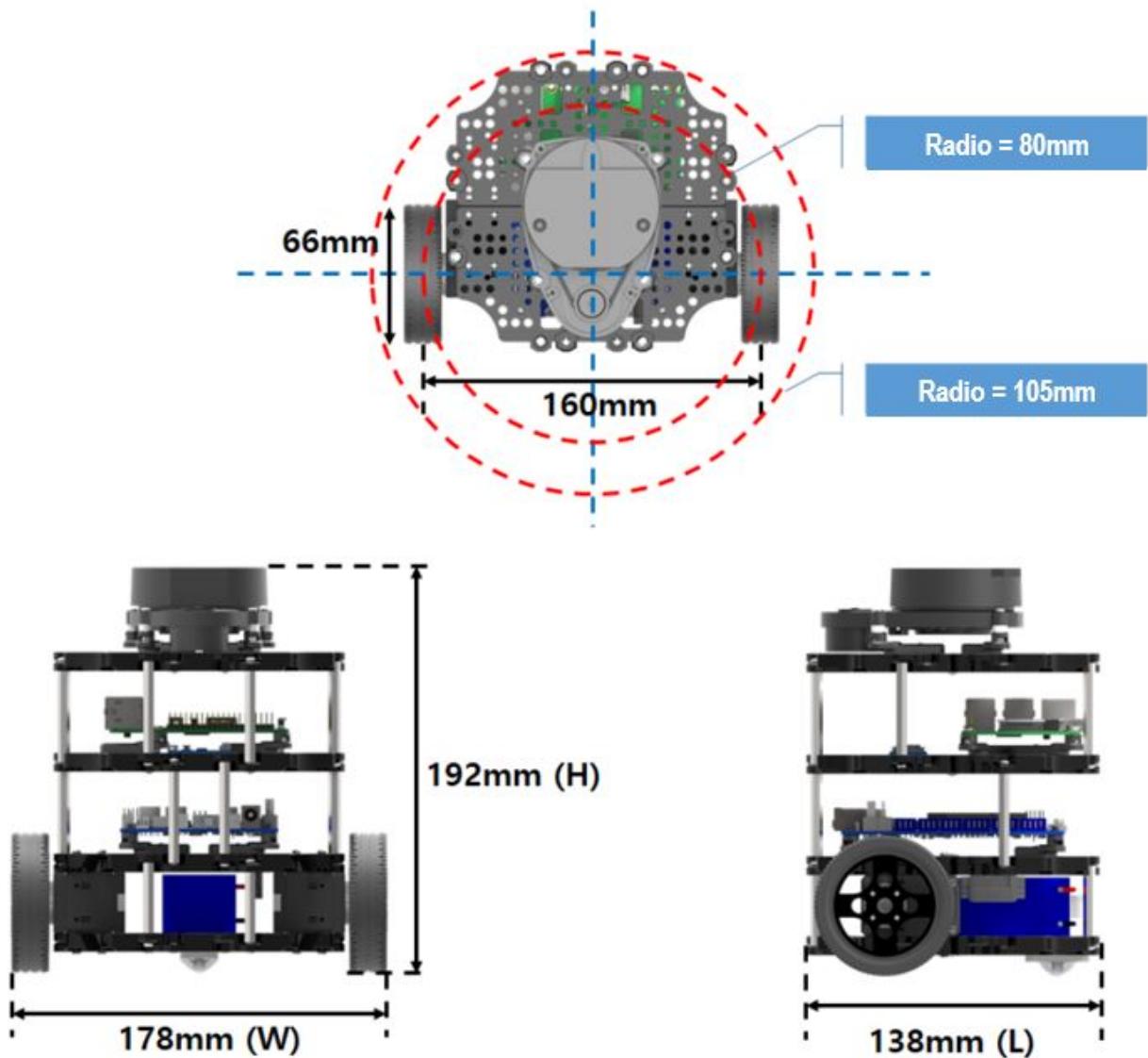


Figura 3.8: Dimensiones TurtleBot3 Burger [26].

### 3.4.2. Área funcional 1: Percepción

En esta área, se obtienen, a través de sensores, tanto los parámetros externos del entorno (Sensor LiDAR) como los internos de los robots (OpenCR1.0, Encoder del Motor). Es decir, se reconoce la existencia de obstáculos en el espacio de trabajo a través de las distancias detectadas alrededor del robot líder así como las posiciones y orientaciones de todos los robots. Esto con el fin de brindar dicha información al sistema de procesamiento para la toma de decisiones como se muestra en el diagrama de la figura 3.9.

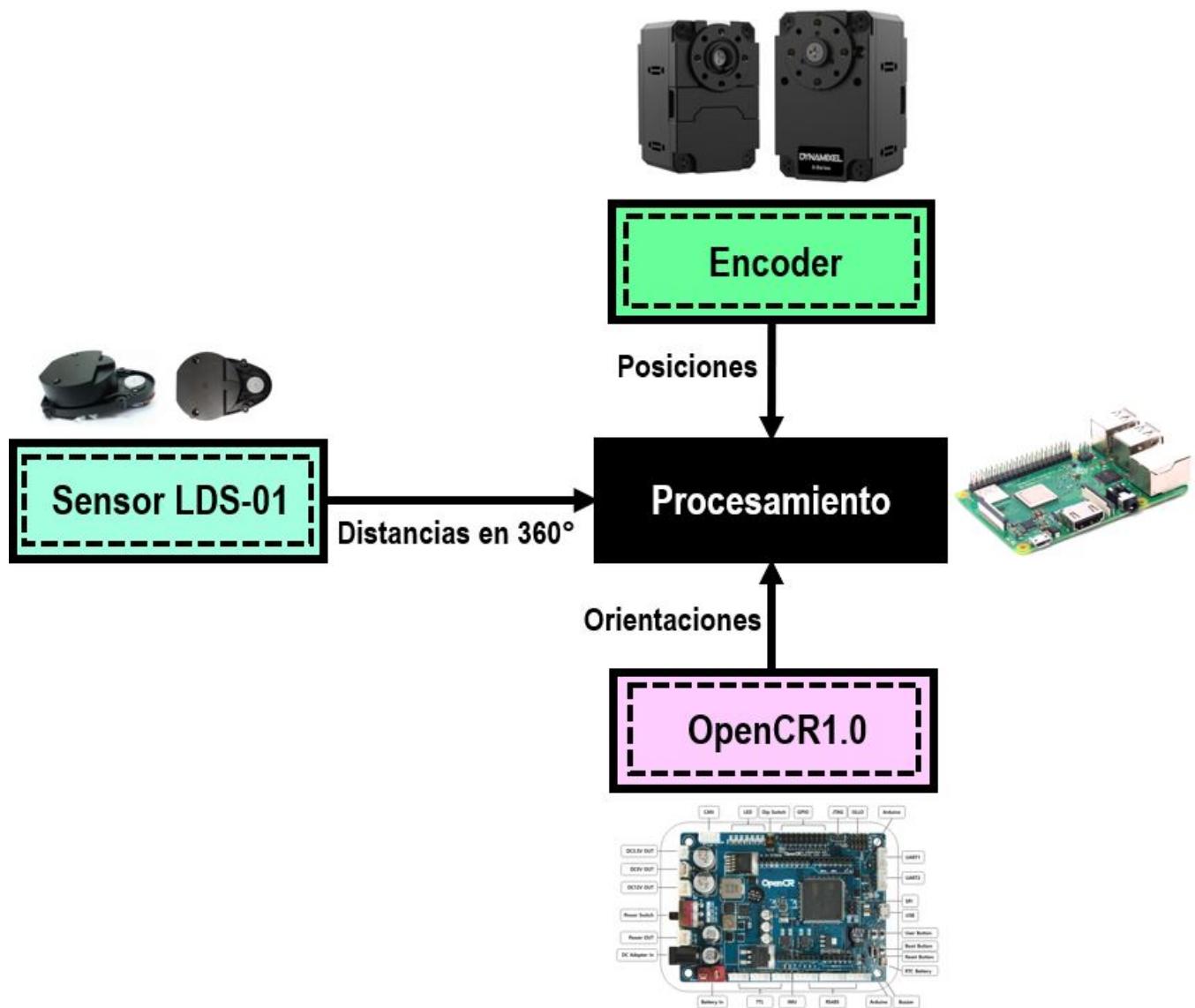


Figura 3.9: Diagrama área funcional percepción.

## Descripción de los componentes

De acuerdo con los requerimientos y la descripción de los robots, los componentes a emplear se describen a continuación.

### Laser Distance Sensor LDS-01

El sensor LSD-01 se compone de un sensor 2D capaz de recolectar información de su entorno a través de su capacidad sensorial en 360° para posteriormente usar esta información en sistemas SLAM y navegación [27].



Figura 3.10: Sensor LDS-01 [27].

Sus características más relevantes para la detección del entorno se presentan a continuación.

Tabla 3.4: Especificaciones sensor LDS-01 [27].

| Elemento                               | Especificación |
|--|----------------|
| Rango de distancia                     | 120 ~ 3,500mm  |
| Exactitud de distancia (120mm~499mm)   | ±15mm          |
| Exactitud de distancia (500mm~3,500mm) | ±5.0 %         |
| Precisión de distancia (120mm~499mm)   | ±10mm          |
| Precisión de distancia (500mm~3,500mm) | ±3.5 %         |
| Velocidad de escaneo                   | 300±10 rpm     |
| Rango angular                          | 360°           |
| Resolución angular                     | 1 °            |

Este sensor, tiene soporte en las plataformas de Windows, Linux y MacOS. Además, permite visualizar la información a través de una herramienta básica GUI encontrada github. Finalmente, se puede conectar a un sistema embebido, como OpenCR y Arduino.

## OpenCR1.0

Es el principal controlador del TurtleBot3. OpenCR está desarrollado para sistemas embebidos en ROS para proveer completamente hardware y software de libre acceso. El principal chip que lo compone, de la serie STM32F7, está basado en un ARM Cortex-M7 con una unidad de punto flotante. Este procesador está diseñado para proveer un rendimiento de alto nivel, manteniendo un excelente tiempo de respuesta y un fácil uso de la arquitectura Armv7-M [28].

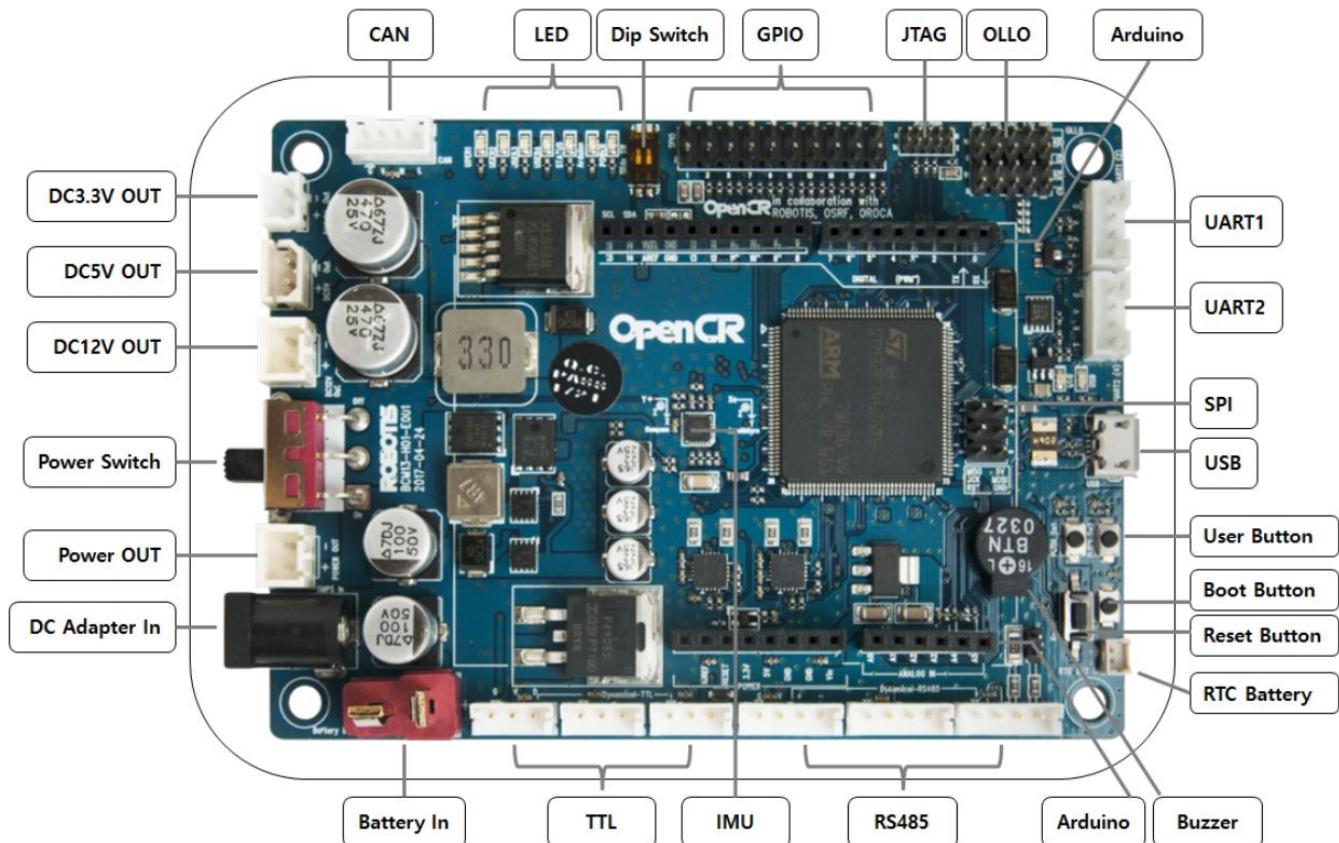


Figura 3.11: OpenCR1.0 [28].

Sus características se presentan en la tabla a continuación.

Tabla 3.5: Especificaciones tarjeta OpenCR1.0 [28].

| Elemento         | Especificación   |
|------------------|--|
| Microcontrolador | STM32F746ZGT6 32-bit ARM Cortex®-M7 con FPU<br>(216MHz, 462DMIPS)            |
| Sensores         | Giroscopio 3 ejes<br>Acelerómetro 3 ejes<br>Magnetómetro 3 ejes<br>(MPU9250) |

## Análisis y Diseño

| Elemento                  | Especificación  |
|---------------------------|---|
| Programador               | Conejero ARM Cortex 10pin JTAG/SWD<br>USB DFU (Device Firmware Upgrade)<br>Serial   |
| Pines de extensión        | 32 pins (L 14, R 18) *Conectividad con Arduino<br>Módulo del sensor x 4 pins<br>Conejero de extensión x 18 pins   |
| Circuitos de comunicación | USB (Micro-B USB/USB 2.0/Host/Peripheral/OTG)<br>TTL (B3B-EH-A / DYNAMIXEL)<br>RS485 (B4B-EH-A / DYNAMIXEL)<br>UART x 2 (20010WS-04)<br>CAN (20010WS-04)  |
| LEDs y botones            | LD2 (rojo/verde) : Comunicación USB<br>LED x 4 : LD3 (rojo), LD4 (verde), LD5 (azul)<br>Botón x 2   |
| Alimentación              | Entrada de alimentación externa<br>5 V (USB VBUS), 7-24 V (Batería or SMPS)<br>Batería de fábrica : LI-PO 11.1V 1,800mAh 19.98Wh<br>SMPS: 12V 5A<br>Salida de alimentación externa<br>12V@1A(SMW250-02), 5V@4A(5267-02A),<br>3.3V@800mA(20010WS-02)<br>Puerto de batería externo para RTC (Real Time Clock) (Molex<br>53047-0210)<br>LED de alimentación: LD1 (rojo, 3.3 V)<br>Botón de reseteo x 1 (Para resetear la tarjeta)<br>Switch de encendido/apagado x 1 |
| Dimensiones               | 105x75 mm   |
| Peso                      | 60g   |

### 3.4.3. Área funcional 2: Procesamiento

Esta área está formada por la computadora y la tarjeta de desarrollo del robot (Raspberry Pi 3 Model B+), las cuales se mantienen en constante comunicación, a través de un medio inalámbrico, para dar y ejecutar las acciones de control pertinentes.

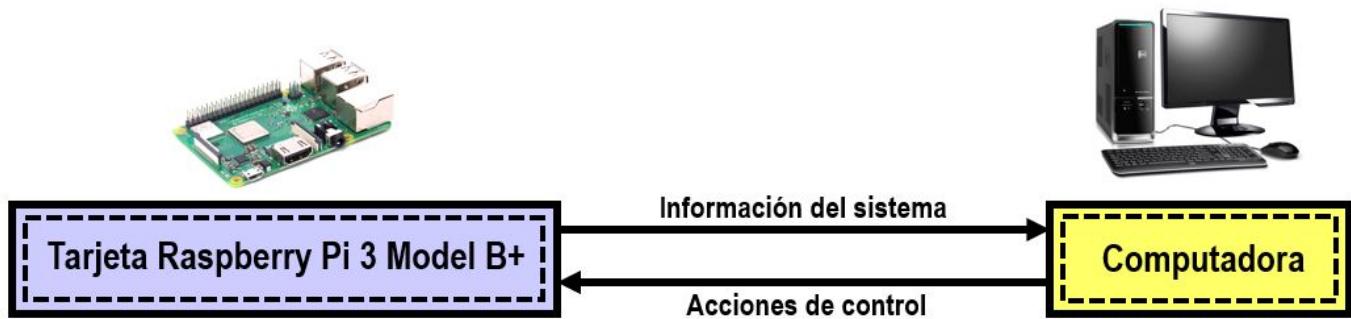


Figura 3.12: Diagrama área funcional Procesamiento.

### Descripción de los componentes

De acuerdo con los requerimientos y la descripción de los robots, los componentes a emplear se describen a continuación.

#### Raspberry Pi 3 Model B+

Su principal función es ser básicamente un ordenador funcional, capaz de procesar todos los datos que se le den. El modelo B+ es el producto más reciente de la Raspberry Pi 3, y cuenta con un procesador de cuatro núcleos a 64-bit, con una velocidad de 1.4GHz, banda dual wifi de 2.4GHz y 5GHz, Bluetooth 4.2, Ethernet y capacidad PoE [29].

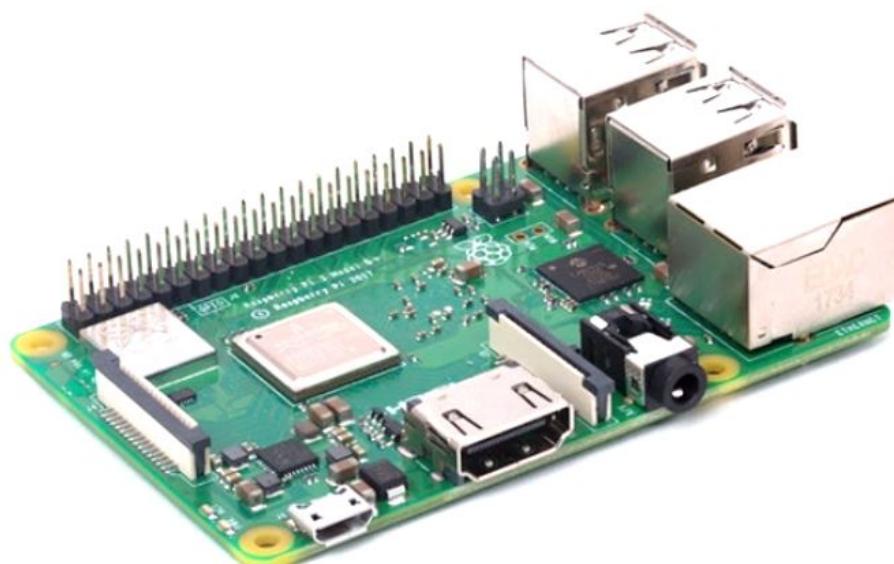


Figura 3.13: Raspberry Pi 3 Model B+ [29].

## Análisis y Diseño

Las características, más detalladamente, de esta tarjeta son:

Tabla 3.6: Especificaciones Raspberry Pi 3 Model B+ [29].

| Elemento                   | Especificación   |
|----------------------------|--|
| Procesador                 | Broadcom BCM2837B0, Cortex-A53 64-bit SoC @ 1.4GHz   |
| Memoria                    | RAM 1GB LPDDR2 SDRAM   |
| Alimentación<br>de entrada | 5V/2.5A DC vía micro USB<br>5V DC vía GPIO   |
| Conectividad               | 2.4GHz/5GHz IEEE 802.11.b/g/n/ac wireless<br>LAN, Bluetooth 4.2, BLE<br>Gigabit Ethernet (300Mbps) |
| Video y Sonido             | 4 × USB 2.0<br><br>1 × HDMI<br><br>Puerto MIPI DSI display<br>Puerto MIPI CSI cámara               |
| Multimedia                 | H.264, MPEG-4 decode (1080p30)<br>H.264 encode (1080p30)<br>OpenGL ES 1.1, 2.0 graphics            |
| Soporte para tarjetas SD   | Formato Micro SD   |
| Temperatura de operación   | 0–50 °C  |

## Sistema de Localización

La lógica para este subsistema fue diseñada en base al escenario de pruebas propuesto y consiste en tomar las lecturas de las distancias obtenidas por el sensor LiDAR del robot, y obtener la distancia más corta para establecer en ese punto el norte del robot.

También, se calcula el complemento del ángulo en donde se obtuvo la distancia más corta para así leer las distancias restantes. Esta lógica funciona para todos los robots y en caso de uno encontrarse alineado con otro, una de sus distancias deberá ser similar y la suma de las distancias complementarias deberá dar el tamaño total del escenario.

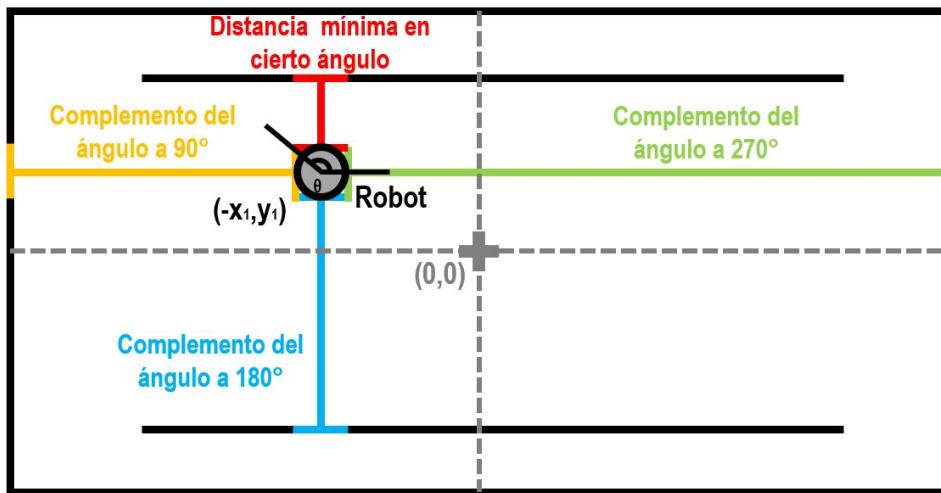


Figura 3.14: Diagrama sistema de localización.

La restricción que se presenta para esta lógica es que los robots deben ubicarse dentro del rectángulo principal y no entre los pasillos que se forman por los bordes y los obstáculos del escenario puesto que no habría forma de saber en cual de los dos lados se encuentran ubicados. La delimitación de la zona en donde pueden iniciar los robots se muestra a continuación (línea roja punteada).

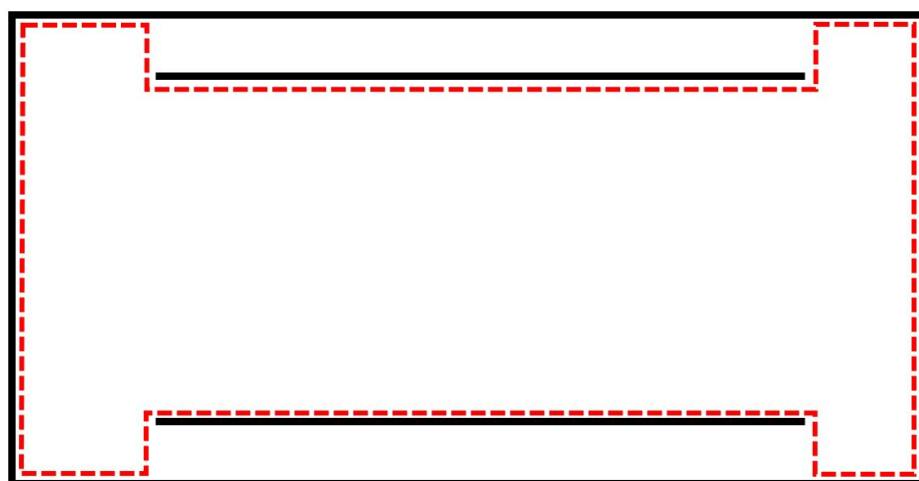


Figura 3.15: Delimitación de posiciones iniciales de los robots.

Para llevar a cabo la simulación de este sistema, se implementó un algoritmo en Python, en donde lo primero que hace es suscribirse a los nodos de lectura del sensor LiDAR del primer robot y llamar a la función “scan\_callback\_tb”, después se hace lo mismo para el segundo y tercer robot. El diagrama de flujo para este código se muestra a continuación.

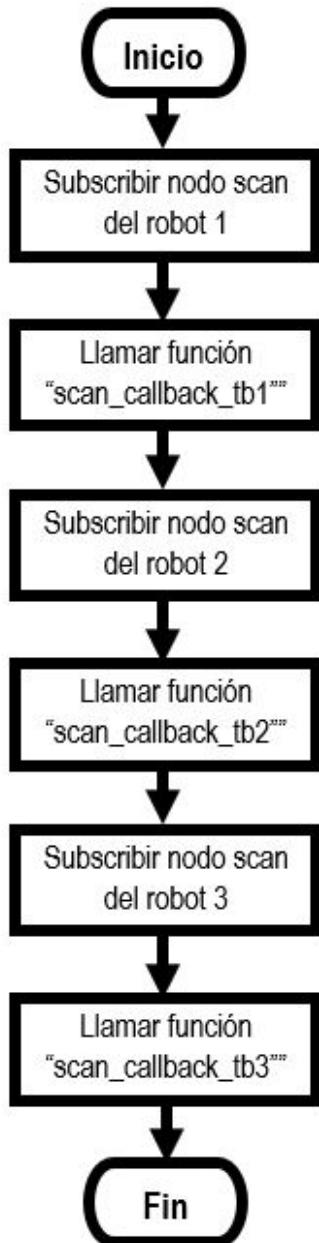


Figura 3.16: Diagrama de flujo del sistema de localización.

La función “can\_callback\_tb” obtiene y almacena en un vector las distancias del sensor LiDAR. Después, ordena los valores de mayor a menor y obtiene el valor e índice de la distancia más corta, crea un nuevo vector con los valores de distancia empezando por el valor obtenido anteriormente. Finalmente, asigna al norte el valor del índice cero, al sur el valor del índice 180, al este el valor del índice 270 y al oeste el valor del índice 90. El diagrama de flujo para esta función se muestra en la siguiente figura.

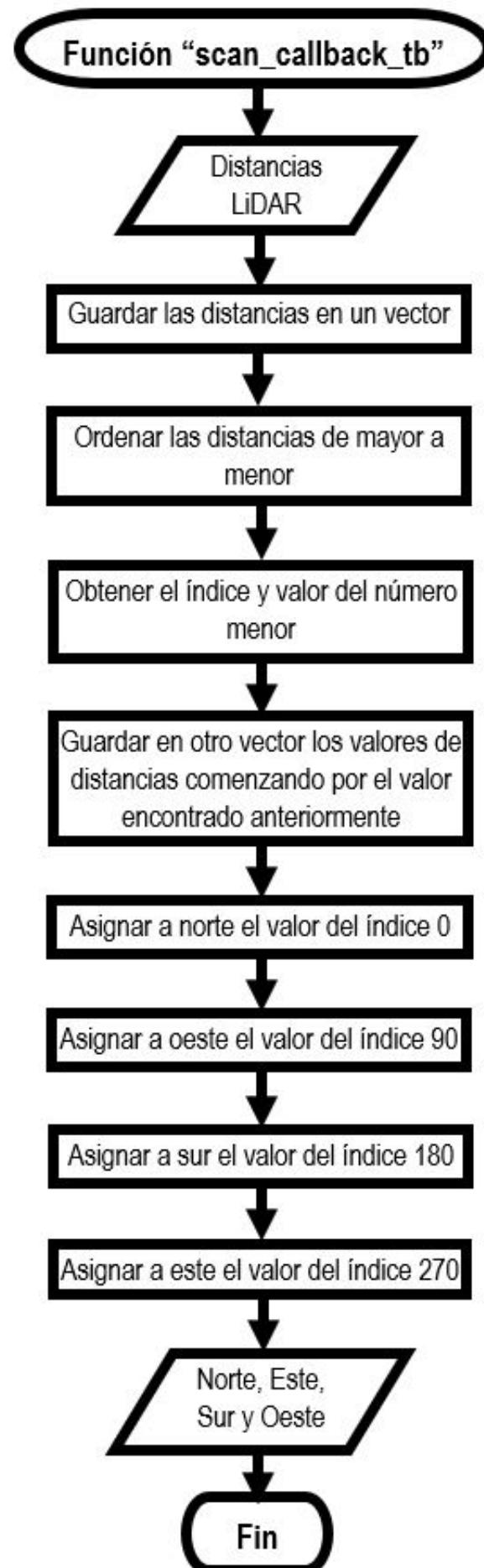


Figura 3.17: Diagrama de flujo de la función "scan\_callback\_tb".

## Sistema de Asignación de Roles

Al trabajar con el conjunto de robots, es necesario asignar quien va a ser el robot líder encargado de seguir la trayectoria dada por el sistema y quienes serán los seguidores. Para determinar cual robot será el líder, se utilizan las distancias euclidianas entre cada robot y el punto inicial de la trayectoria. Así, el que tuviera la distancia más corta, se le asignaría el rol de líder, mientras que los otros se les asignaría el rol de seguidores.

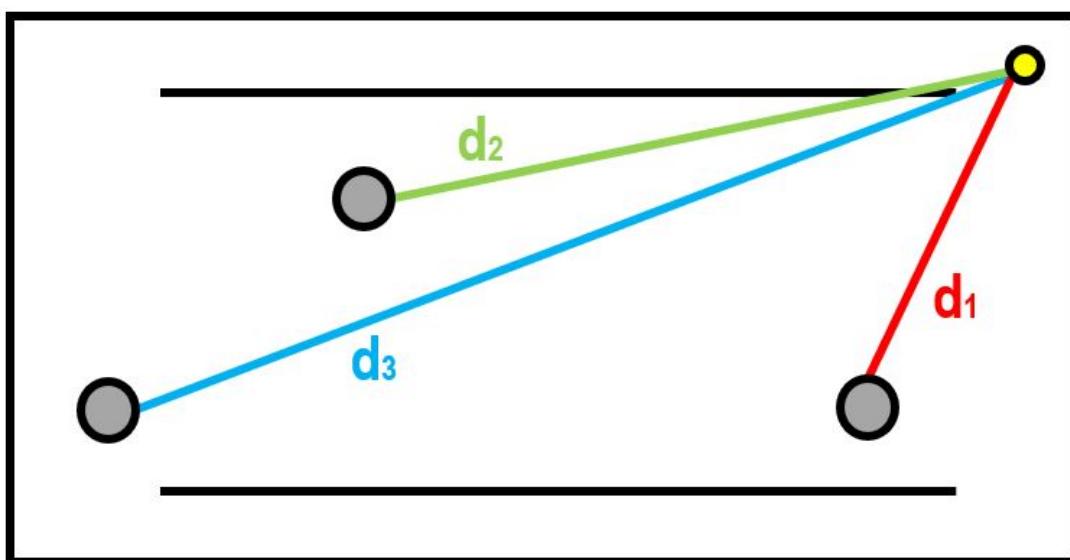


Figura 3.18: Diagrama sistema de asignación de roles.

Para esto, se realiza un código, en donde lo primero que se hace es leer la posición inicial de cada robot, y el punto destino (o punto inicial de la trayectoria). Luego, se calcula la distancia eucliana de cada robot al punto destino y se obtiene el valor mínimo de estas distancias. Posteriormente, al robot con la distancia menor se le asigna el rol de líder y a los dos restantes el rol de seguidor 1 y seguidor 2. Para un mejor entendimiento del sistema se presenta el diagrama de flujo de este en la figura 3.19.

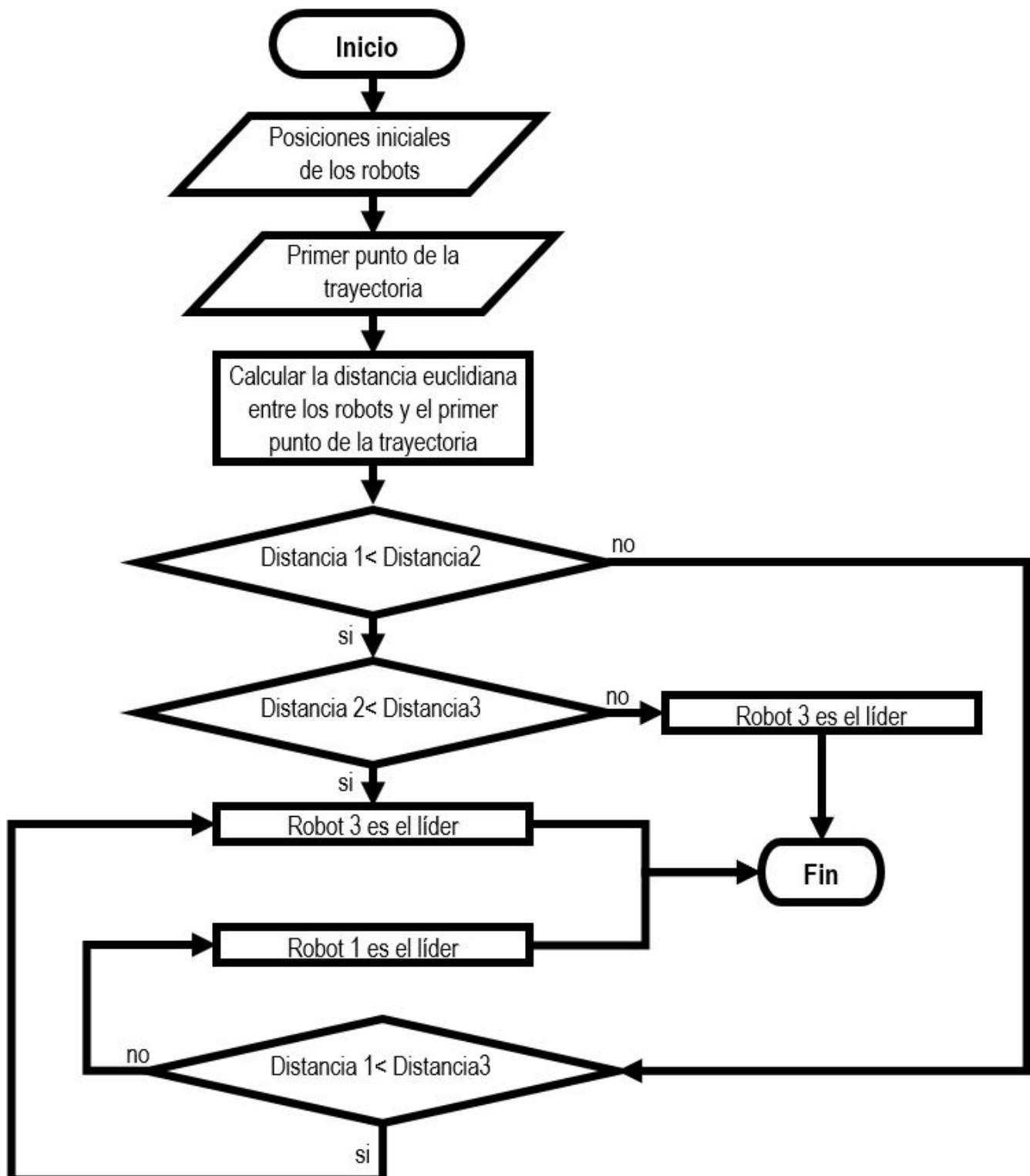


Figura 3.19: Diagrama de flujo del sistema de asignación de roles.

## Sistema de Navegación

En la navegación de robots móviles, se debe considerar la planificación de movimientos como una pieza fundamental para su buen funcionamiento. El objetivo de esta planificación es poder especificar a cada robot que tareas debe hacer, por medio de un lenguaje de alto nivel, y traducir a comandos de bajo nivel esta especificación que les permita llevar a cabo las tareas asignadas, transformando lo interpretado por el robot a movimientos primitivos [30].

El principal problema a abordar se denomina Planificación de Caminos (path planning), clasificado por Th. Fraichard [31], en el cual el entorno es invariante a lo largo del tiempo. A continuación, se muestra un diagrama simple de la planificación de movimiento para los robots, pudiendo observar sus dos componentes: planificación y control.

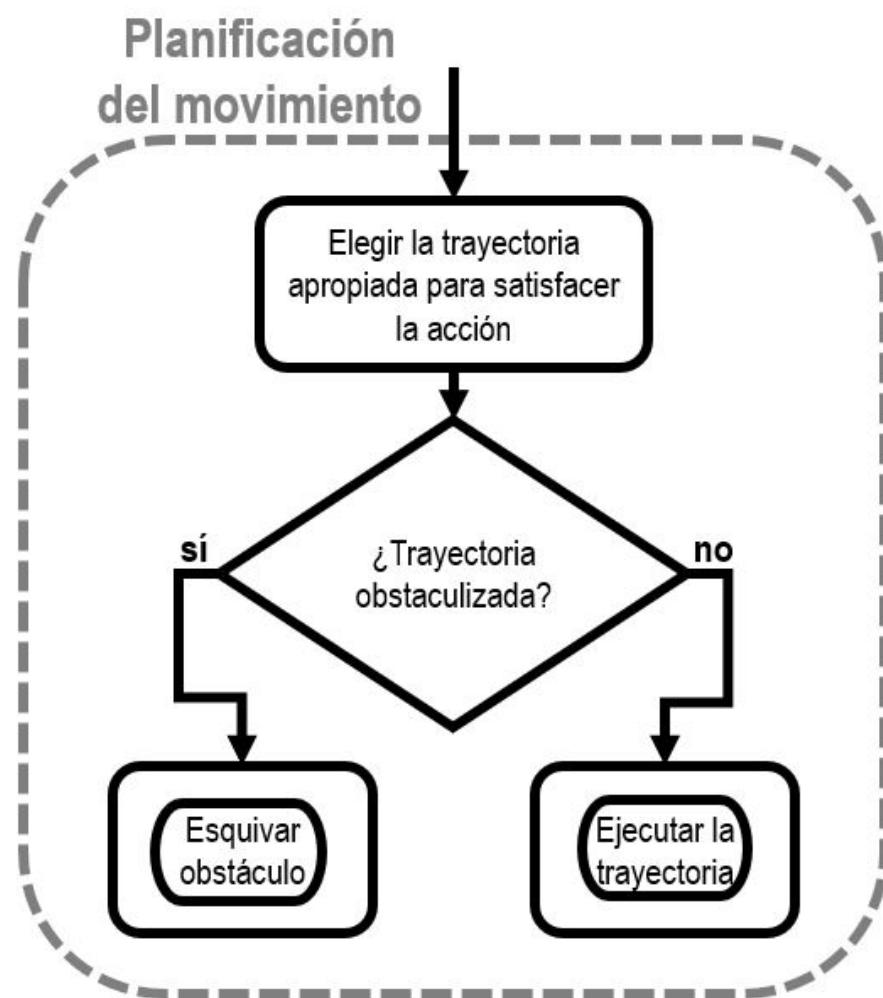


Figura 3.20: Esquema de planeación de movimiento para un robot móvil [31].

En el presente trabajo, se describe esta planificación y control en 2 etapas distintas:

- El seguimiento de trayectorias
- La detección y evasión de obstáculos

## Seguimiento de trayectoria

Se considera que este sistema únicamente asegure la llegada del robot hacia el punto objetivo. Para abordar el seguimiento de trayectorias, se debe identificar primero las características de los robots que restringen la geometría de las trayectorias.

- Sistema no-holonómico. El movimiento de los robots es limitado.
- Posición del eje. El eje de rotación de las ruedas puede encontrarse desplazado horizontalmente respecto al centro del robot.
- Límite de velocidades. La velocidad en cada rueda es limitada a un intervalo.

## Control P

Se parte del modelo cinemático del robot diferencial descrito en la sección 2.3.7. En donde las ecuaciones del movimiento del robot están dadas por:

$$\begin{bmatrix} \dot{x} \\ \dot{y} \\ \dot{\theta} \end{bmatrix} = \begin{bmatrix} \cos(\theta) & 0 \\ \sin(\theta) & 0 \\ 0 & 1 \end{bmatrix} \begin{bmatrix} V \\ W \end{bmatrix} \quad (3.1)$$

Una de las ventajas que proporciona ROS, es poder simplificar de cierto modo el control del robot, ya que las librerías para los Turtlebot3 Burger, brindan un método simple donde únicamente se proporciona el valor de la velocidad linear y angular que tendrá el robot durante su desplazamiento.

El control realizado es una versión similar al control LV (Linear Velocity Lyapunov approach) [30], cuyas expresiones son las siguientes:

$$v = \gamma e \quad (3.2)$$

$$w = \gamma \sin(\alpha) + \frac{h\theta \sin(\alpha)}{a} + \beta \alpha \quad (3.3)$$

En donde:

$v$ =Velocidad lineal

$w$ =Velocidad angular

$e$ =Distancia euclíadiana al punto destino

$\theta$ =Ángulo entre el eje horizontal y la recta que une el centro del robot y el punto destino

$\alpha$ =Diferencia entre  $\theta$  y la orientación del robot

Una adaptación realizada al control LV original, consiste en la consideración de la velocidad máxima alcanzada por los robots, pues originalmente, este considera la velocidad proporcional a la distancia, la cual es un problema ante distancias muy grandes pues superaría el límite de velocidad de los robots.

Otra cosa a considerar, es el hecho de que este tipo de control es aplicado a cada una de las ruedas del motor. Sin embargo, debido a que las librerías de ROS permiten controlar las velocidades del robot sin necesidad de controlar cada motor por separado, se pueden simplificar las expresiones del control LV quedando de la siguiente forma:

$$V = K_v d \quad (3.4)$$

$$W = K_w(\psi - \theta) = K_w\alpha \quad (3.5)$$

En donde:

$K_v$ =Ganancia de la velocidad lineal

$K_w$ =Ganancia de la velocidad angular

$d$ =Distancia euclíadiana entre los puntos

$\theta$ =Orientación del robot

$\psi$ =Ángulo entre el eje horizontal y la recta que une el centro del robot con el punto destino

$\alpha$ =Diferencia entre la orientación del robot y  $\phi$

En el siguiente diagrama se observa lo antes mencionado.

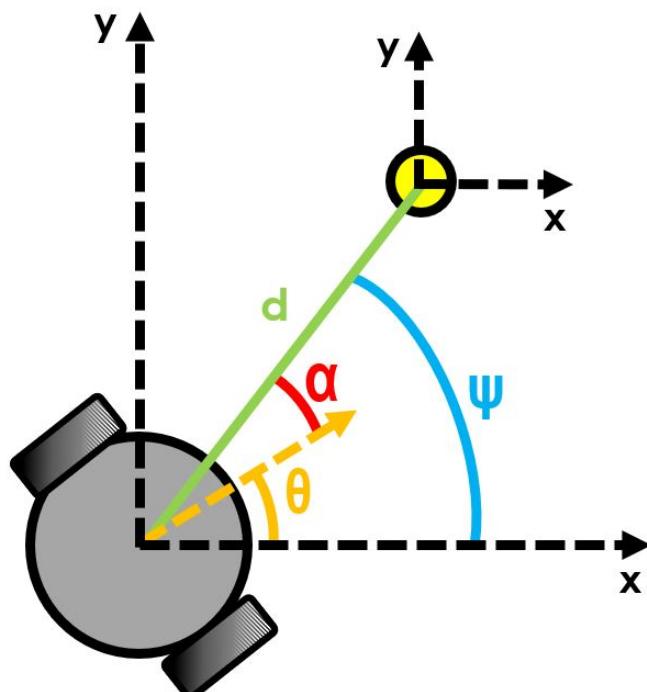


Figura 3.21: Parámetros considerados para el control del seguimiento de trayectoria [30].

## Validación

Como parte del diseño del controlador, es importante validar su correcto funcionamiento ante diversas entradas, y observar el comportamiento del error de la distancia  $d$  y el ángulo  $\alpha$  con varios valores de  $K_v$  y  $K_w$ . Para observar el comportamiento de las salidas del controlador, como herramienta auxiliar se utiliza el software MATLAB. El diagrama de bloques utilizado se muestra en la figura a continuación; en él, se puede observar que se coloca el punto al que se desea llegar como entrada del controlador, este realiza el debido procesamiento para dar las señales de entrada a la planta del sistema. Para modificar la posición inicial del robot, se cambian las condiciones iniciales del integrador.

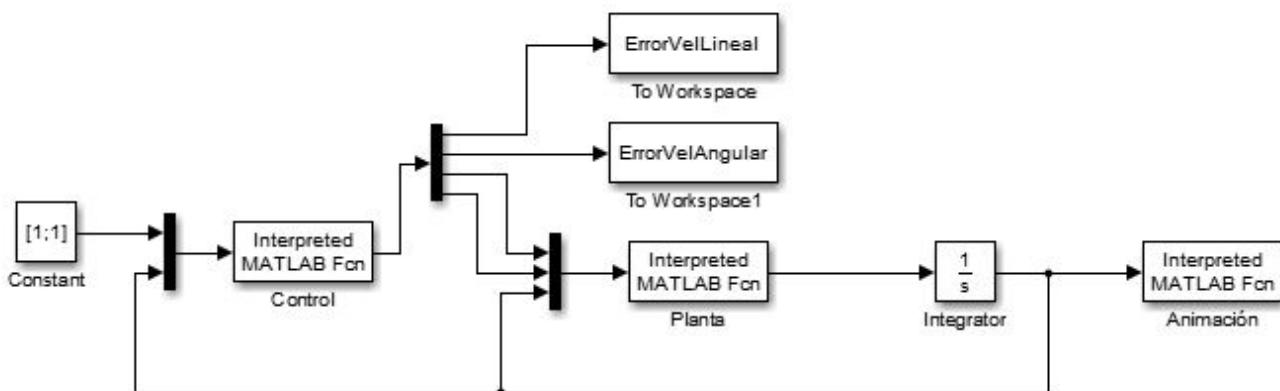


Figura 3.22: Diagrama de bloques para el seguimiento de trayectorias (Control P).

La primera prueba se realiza con una ganancia unitaria tanto para la velocidad lineal como para la velocidad angular. Se parte del punto  $(0,0)$  con una orientación de  $\frac{\pi}{2}$  y se desea llegar al punto  $(1,1)$ .

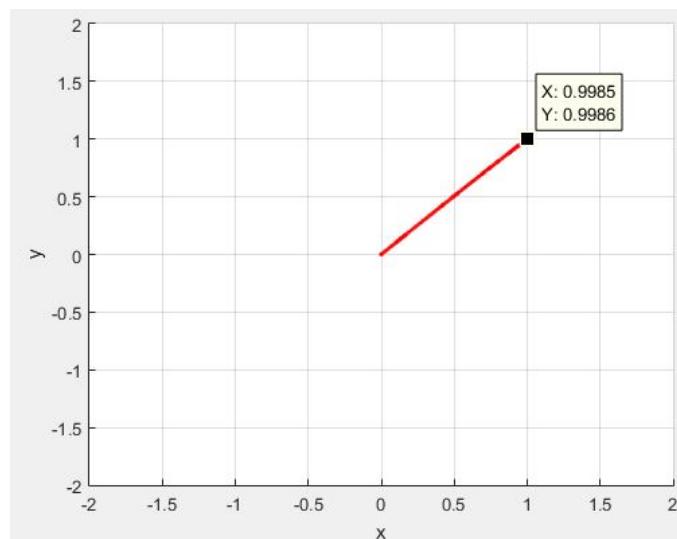


Figura 3.23: Trayectoria seguida de  $(0,0)$  a  $(1,1)$  para  $K_v = 1$  y  $K_w = 1$  (Control P).

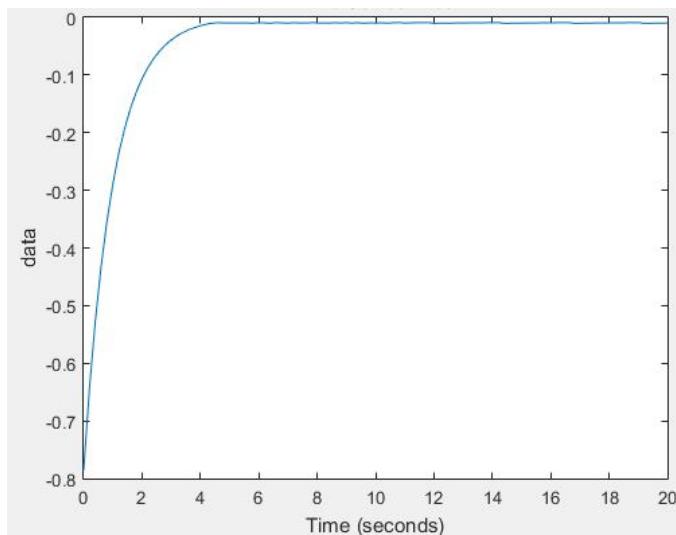


Figura 3.24: Error en la velocidad angular para  $K_v = 1$  y  $K_w = 1$  (Control P).

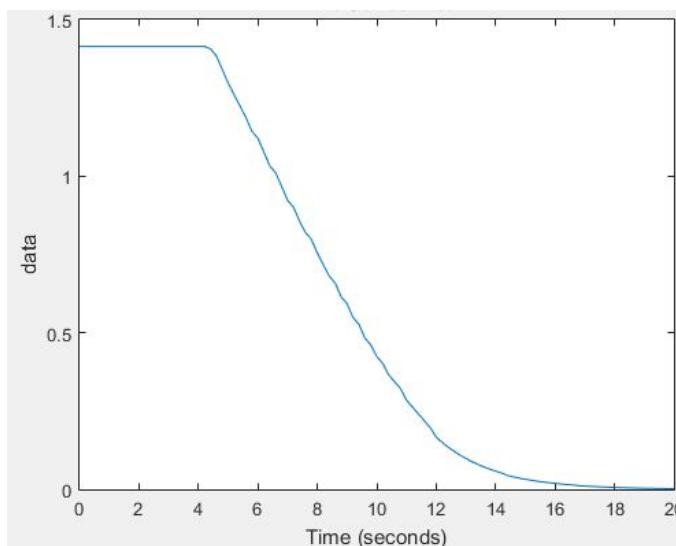


Figura 3.25: Error en la velocidad lineal para  $K_v = 1$  y  $K_w = 1$  (Control P).

Para estas condiciones, puede observar que el robot se aproxima bien al punto deseado y que los errores tienden a cero; sin embargo, el tiempo que le toma al robot llegar al punto es mucho. Por lo tanto, se realizó una prueba con ganancias iguales a 5 para ambas velocidades, con las misma posición inicial y yendo al mismo punto deseado (figuras 3.26, 3.27 y 3.28).

Al aumentar la ganancia de la velocidad angular, el tiempo que le toma al robot llegar al ángulo deseado es menor pero entre mayor sea dicha ganancia, las perturbaciones en la gráfica del ángulo son mayores. En consecuencia, para la tercera prueba se propone una ganancia de 3 en la velocidad angular ya que esta ayudaría a reducir el tiempo de ejecución de una forma significativa sin generar tantas perturbaciones como con una ganancia igual a 5. Y, la ganancia en la velocidad lineal se propone igual a 10 para que el robot se desplace al punto un poco más rápido, sin embargo, debido a la limitación en la velocidad lineal, no se obtiene una gran diferencia con respecto a las otras ganancias propuestas (figuras 3.29, 3.30 y 3.31).

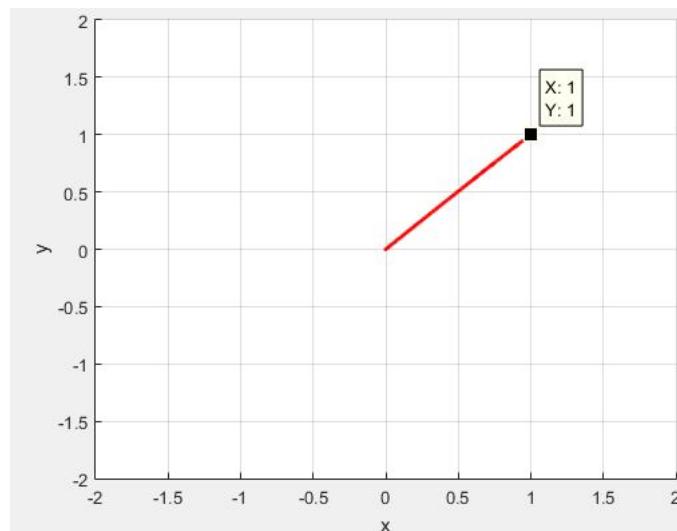


Figura 3.26: Trayectoria seguida de  $(0,0)$  a  $(1,1)$  para  $K_v = 5$  y  $K_w = 5$  (Control P).

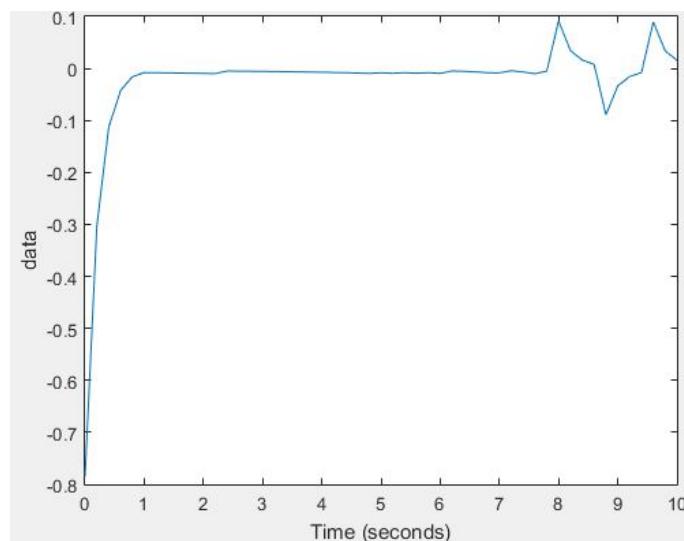


Figura 3.27: Error en la velocidad angular para  $K_v = 5$  y  $K_w = 5$  (Control P).

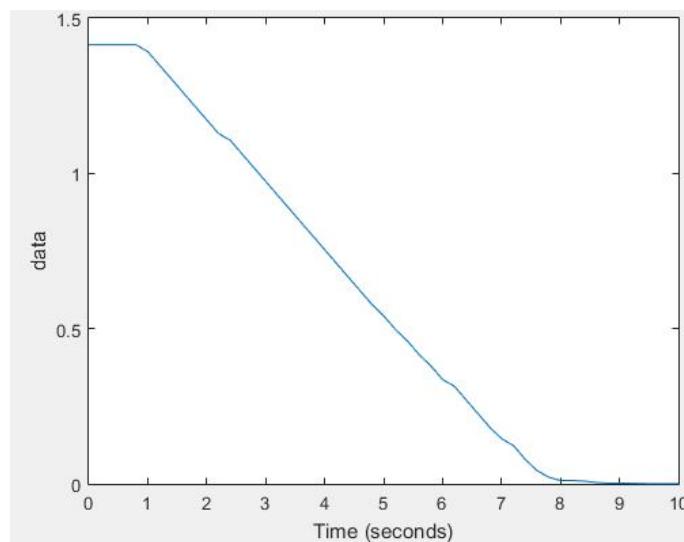


Figura 3.28: Error en la velocidad lineal para  $K_v = 5$  y  $K_w = 5$  (Control P).

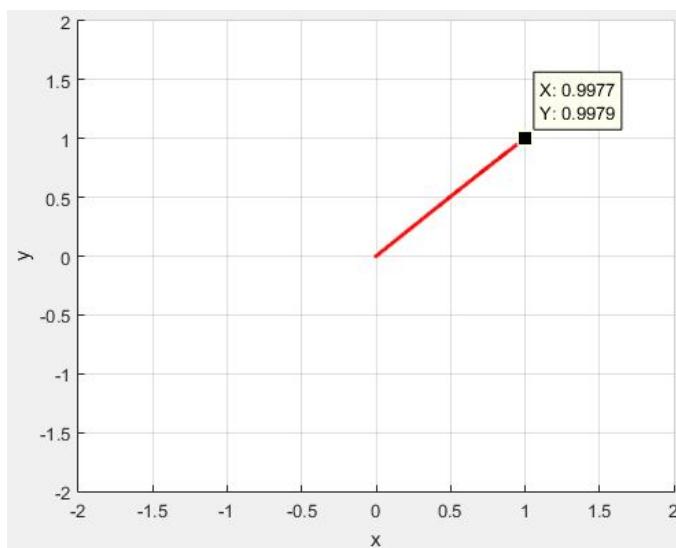


Figura 3.29: Trayectoria seguida de  $(0,0)$  a  $(1,1)$  para  $K_v = 10$  y  $K_w = 3$  (Control P).

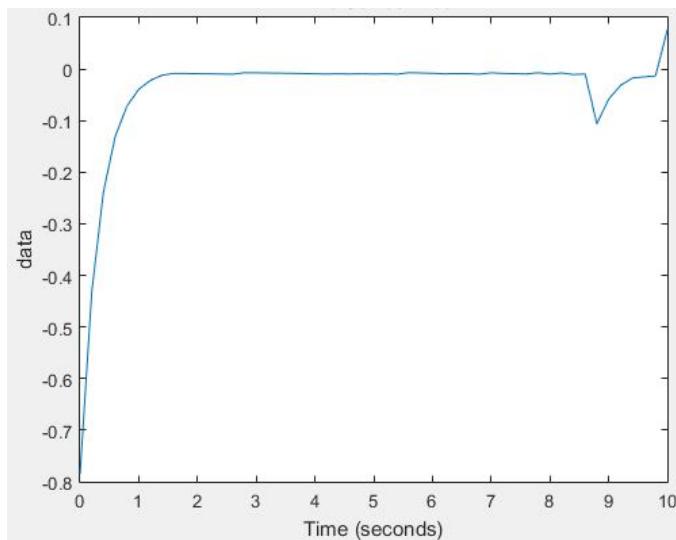


Figura 3.30: Error en la velocidad angular para  $K_v = 10$  y  $K_w = 3$  (Control P).

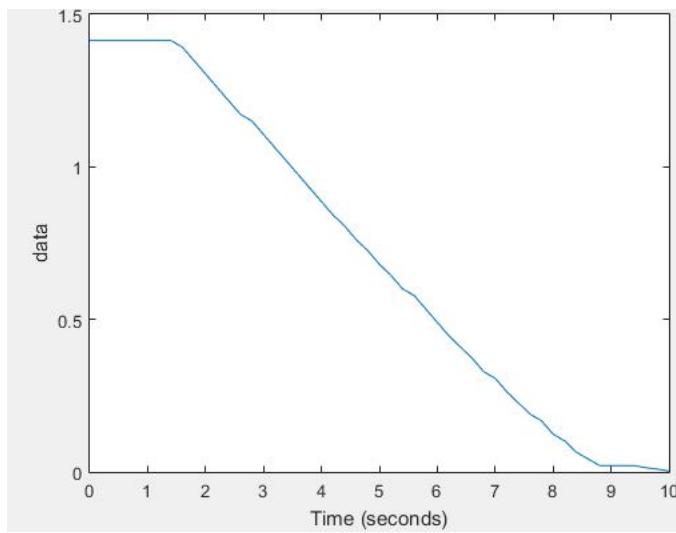


Figura 3.31: Error en la velocidad lineal para  $K_v = 10$  y  $K_w = 3$  (Control P).

En las gráficas se puede observar que las últimas ganancias propuestas generan el comportamiento esperado debido a que el tiempo que le toma al robot llegar al ángulo deseado es menor en comparación con el tiempo que le toma llegar cuando la ganancia es unitaria y las perturbaciones son mínimas. Asimismo, aunque no llega al punto (1,1) exactamente, el error es menor al 2 %.

Finalmente, se hace una prueba más para esos valores de ganancia pero cambiando la posición inicial a (-1,1) con una orientación de  $\pi$  y fijando el punto deseado en (1,1).

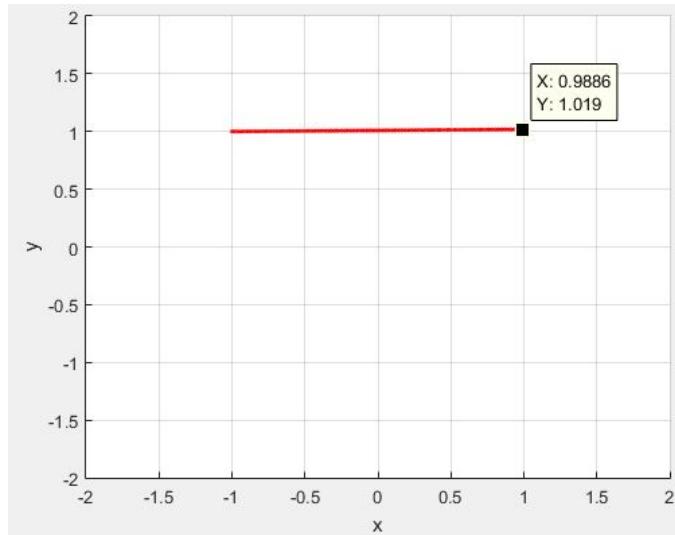


Figura 3.32: Trayectoria seguida de (-1,1) a (1,1) para  $K_v = 10$  y  $K_w = 3$  (Control P).

Se puede observar que estas constantes funcionan adecuadamente y, aunque existe un error en el punto final de llegada, este se encuentra dentro la tolerancia especificada en los requerimientos (5cm).

Una vez comprobado que el sistema funciona de manera correcta, se procede a su implementación en Python. En el código, por medio de los valores de posición obtenidos del parámetro de odometría del robot, se puede conocer la orientación y posición que tiene el robot en todo momento con respecto al marco de referencia fijo y, el punto destino sería previamente asignado. Para las pruebas se asignó una secuencia de puntos a los cuales debe dirigirse el robot, y el programa se encarga de calcular la distancia euclídea  $d$  y el ángulo  $\alpha$  para posteriormente calcular, con las expresiones anteriormente descritas, el valor de la velocidad linear y angular. Dichas velocidades se publican en el nodo correspondiente para que el robot comience a moverse (la velocidad linear máxima que se le puede asignar al robot es de 0.22 m/s y la velocidad angular máxima de 2.84 rad/s). Para un mejor entendimiento del algoritmo, se presenta el diagrama de flujo de la figura 3.33.

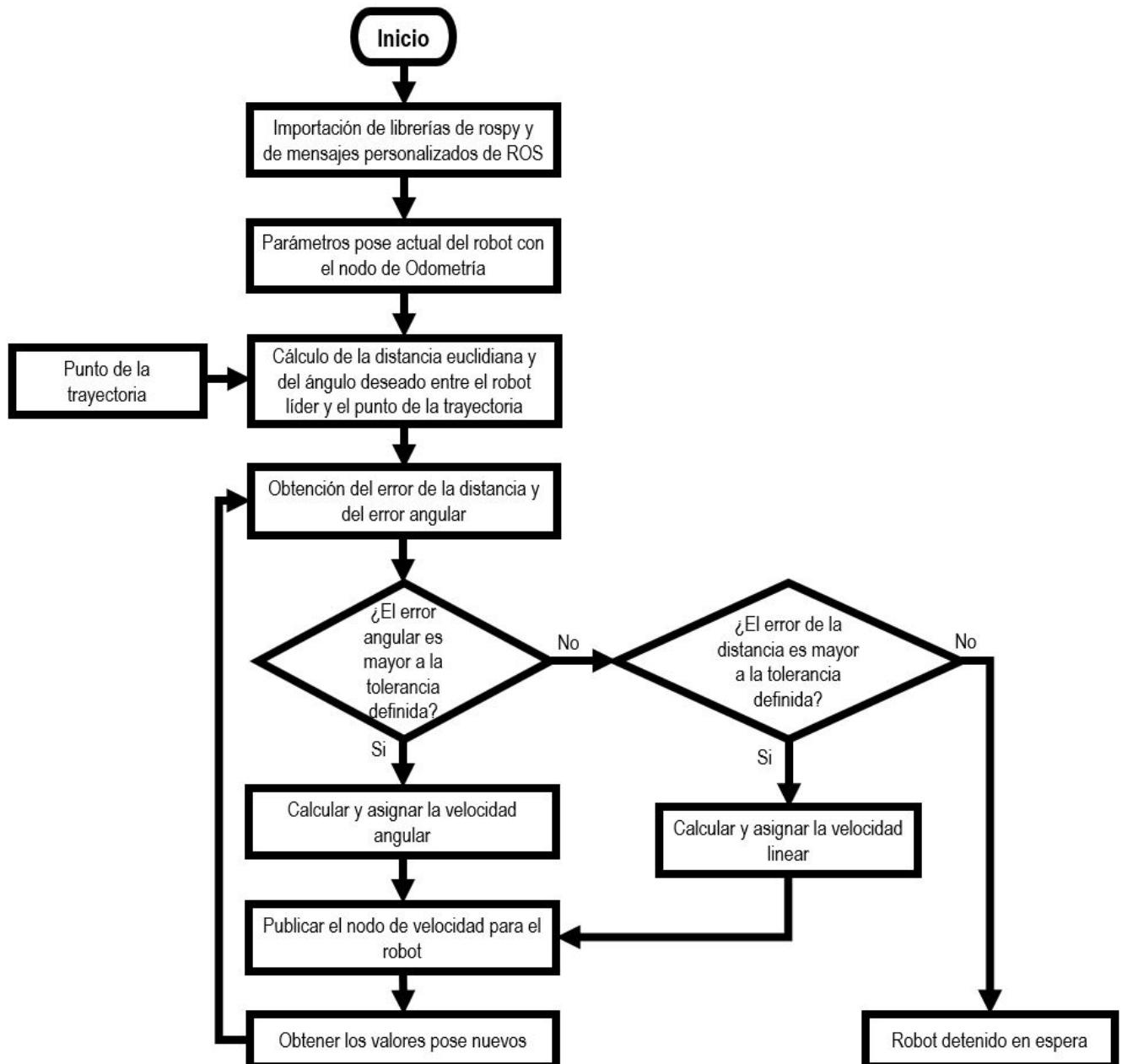


Figura 3.33: Diagrama de flujo del sistema de seguimiento de trayectoria (Control P).

Para el seguimiento de trayectoria, además, se optó por realizarlo en dos pasos, los cuales consisten en primero rotar en dirección a donde debe dirigirse, para después avanzar hacia el punto destino.

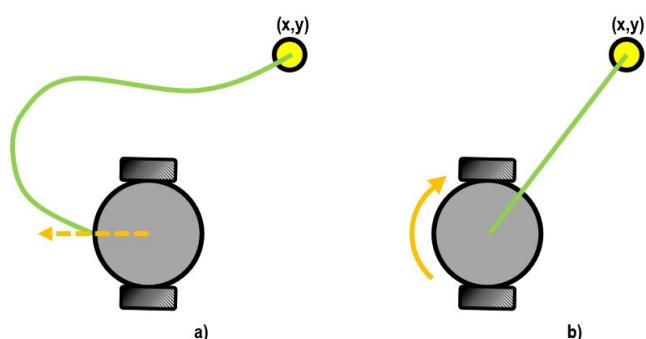


Figura 3.34: Movimiento en dos pasos para dirigirse a un punto.

## Control Difuso

Para este sistema, también se diseñó un controlador difuso de lazo cerrado, el cual, tiene como objetivo que el robot se desplace a un punto dado en coordenadas rectangulares dentro del escenario de pruebas.

Se utilizó un modelo de inferencia difusa tipo Mamdani para el diseño del controlador, el cual tiene como entrada el error del ángulo y la distancia euclídea del robot respecto al punto al que se quiere llegar y como salida la velocidad angular y lineal del robot.

El primer conjunto difuso de entrada es llamado “Error de ángulo”, cuyo universo de discurso va de  $-3.15$  a  $3.15$  y está compuesto por cinco funciones de membresía de forma triangular. Sus variables lingüísticas son: negativo grande (NG), negativo chico (NC), cero (cero), positivo chico (PC) y positivo grande (PG).

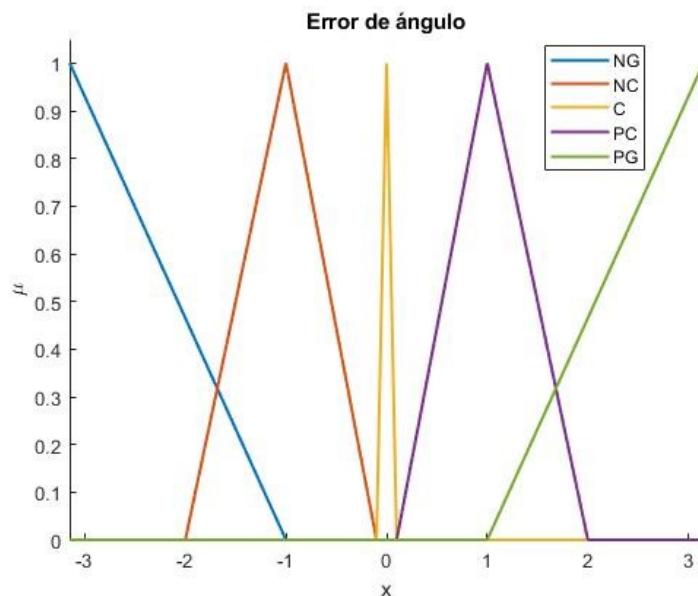


Figura 3.35: Conjunto difuso “Error de ángulo”.

El segundo conjunto difuso de entrada se llama “Distancia euclídea”, lo conforman tres funciones de membresía, dos de forma triangular y una trapezoidal y, su universo de discurso va desde 0 hasta 3.2, debido a que 3.2 m sería la máxima distancia que puede medir el robot dentro del escenario de pruebas. Sus variables lingüísticas son: cero, chica y grande.

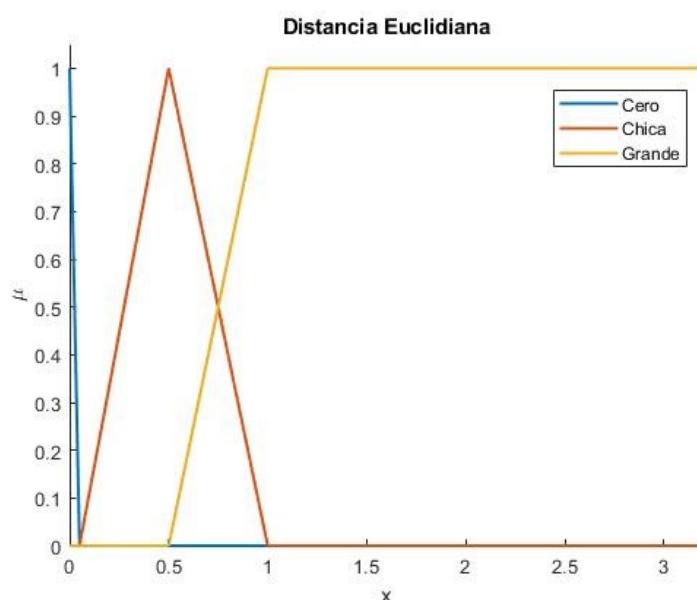


Figura 3.36: Conjunto difuso “Distancia euclidiana”.

El primer conjunto difuso de salida se nombra “Velocidad angular”, su universo de discurso va de  $-2.84$  a  $2.84$ , debido a que la velocidad angular máxima del robot es de  $2.84$  rad/s. Está compuesto por cinco funciones de membresía de forma triangular y sus variables lingüísticas son: derecha grande (DG), derecha chica (DC), cero (cero), izquierda chica (IC) e izquierda grande (IG).

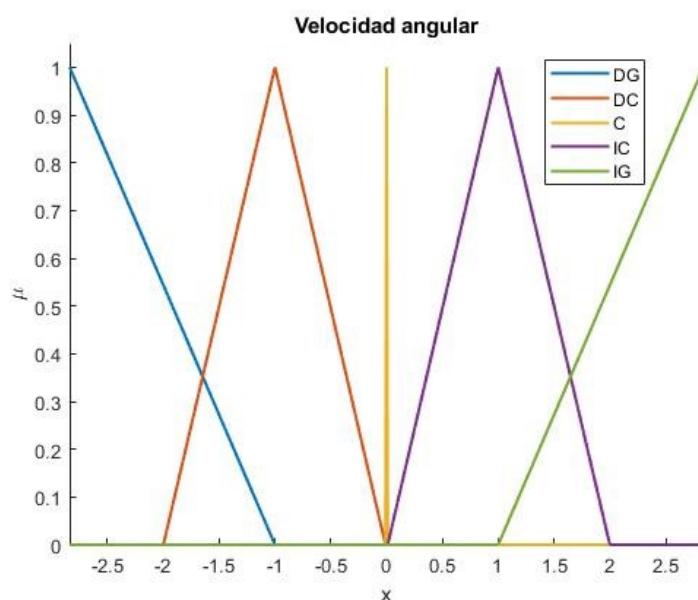


Figura 3.37: Conjunto difuso “Velocidad angular”.

El último conjunto difuso de salida se llama “Velocidad lineal”, lo conforman tres funciones de membresía forma triangular y sus variables lingüísticas son: cero, lento y rápido. Su universo de discurso va desde  $0$  a  $0.22$ , esto se definió de acuerdo con la velocidad lineal máxima del robot que es de  $0.22$  m/s.

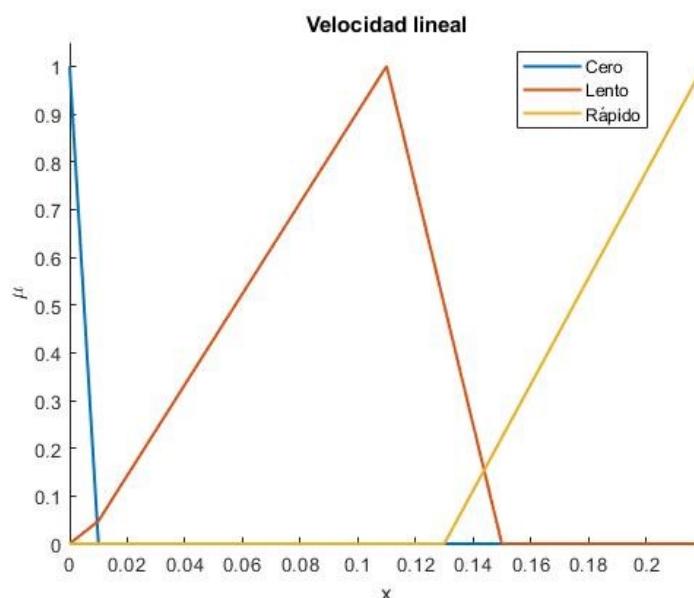


Figura 3.38: Conjunto difuso “Velocidad lineal”.

Las reglas de inferencia difusa que se proponen para las salidas “Velocidad angular” y “Velocidad lineal” se muestran a continuación. El método de defusificación utilizado es el del centroide.

- SI** Error ángulo:NG Y Dist.euclíadiana:Cero, **ENTONCES** Vel.angular:C, Vel.lineal:Cero
- SI** Error ángulo:NG Y Dist.euclíadiana:Chica, **ENTONCES** Vel.angular:DG, Vel.lineal:Cero
- SI** Error ángulo:NG Y Dist.euclíadiana:Grande, **ENTONCES** Vel.angular:DG, Vel.lineal:Cero
- SI** Error ángulo:NC Y Dist.euclíadiana:Cero, **ENTONCES** Vel.angular:C, Vel.lineal:Cero
- SI** Error ángulo:NC Y Dist.euclíadiana:Chica, **ENTONCES** Vel.angular:DC, Vel.lineal:Cero
- SI** Error ángulo:NC Y Dist.euclíadiana:Grande, **ENTONCES** Vel.angular:DC, Vel.lineal:Cero
- SI** Error ángulo:C Y Dist.euclíadiana:Cero, **ENTONCES** Vel.angular:C, Vel.lineal:Cero
- SI** Error ángulo:C Y Dist.euclíadiana:Chica, **ENTONCES** Vel.angular:C, Vel.lineal:Lento
- SI** Error ángulo:C Y Dist.euclíadiana:Grande, **ENTONCES** Vel.angular:C, Vel.lineal:Rápido
- SI** Error ángulo:PC Y Dist.euclíadiana:Cero, **ENTONCES** Vel.angular:C, Vel.lineal:Cero
- SI** Error ángulo:PC Y Dist.euclíadiana:Chica, **ENTONCES** Vel.angular:IC, Vel.lineal:Cero
- SI** Error ángulo:PC Y Dist.euclíadiana:Grande, **ENTONCES** Vel.angular:IC, Vel.lineal:Cero
- SI** Error ángulo:PG Y Dist.euclíadiana:Cero, **ENTONCES** Vel.angular:C, Vel.lineal:Cero
- SI** Error ángulo:PG Y Dist.euclíadiana:Chica, **ENTONCES** Vel.angular:IG, Vel.lineal:Cero
- SI** Error ángulo:PG Y Dist.euclíadiana:Grande, **ENTONCES** Vel.angular:IG, Vel.lineal:Cero

## Validación

Para la validación del sistema se realizó la simulación en Simulink. Primero se generó el controlador difuso con ayuda del Toolbox Fuzzy Logic Designer, como se muestra en la figura 3.39, utilizando un controlador tipo Mamdani, con dos entradas, dos salidas y método de defusificación de centroide.

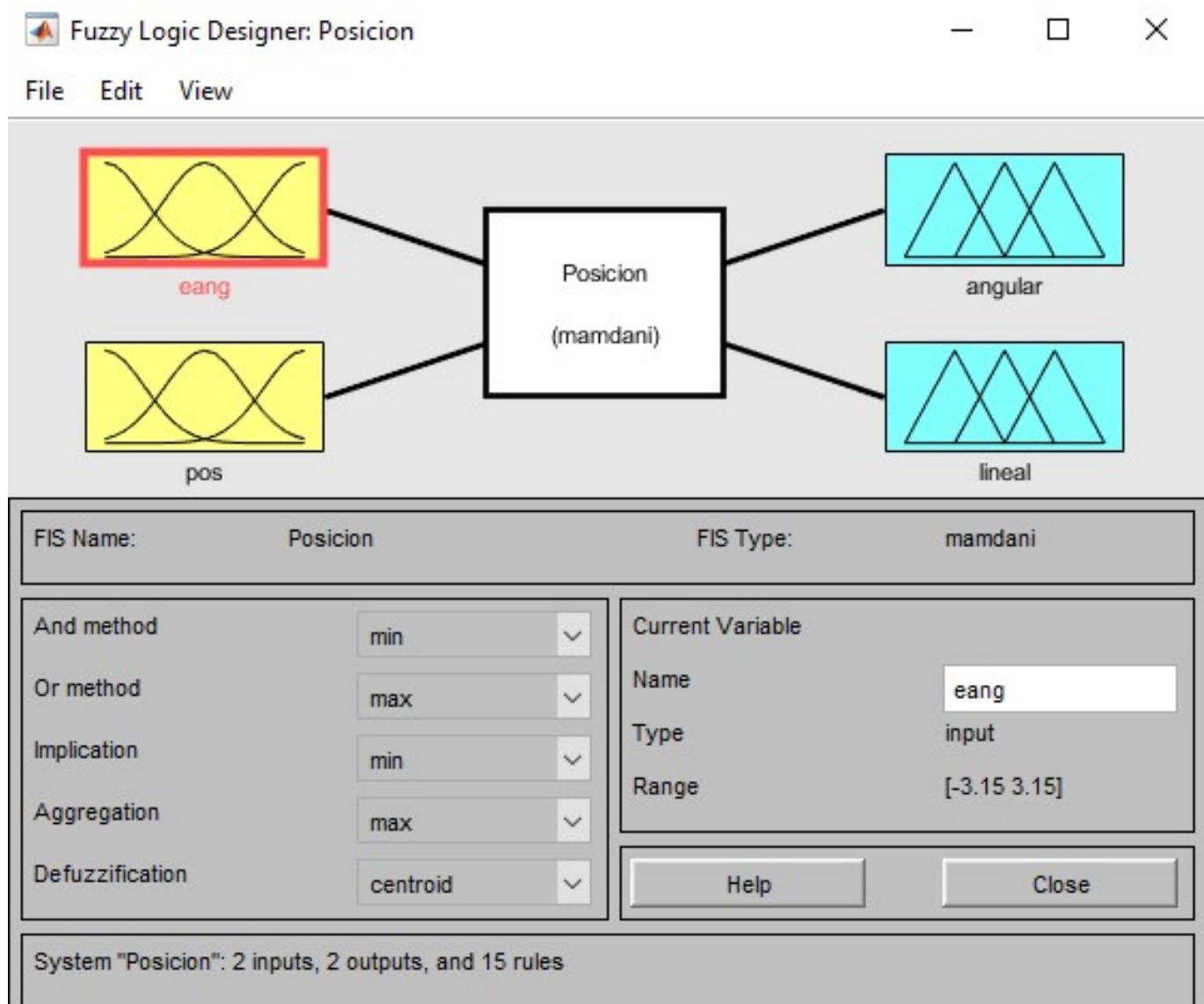


Figura 3.39: Construcción del controlador difuso en el toolbox Fuzzy Logic Designer de MATLAB.

Después se generaron las superficies de control que describen el comportamiento del controlador para el intervalo de entrada (distancia euclíadiana y error de ángulo) en el que opera. La superficie de control de la salida velocidad angular para el controlador propuesto se muestra en la figura 3.40.

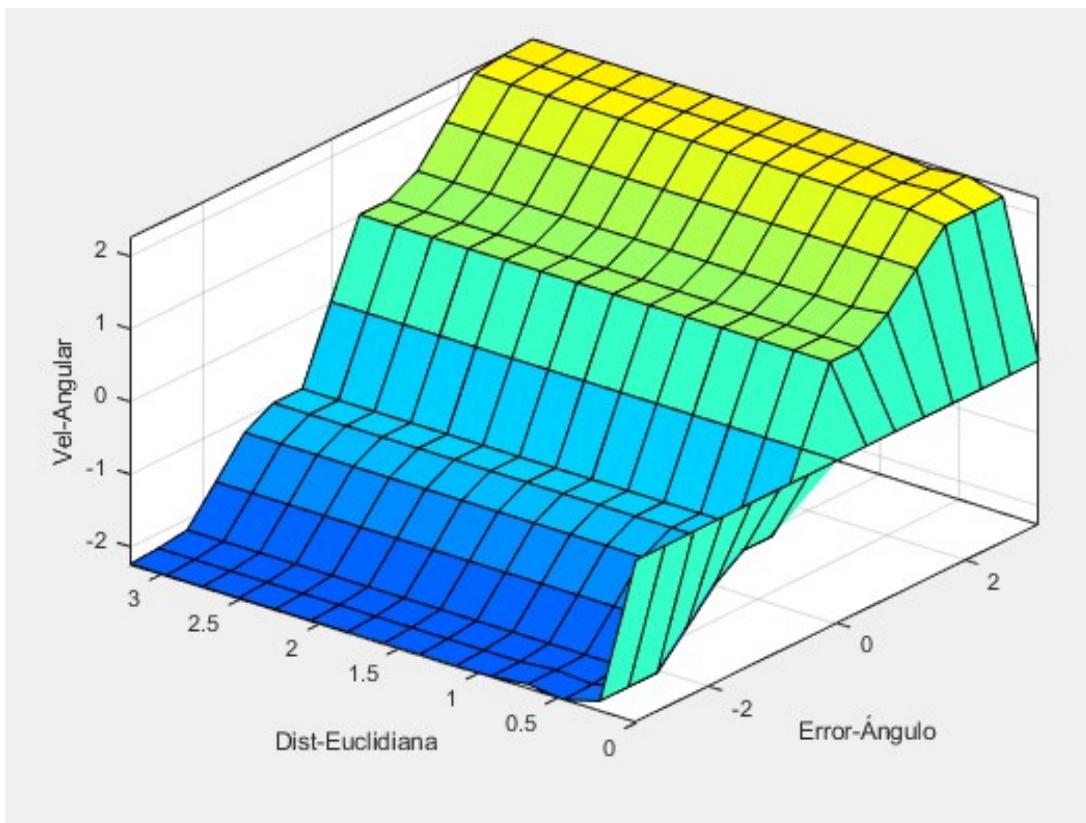


Figura 3.40: Superficie de control de la salida velocidad angular.

La superficie de control de la salida velocidad lineal, se muestra en la figura 3.41. En ella, se pude observar que el comportamiento para esta salida cambia de manera abrupta esto debido a que el robot primero cambia su orientación y después avanza.

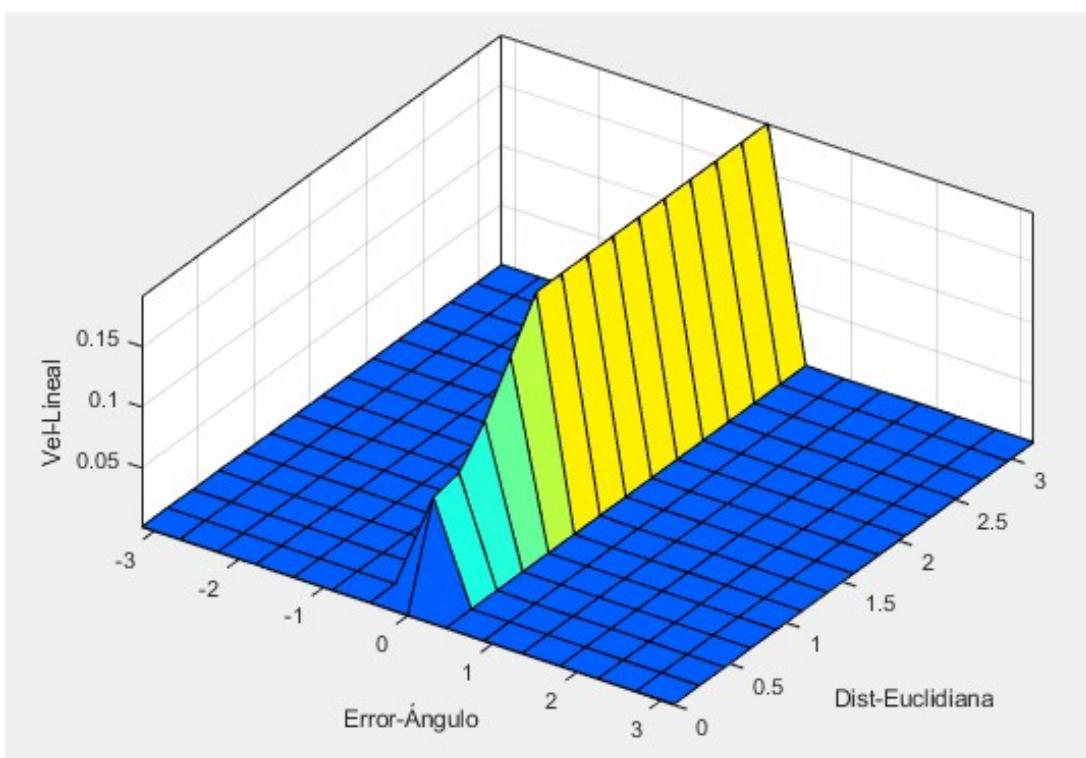


Figura 3.41: Superficie de control de la salida velocidad lineal.

Para visualizar el movimiento del robot, se implementó el siguiente diagrama de bloques en Simulink de MATLAB.

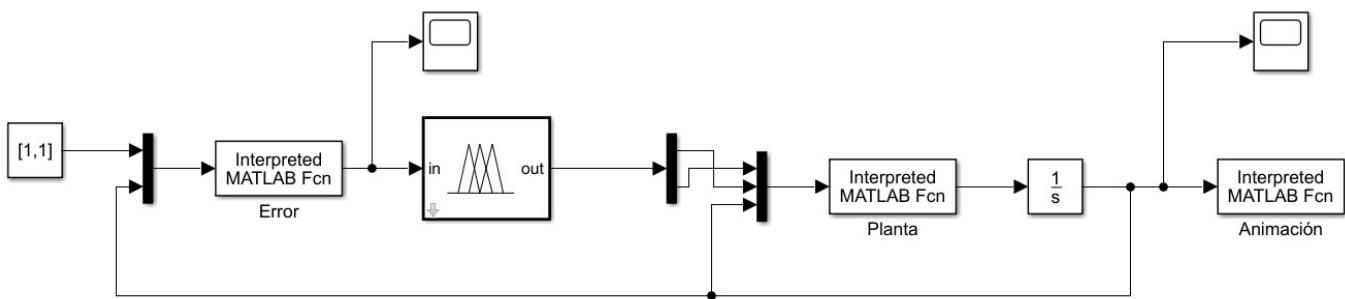


Figura 3.42: Diagrama de bloques para el seguimiento de trayectorias (Control Difuso).

En la simulación, las coordenadas objetivo son (1,1). En la figura 3.43 se muestra el desplazamiento del robot.

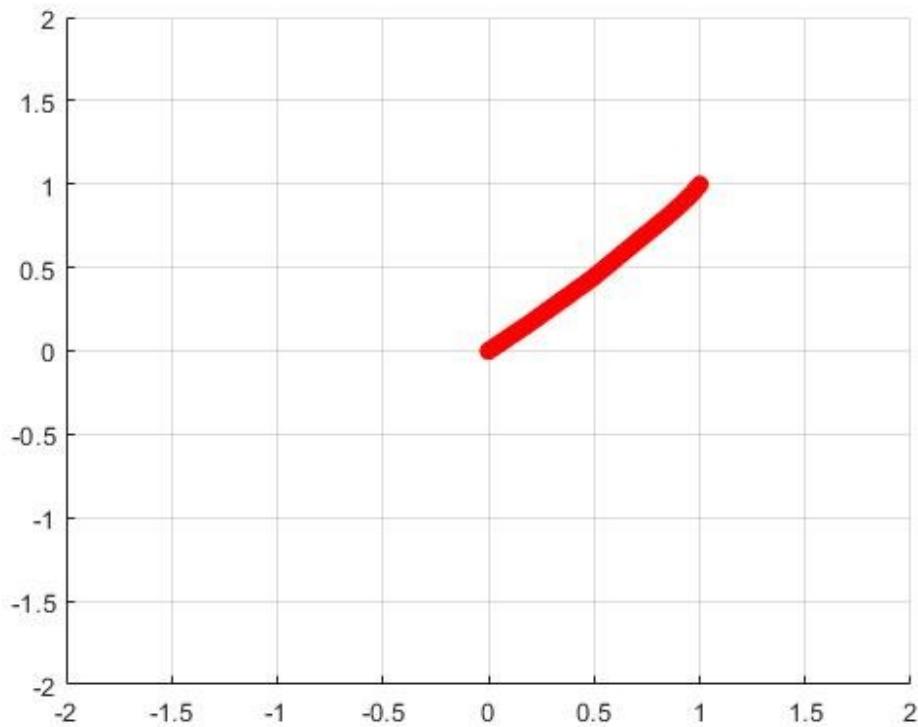


Figura 3.43: Simulación del desplazamiento del robot (Control Difuso) 1a prueba.

En la figura 3.44 se observa la gráfica del error de las velocidades. El error de ángulo se acerca a cero, pero después vuelve a incrementarse debido a que conforme el robot avanza se desvía un poco y debe corregir nuevamente su ángulo.

En la gráfica de posición (figura 3.45) se puede ver que se alcanza al mismo tiempo la posición deseada en ambos ejes, debido a que se sigue la distancia más corta entre los dos puntos, esto es en línea recta.

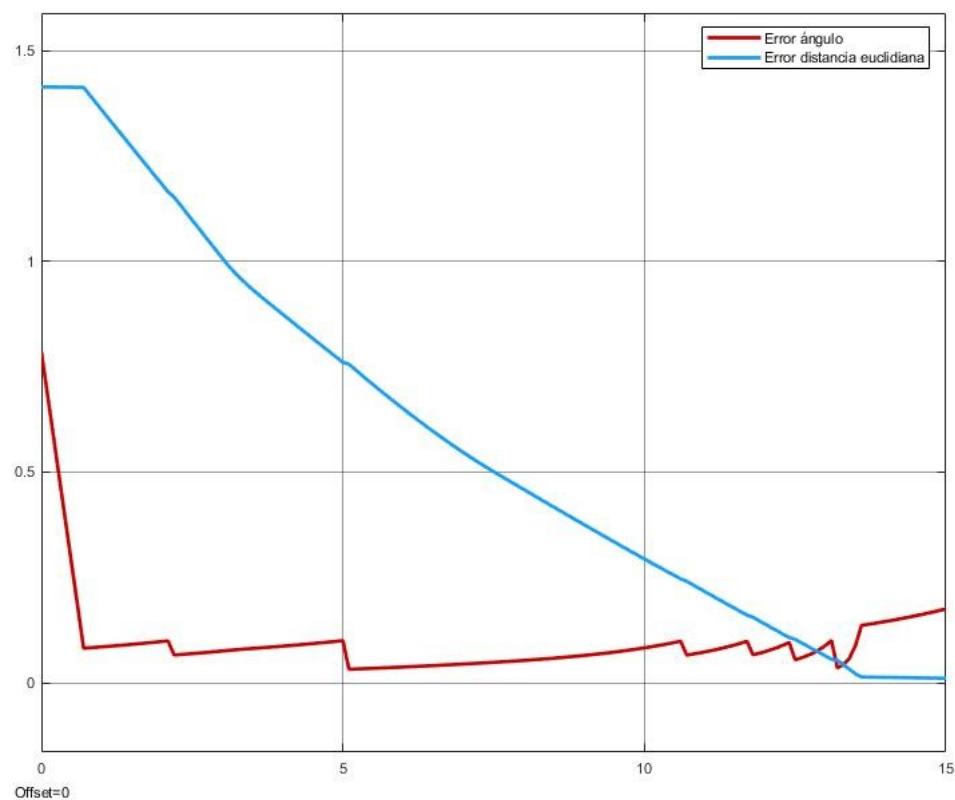


Figura 3.44: Gáfica del error de la velocidad (Control Difuso) 1a prueba.

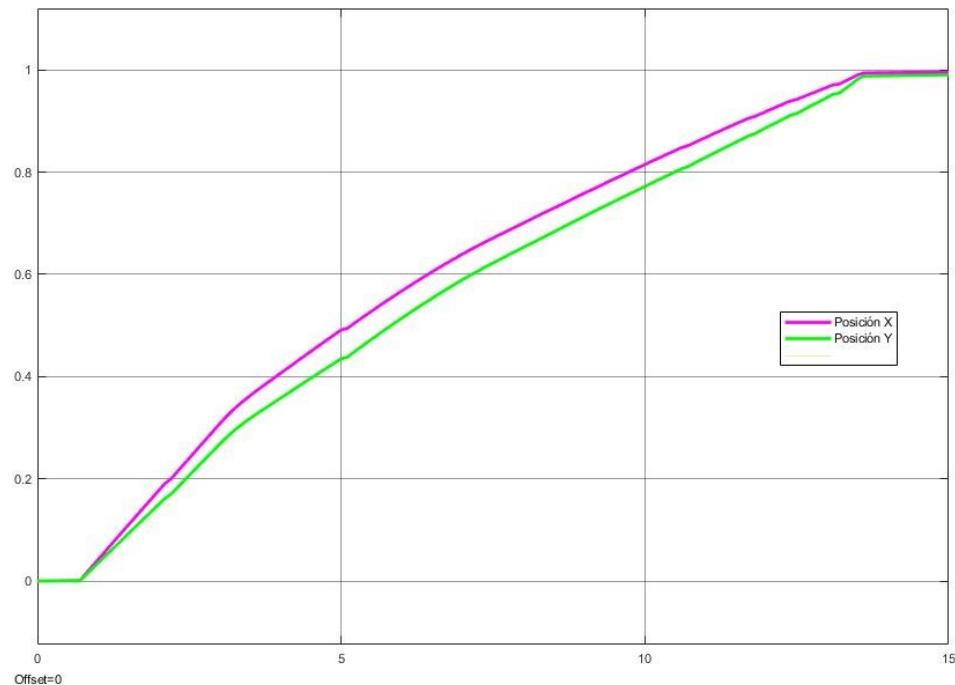


Figura 3.45: Gáfica de la posición del robot (Control Difuso) 1a prueba.

También se simuló para otro punto con coordenadas (0.8, -0.8). En la figura 3.46 se muestra el desplazamiento del robot.

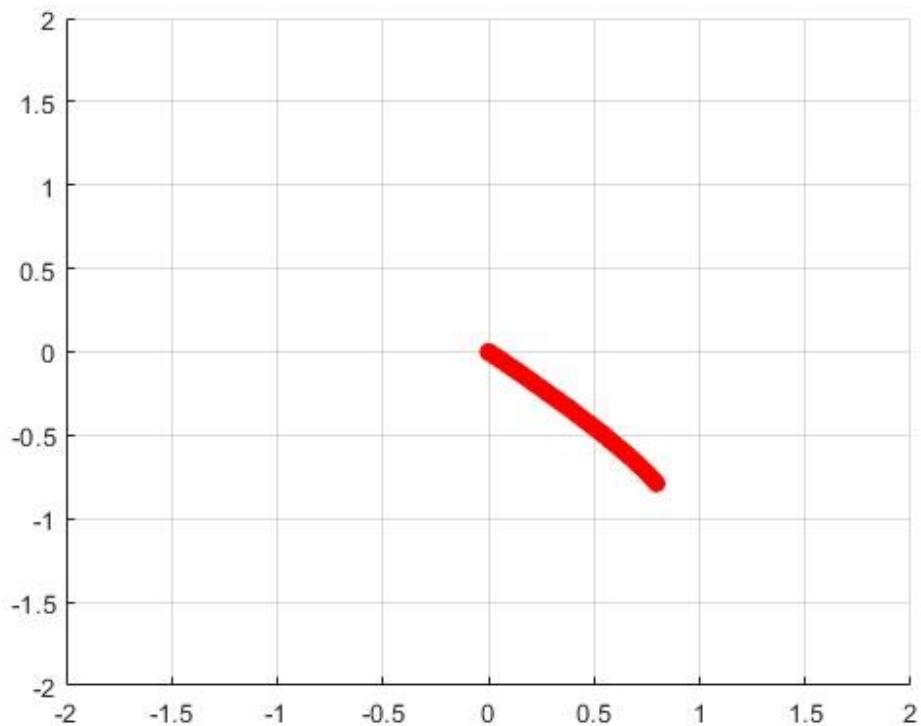


Figura 3.46: Simulación del desplazamiento del robot (Control Difuso) 2a prueba.

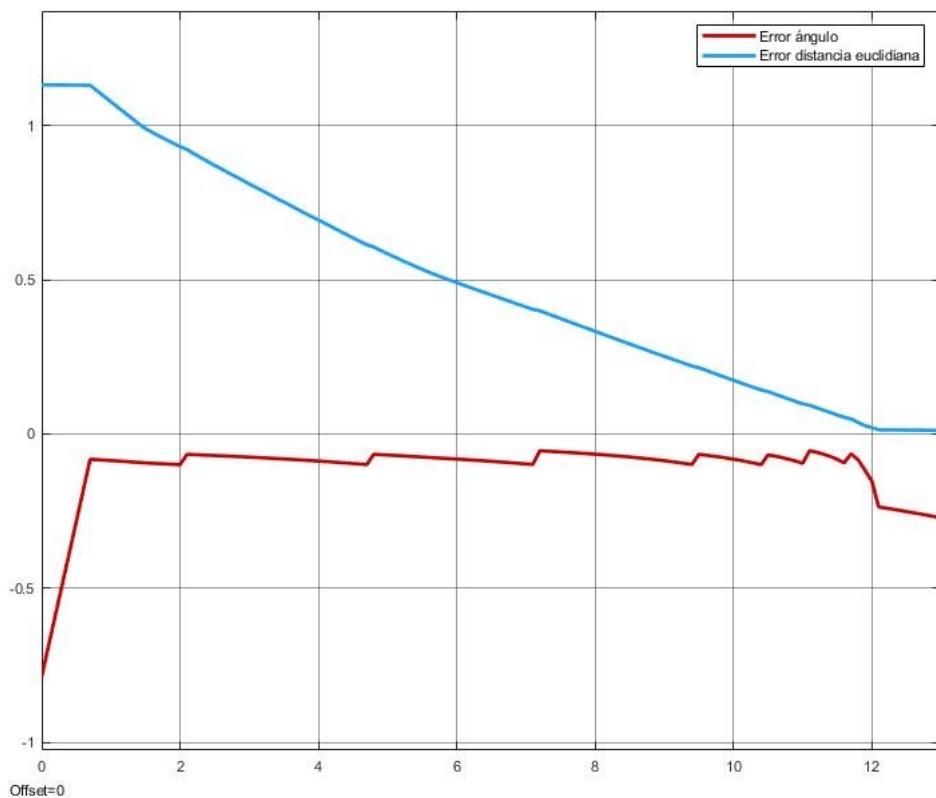


Figura 3.47: Gáfica del error de la velocidad (Control Difuso) 2a prueba.

En la figura 3.47 se observa la gráfica del error en las velocidades. En esta, el error de ángulo al inicio es negativo debido a que en punto objetivo está en otro cuadrante.

La gráfica de posición (figura 3.48) muestra que se alcanza a la vez la posición objetivo en los dos ejes.

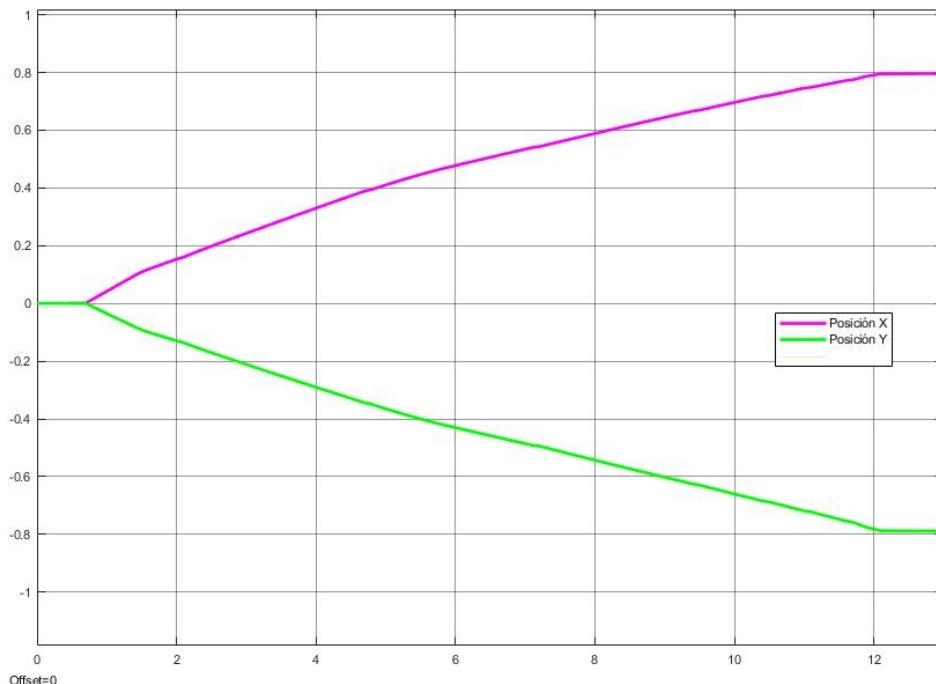


Figura 3.48: Gáfica de la posición del robot (Control Difuso) 2a prueba.

Con los resultados de simulación en MATLAB, se puede concluir que el funcionamiento del controlador es el adecuado y se procede a implementarlo en Python con la ayuda de la librería Scikit-Fuzzy, la cual es una colección de algoritmos de lógica difusa escritos en este lenguaje.

En el código, primero se declaran las clases Posición y Robot, las cuales contienen las variables a utilizar en el algoritmo. Después se declaran dos funciones llamadas controlador y pose\_callback.

La función “controlador” recibe como argumentos el error de ángulo y el error de distancia euclíadiana, y devuelve el valor de velocidad angular y velocidad lineal. Primero, se declaran los conjuntos difusos y después cada una de las funciones de membresía de estos. Después, se obtiene el mínimo entre cada función de membresía de ambas entradas de acuerdo con el valor recibido del error de ángulo y error de distancia euclíadiana para cada una de las quince reglas de inferencia. Posteriormente, se obtiene el máximo entre cada regla de inferencia para cada una de las funciones de membresía correspondientes a cada una de las salidas, también se obtiene el mínimo entre las funciones de membresía de cada salida y después se obtiene el máximo de los mínimos antes encontrados.

## Análisis y Diseño

Para finalizar, se obtiene el valor de defusificación utilizando el método de centroide, esto con ayuda de la función “defuzz” que incluye la librería antes mencionada.

La función “pose\_callback” obtiene la pose del robot, es decir la posición en los ejes “X” y “Y”, y la orientación en cuaterniones, después transforma los cuaterniones a ángulos de Euler. Se define el punto al que se quiere llegar y se obtiene el error del ángulo entre el robot y el punto antes definido, de igual forma, se obtiene la distancia euclídea entre el robot y el punto. Posteriormente, se manda a llamar la función controlador y se obtienen los valores de velocidad angular y velocidad lineal. Por último, se publican estos valores al objeto move.

Al final del código, se crea un objeto tipo Twist y se inicia el nodo de odometría. Se publica el objeto antes creado y se subscribe al nodo de odometría para obtener la pose del robot. Por último, se crea un objeto de la clase Robot y se utiliza la función spin para que se repita el proceso, dicha función es parte de la librería rosipy.

Para facilitar la comprensión del algoritmo, a continuación, se muestra el diagrama de flujo de este.

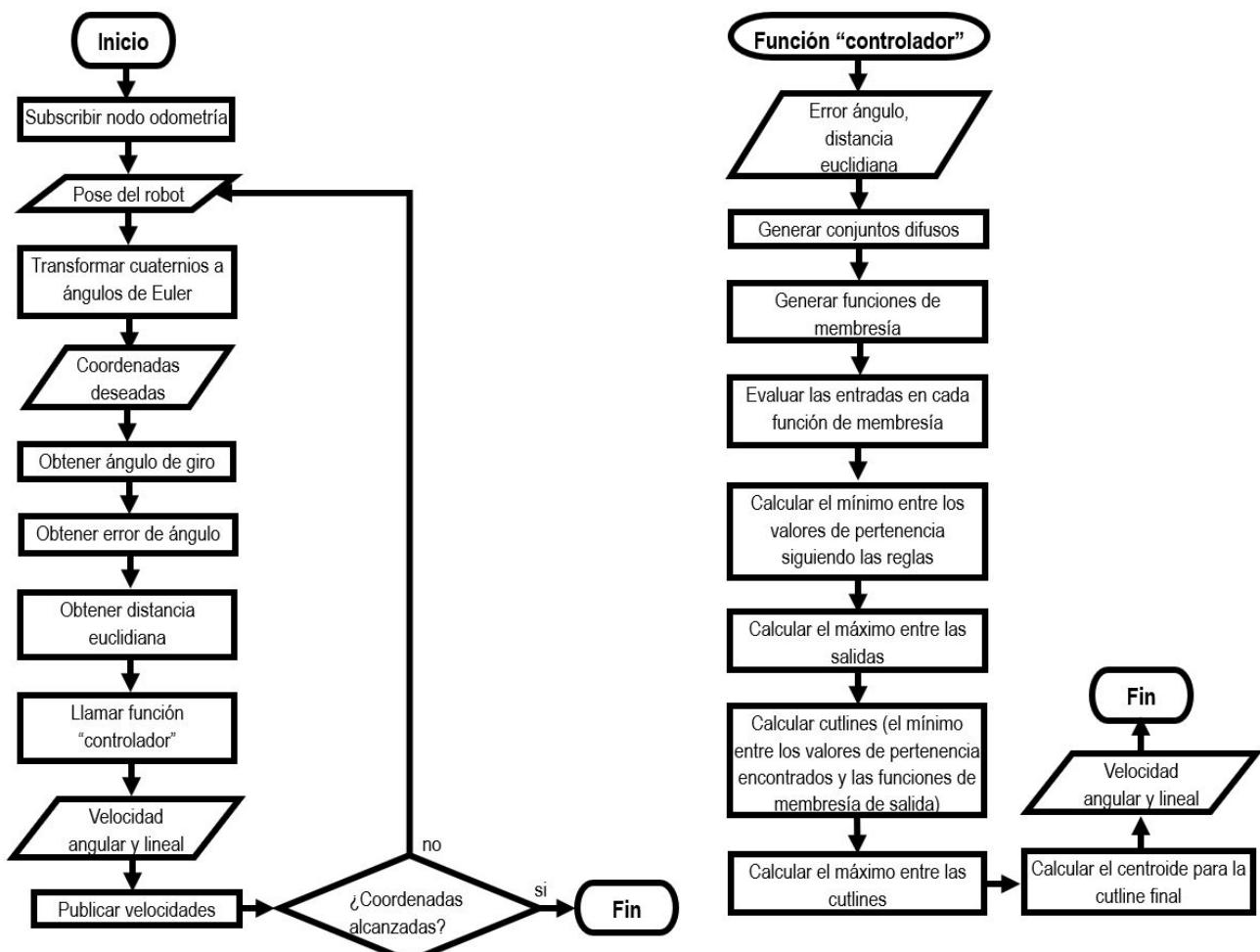


Figura 3.49: Diagrama de flujo del sistema de seguimiento de trayectoria (Control Difuso).

## Evasión de Obstáculos

Este sistema tiene como objetivo que el robot avance y evada obstáculos, además de indicar al sistema de formación cuando se debe modificar la formación. Para esto, se diseñó un controlador difuso de lazo abierto.

Se utilizó un modelo de inferencia difusa tipo Mamdani para el diseño del controlador, el cual tiene como entrada la distancia medida por el sensor LIDAR a distintos ángulos, en el norte, este y oeste, como salida se tiene la velocidad angular, velocidad lineal y una señal de formación.

El primer conjunto difuso de entrada es llamado “Norte”, cuyo universo de discurso va de 0 a 2.5, está compuesto por dos funciones de membresía de forma trapezoidal y una triangular. Sus variables lingüísticas son: cerca, medio y lejos.

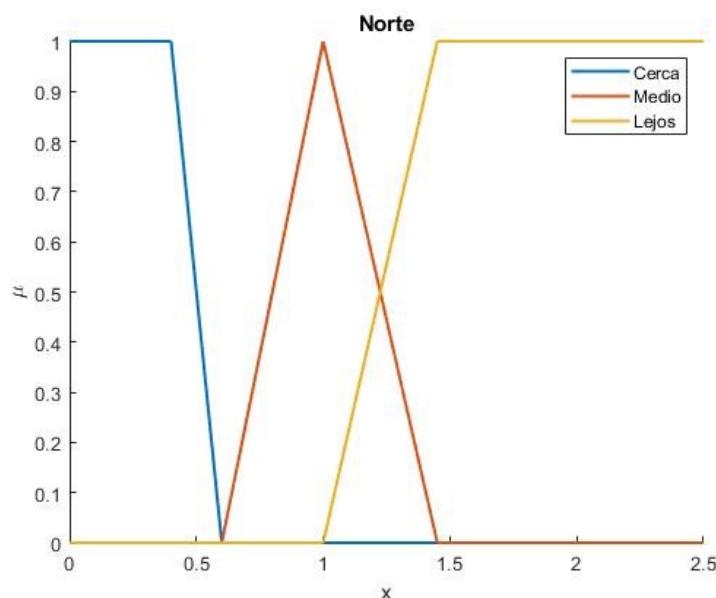


Figura 3.50: Conjunto difuso “Norte”.

Los siguientes conjuntos difusos de entrada se llaman “Este” y “Oeste”, y son iguales al conjunto “Norte”.

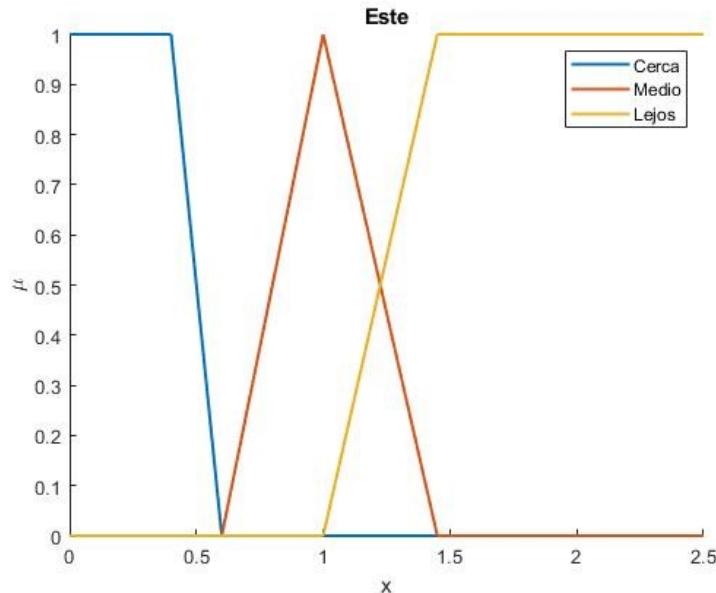


Figura 3.51: Conjunto difuso “Este”.

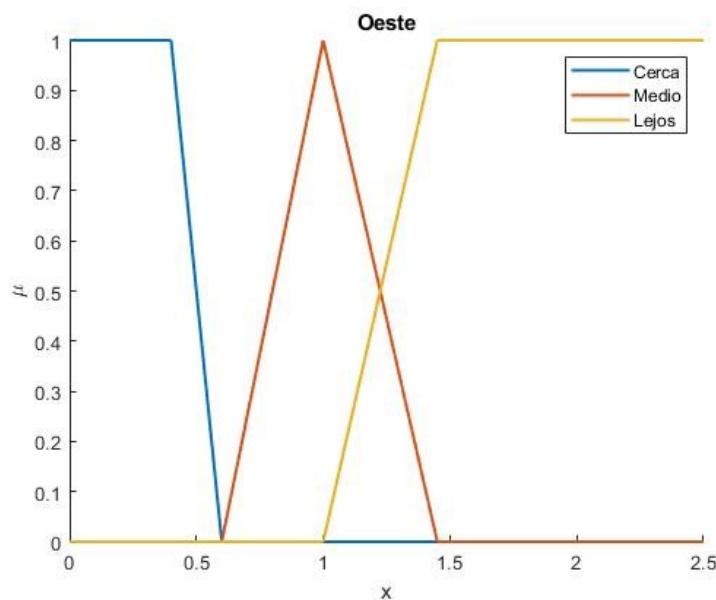


Figura 3.52: Conjunto difuso “Oeste”.

El primer conjunto difuso de salida se nombra “Velocidad angular”, su universo de discurso va desde  $-2.84$  hasta  $2.84$ , debido a que la velocidad angular máxima del robot es de  $2.84$  rad/s. Está compuesto por tres funciones de membresía de forma triangular y sus variables lingüísticas son: derecha, cero e izquierda.

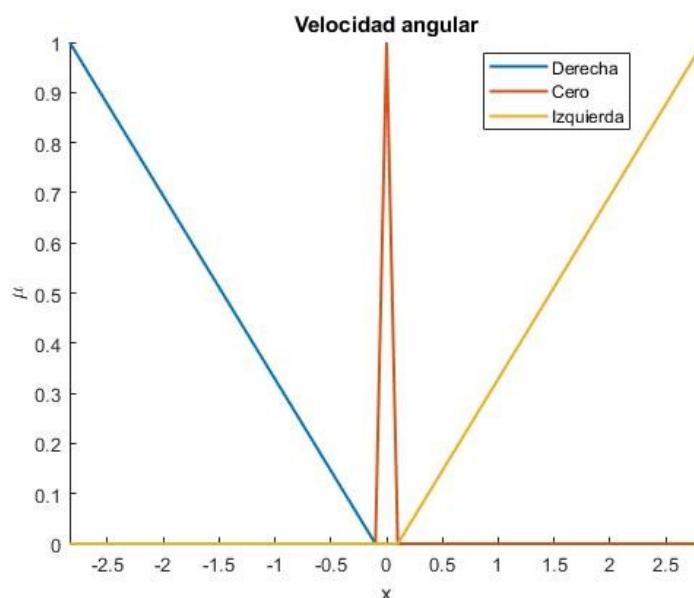


Figura 3.53: Conjunto difuso “Velocidad angular”.

El segundo conjunto difuso de salida se llama “Velocidad lineal”, lo conforman tres funciones de membresía de forma triangular y sus variables lingüísticas son: lento, medio y rápido. Su universo de discurso va de 0 a 0.22, esto se definió de acuerdo con la velocidad lineal máxima del robot que es de 0.22 m/s.

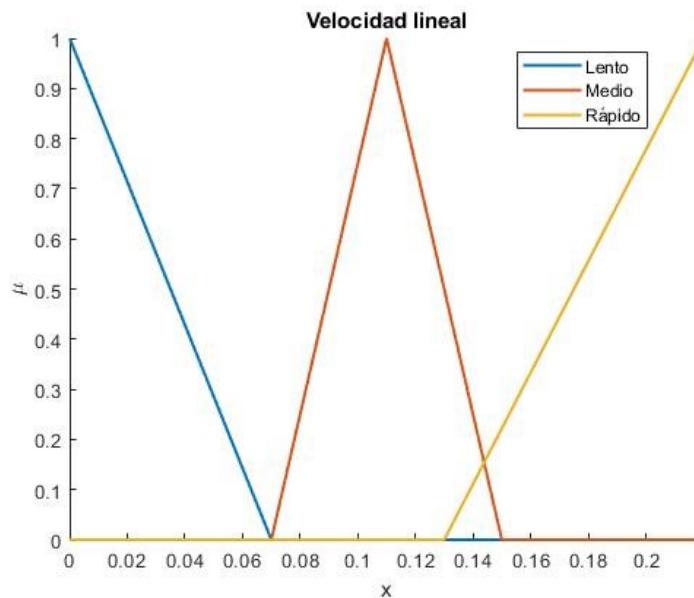


Figura 3.54: Conjunto difuso “Velocidad lineal”.

El último conjunto difuso de salida se llama “Formación” y se compone de dos funciones de membresía de forma triangular. Su universo de discurso va desde 0 hasta 5, sus variables lingüísticas son: constante y cambio.

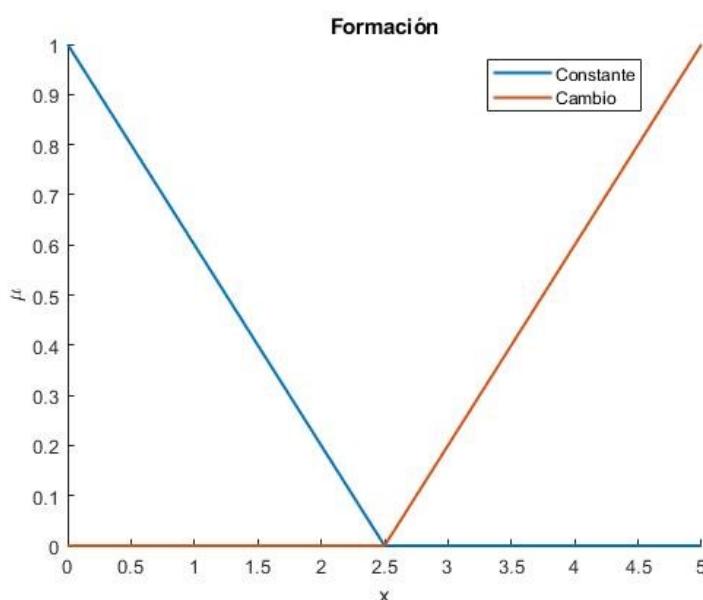


Figura 3.55: Conjunto difuso “Formación”.

Las reglas de inferencia difusa que se proponen para las salidas “Velocidad angular”, “Velocidad lineal” y “Formación” se muestran a continuación. El método de defusificación utilizado es el del centroide.

**SI N:Cerca Y E:Cerca Y O:Cerca, ENTONCES Vel.angular:I, Vel.lineal:Lento, Form:Cambio**

**SI N:Cerca Y E:Cerca Y O:Medio, ENTONCES Vel.angular:I, Vel.lineal:Lento, Form:Constante**

**SI N:Cerca Y E:Cerca Y O:Lejos, ENTONCES Vel.angular:I, Vel.lineal:Lento, Form:Constante**

**SI N:Cerca Y E:Medio Y O:Cerca, ENTONCES Vel.angular:D, Vel.lineal:Lento, Form:Constante**

**SI N:Cerca Y E:Medio Y O:Medio, ENTONCES Vel.angular:I, Vel.lineal:Lento, Form:Constante**

**SI N:Cerca Y E:Medio Y O:Lejos, ENTONCES Vel.angular:I, Vel.lineal:Lento, Form:Constante**

**SI N:Cerca Y E:Lejos Y O:Cerca, ENTONCES Vel.angular:D, Vel.lineal:Lento, Form:Constante**

**SI N:Cerca Y E:Lejos Y O:Medio, ENTONCES Vel.angular:D, Vel.lineal:Lento, Form:Constante**

**SI N:Cerca Y E:Lejos Y O:Lejos, ENTONCES Vel.angular:I, Vel.lineal:Lento, Form:Constante**

**SI N:Medio Y E:Cerca Y O:Cerca, ENTONCES Vel.angular:C, Vel.lineal:Medio, Form:Cambio**

**SI N:Medio Y E:Cerca Y O:Medio, ENTONCES Vel.angular:C, Vel.lineal:Medio, Form:Constante**

**SI N:Medio Y E:Cerca Y O:Lejos, ENTONCES Vel.angular:C, Vel.lineal:Medio, Form:Constante**

**SI N:Medio Y E:Medio Y O:Cerca, ENTONCES Vel.angular:C, Vel.lineal:Medio, Form:Constante**

**SI N:Medio Y E:Medio Y O:Medio, ENTONCES Vel.angular:C, Vel.lineal:Medio, Form:Constante**

**SI N:Medio Y E:Medio Y O:Lejos, ENTONCES Vel.angular:C, Vel.lineal:Medio, Form:Constante**

**SI N:Medio Y E:Lejos Y O:Cerca, ENTONCES Vel.angular:C, Vel.lineal:Medio, Form:Constante**

**SI N:Medio Y E:Lejos Y O:Medio, ENTONCES Vel.angular:C, Vel.lineal:Medio, Form:Constante**

**SI N:Medio Y E:Lejos Y O:Lejos, ENTONCES Vel.angular:C, Vel.lineal:Medio, Form:Constante**

**SI N:Lejos Y E:Cerca Y O:Cerca, ENTONCES Vel.angular:C, Vel.lineal:Rápido, Form:Cambio**

.....

**SI N:Lejos Y E:Cerca Y O:Medio, ENTONCES Vel.angular:C, Vel.lineal:Rápido, Form:Constante**

**SI N:Lejos Y E:Cerca Y O:Lejos, ENTONCES Vel.angular:C, Vel.lineal:Rápido, Form:Constante**

**SI N:Lejos Y E:Medio Y O:Cerca, ENTONCES Vel.angular:C, Vel.lineal:Rápido, Form:Constante**

**SI N:Lejos Y E:Medio Y O:Medio, ENTONCES Vel.angular:C, Vel.lineal:Rápido, Form:Constante**

**SI N:Lejos Y E:Medio Y O:Lejos, ENTONCES Vel.angular:C, Vel.lineal:Rápido, Form:Constante**

**SI N:Lejos Y E:Lejos Y O:Cerca, ENTONCES Vel.angular:C, Vel.lineal:Rápido, Form:Constante**

**SI N:Lejos Y E:Lejos Y O:Medio, ENTONCES Vel.angular:C, Vel.lineal:Rápido, Form:Constante**

**SI N:Lejos Y E:Lejos Y O:Lejos, ENTONCES Vel.angular:C, Vel.lineal:Rápido, Form:Constante**

## Validación

Para la validación del sistema se generó el controlador difuso con ayuda del Toolbox Fuzzy Logic Designer como se muestra en la figura 3.56. Se utilizó un controlador tipo Mamdani, con tres entradas y tres salidas.

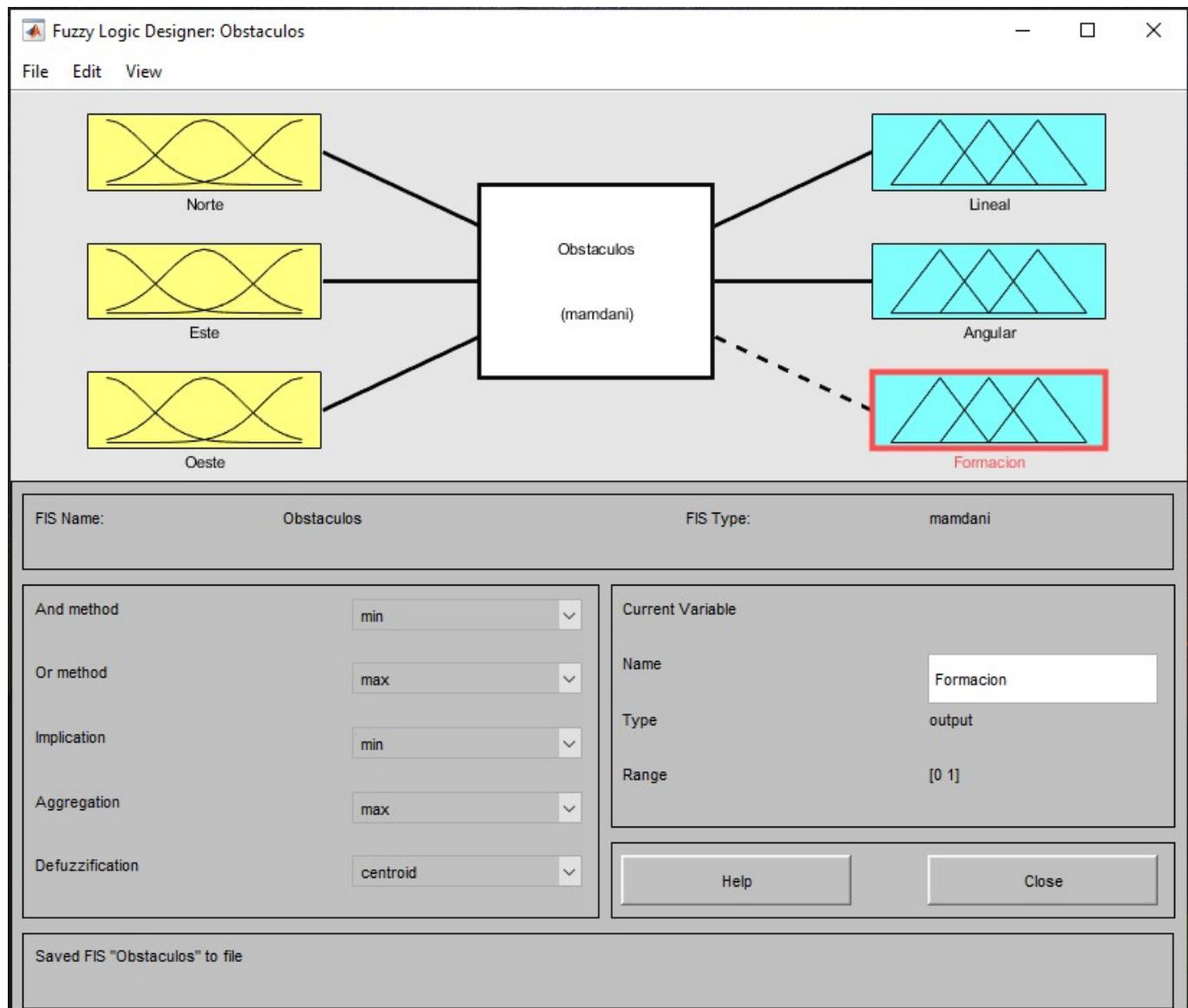


Figura 3.56: Construcción del controlador difuso en el toolbox Fuzzy Logic Designer de MATLAB.

Posteriormente, se obtuvieron las superficies de control de las salidas velocidad angular, velocidad lineal y formación.

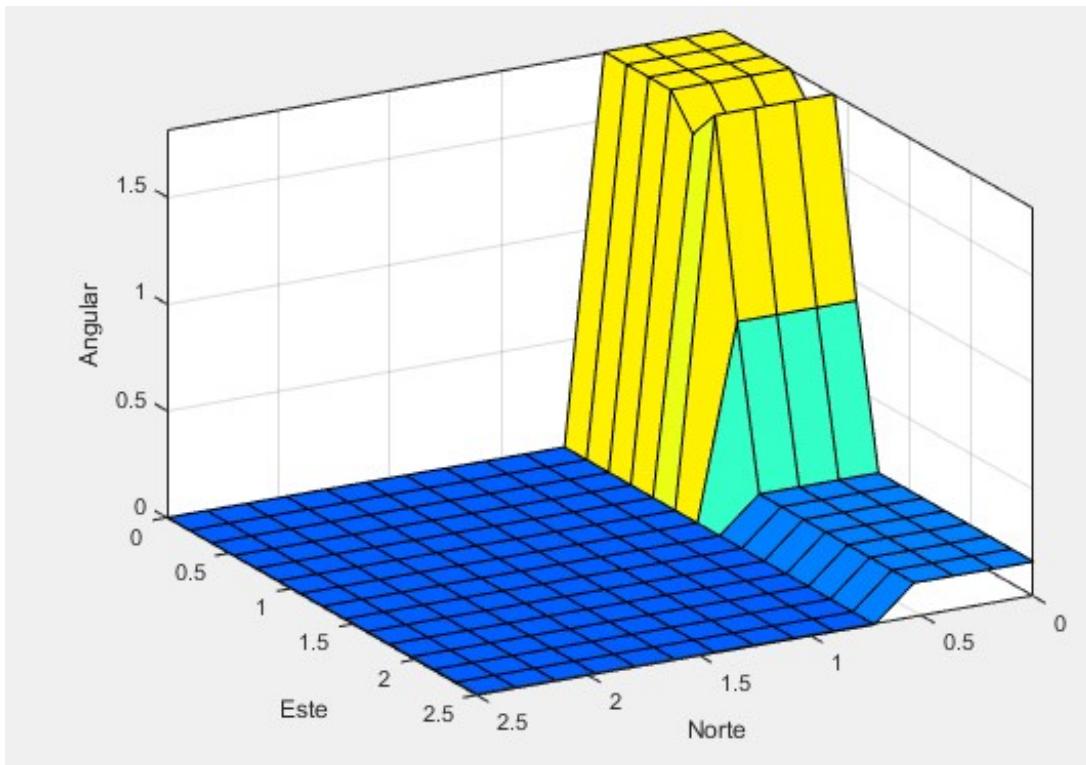


Figura 3.57: Superficie de control de la salida velocidad angular.

En la gráfica de la figura 3.57, se puede ver que la velocidad angular solo tiene cambios cuando la distancia de la entrada norte se encuentre entre cero y uno.

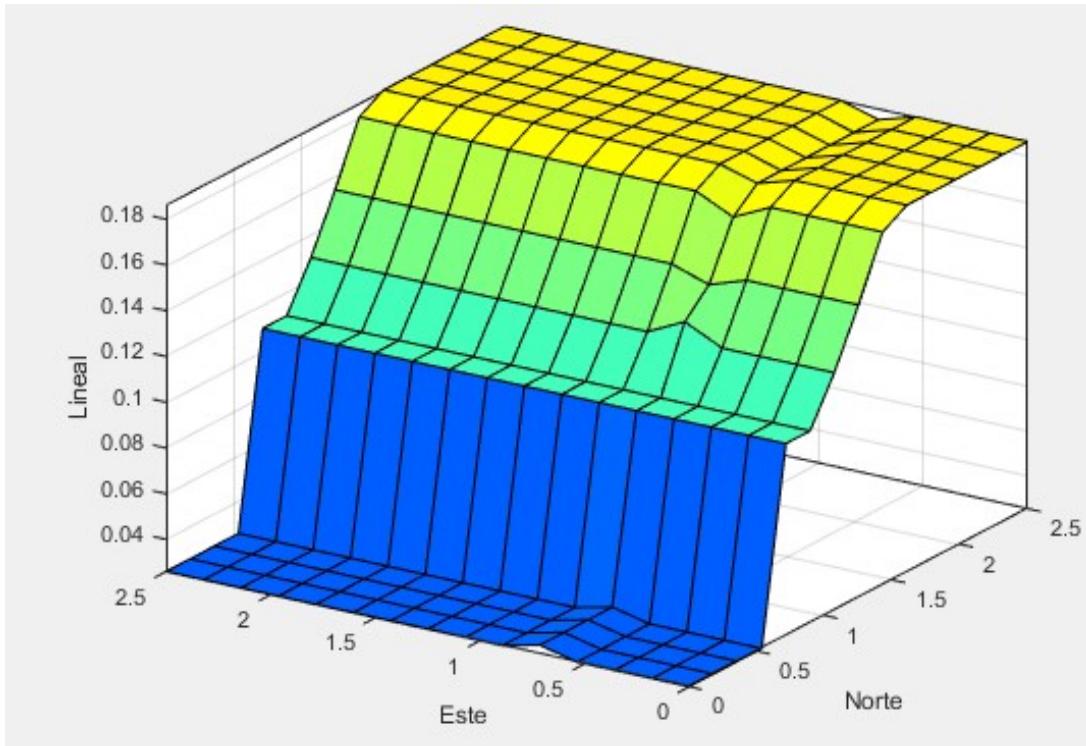


Figura 3.58: Superficie de control de la salida velocidad lineal.

En la gráfica de la figura 3.58, se observa que la velocidad lineal tiene cambios repentinos cuando la distancia de la entrada norte cambia.

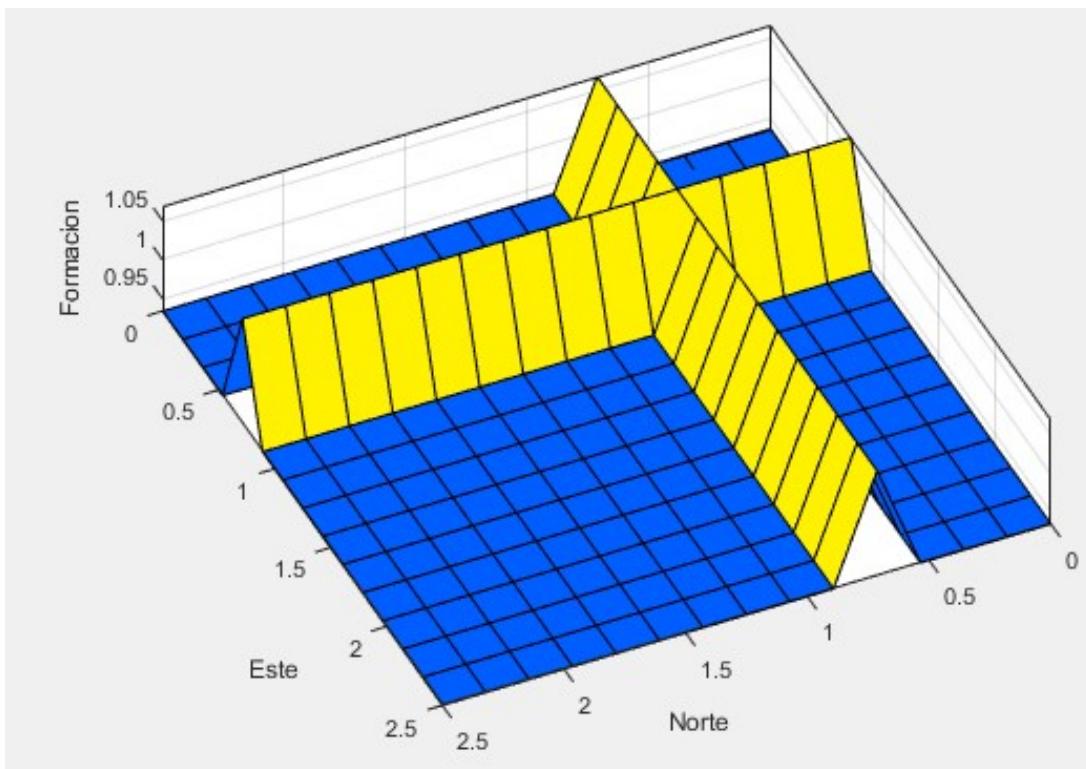


Figura 3.59: Superficie de control de la salida formación.

En la gráfica de la figura 3.59, se puede percibir un cambio violento en la salida formación, esto se debe a que solo se tienen dos funciones de membresía en la salida.

Una vez que se comprobó que el controlador funciona correctamente, se implementó del controlador en Python. En el código, primero se declara una función llamada callback que obtiene las mediciones del sensor LIDAR en los ángulos  $0^\circ$ ,  $7^\circ$  y  $353^\circ$  y asigna la menor distancia a la variable Norte, además asigna la distancia de los ángulos  $90^\circ$  y  $270^\circ$  a Este y Oeste. A continuación, se declaran los conjuntos difusos y cada una de las funciones de membresía de estos. Después, se obtiene el mínimo entre cada función de membresía de ambas entradas de acuerdo con el valor recibido del error de ángulo y error de distancia euclidiana para cada una de las quince reglas de inferencia. Posteriormente, se obtiene el máximo entre cada regla de inferencia para cada una de las funciones de membresía correspondientes a cada una de las salidas, también se obtiene el mínimo entre las funciones de membresía de cada salida y luego se obtiene el máximo de los mínimos antes encontrados. Para finalizar, se obtiene el valor de defusificación utilizando el método de centroide para la velocidad angular, velocidad lineal y formación. Por último, se publican estos valores al objeto move.

Al final del código, se crea un objeto tipo Twist y se inicia el nodo de LaserScan. Se publica el objeto antes creado y se subscribe al nodo LaserScan para obtener las distancias medidas por el sensor LIDAR del robot. Finalmente, se utiliza la función spin para que se repita el proceso.

Para facilitar la comprensión del algoritmo, en la siguiente figura, se muestra el diagrama de flujo de este.

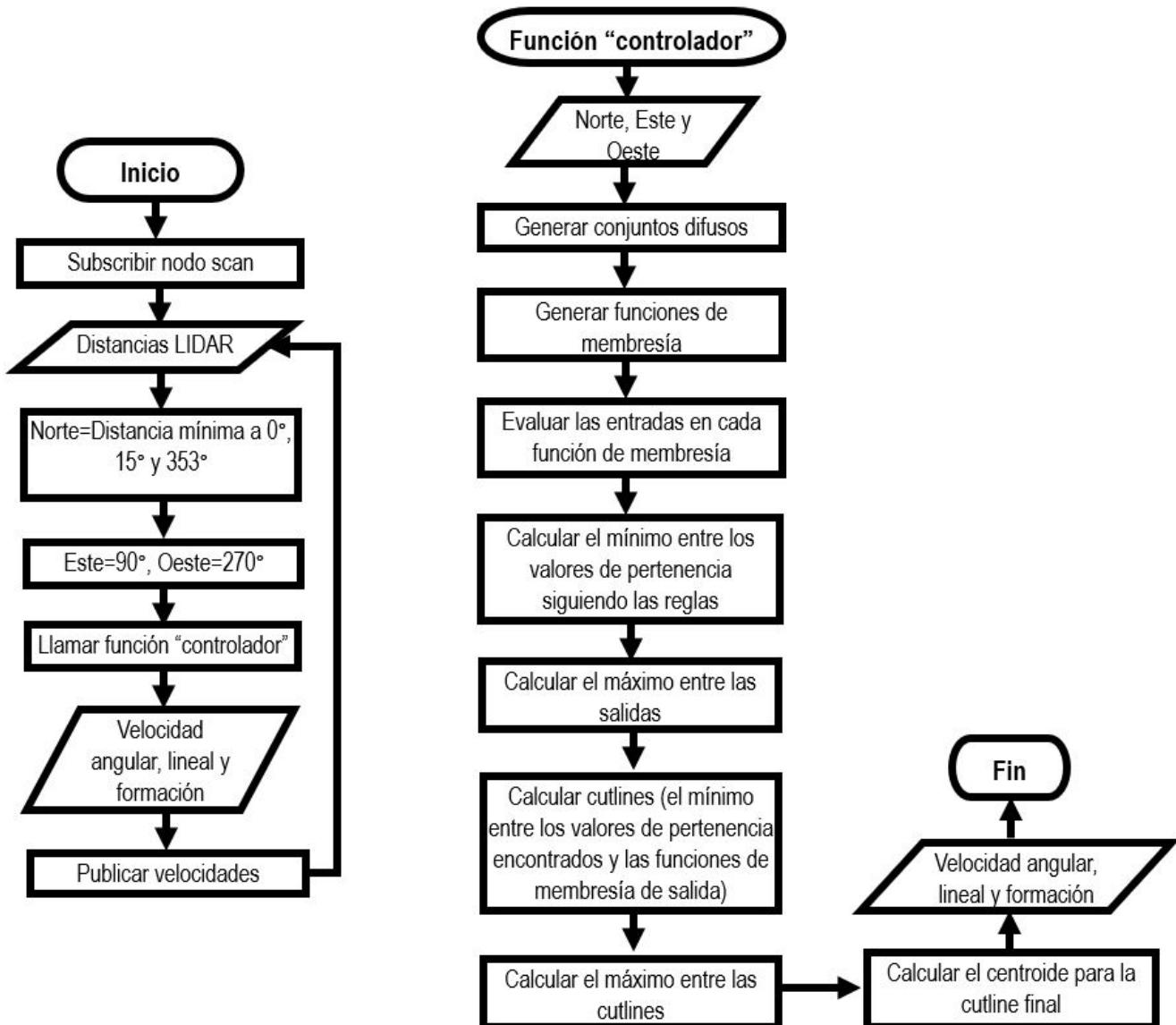


Figura 3.60: Diagrama de flujo del sistema de evasión.

## Sistema de Formación

Para coordinar la formación de los robots, se tienen tres objetivos:

1. Realizar la formación inicial
2. Mantener la formación indicada
3. Modificar la formación dependiendo de las condiciones del entorno

El primer objetivo es el más sencillo de realizar, este se enfocará en mandar a los robots a una posición  $(x,y)$  con respecto a la posición del líder, esto es, de la posición actual que tenga el robot asignado como líder, asignar un offset a esas coordenadas y mandarlas como puntos destino para que los robots seguidores puedan desplazarse hacia ellos. Es importante tener en cuenta que, para la obtención de este punto, también se tiene que considerar la rotación que tiene el robot líder.

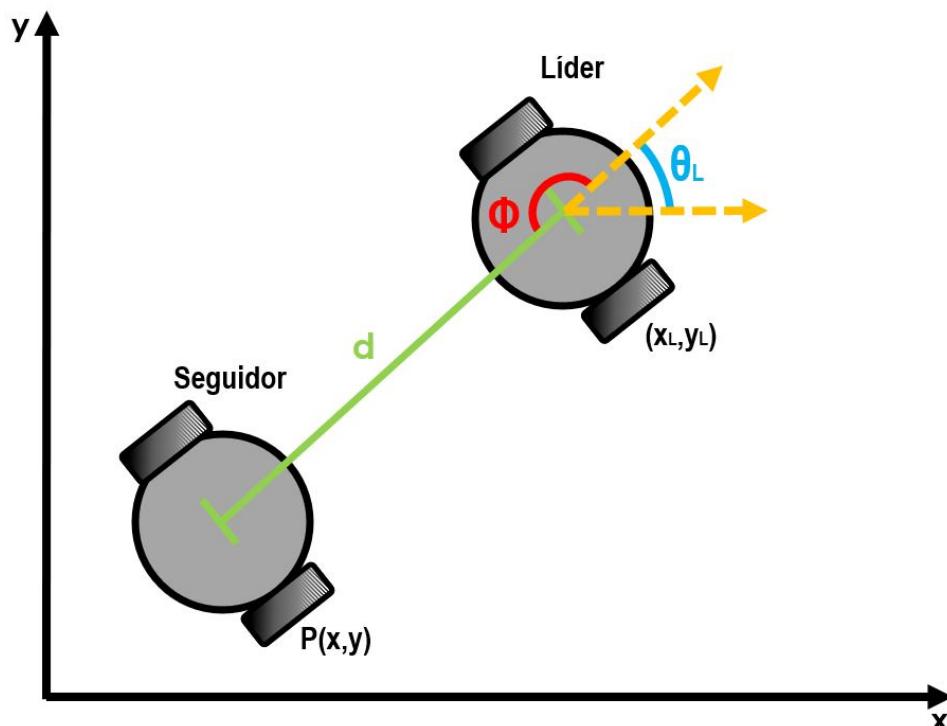


Figura 3.61: Representación del cálculo del punto destino para un robot seguidor.

Considerando que, la distancia  $d$  es la deseada entre el robot líder y el robot seguidor y que el ángulo  $\phi$  es el requerido por la formación, el punto destino para cada robot seguidor se obtiene de la siguiente forma:

$$P = [d \cos(\phi + \theta_L), d \sin(\phi + \theta_L)] \quad (3.6)$$

Una vez obtenidos estos puntos, se utiliza el controlador empleado en el seguimiento de trayectorias para guiar a los robots hasta ellos.

Para el segundo objetivo, mantener la formación durante el recorrido, se pueden encontrar una variedad de técnicas que proporcionan un control con un buen funcionamiento para los robots. En la primera de ellas, el robot líder se encarga de ir asignando puntos destino, de la misma forma de cómo se realizó el primer objetivo, cada periodo  $T$  de tiempo. Las ventajas de esta técnica, es que únicamente se enfocaría en calcular el punto destino continuamente, para luego mandárselo a los seguidores y, aplicando el controlador para el seguimiento, guiar al robot hasta este punto. La desventaja principal radica en su funcionamiento, ya que se requiere trabajar con cambios en el marco de referencia para que no surjan problemas cuando el robot tenga que aproximarse a ángulos cercanos a  $\pi$  o  $-\pi$ .

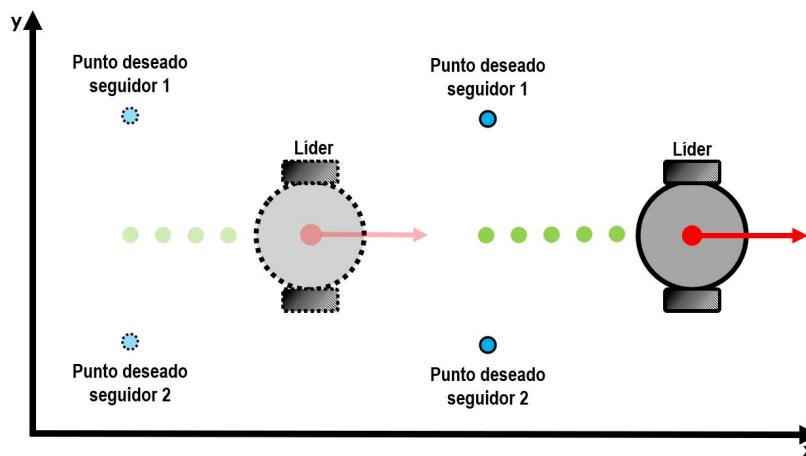


Figura 3.62: Representación del cálculo constante de los puntos destino durante el recorrido.

Otra técnica que se puede aplicar, sigue un simple objetivo, mantener al robot seguidor a una distancia  $d$  del robot líder con un ángulo  $\phi$  generado entre el robot líder y el robot seguidor [7].

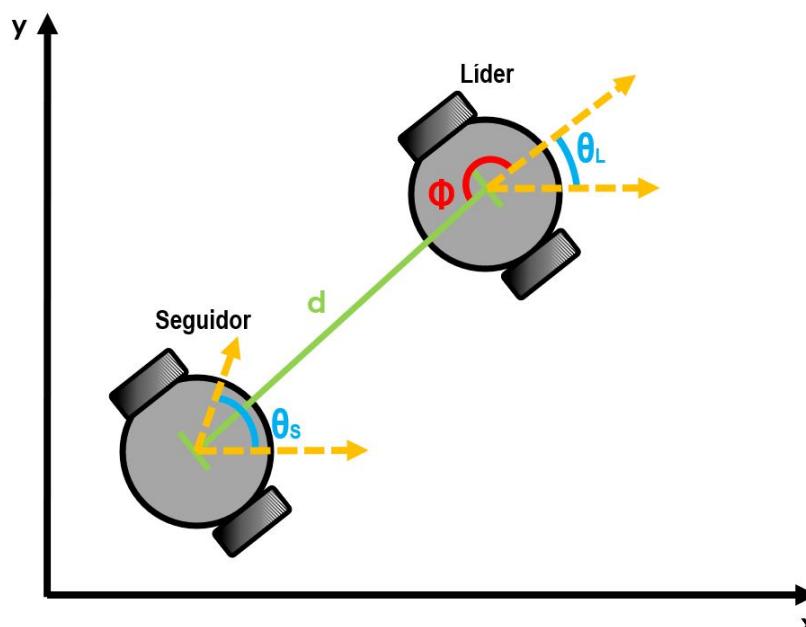


Figura 3.63: Representación del control de formación mediante un ángulo y una distancia.

## Análisis y Diseño

---

La principal ventaja que se tiene, radica en un mejor desempeño con respecto a la primer técnica, pues en este caso, el controlador solo se encarga de hacer las modificaciones pertinentes a la velocidad linear y angular para cumplir con la condición de mantener la distancia y el ángulo entre el líder y seguidor. La desventaja que se presenta, es que, posiblemente, sea necesario recurrir a la implementación de un controlador difuso para el sistema, dependiendo del número de robots que se este manejando.

Para el tercer objetivo, se recibe una señal del sistema de evasión, la cual indica que el robot líder ha detectado un obstáculo y es necesario realizar el cambio de formación. Al recibir dicha señal, se le indica al sistema para que los puntos destino de los seguidores cambien y así, en vez de que estos se formen en delta, lo hagan en convoy. Con esta opción, al sistema le tomaría menos tiempo reformarse ya que se realizaría mientras el líder sigue en movimiento. Otra propuesta es detener el sistema por completo, definir un tiempo para que se reformen los robots y después retomar el desplazamiento.

Considerando que los robots tienen un radio de 21 cm, se propone una distancia deseada para la formación de 25cm. Sin embargo, esta podría ser modificada dentro de un rango que va de los 15cm a los 25cm, definido así debido a las dimensiones de los robots y del escenario de pruebas.

## Validación

Para validar el método planteado para realizar las formaciones, se utiliza como herramienta auxiliar el software MATLAB. Primero, se presenta la formación delta, utilizando como posición inicial del líder el punto (1,1) con una orientación de  $0^\circ$ , se calculan los puntos destino de cada seguidor:

### Seguidor 1   Seguidor 2

$$x=0.5670 \quad x=0.5670$$

$$y=1.2500 \quad y=0.7500$$

Luego, se presenta la gráfica del movimiento generado para los robots seguidores. En ella, se puede observar que los robots seguidores llegan a la posición calculada para realizar la formación inicial.

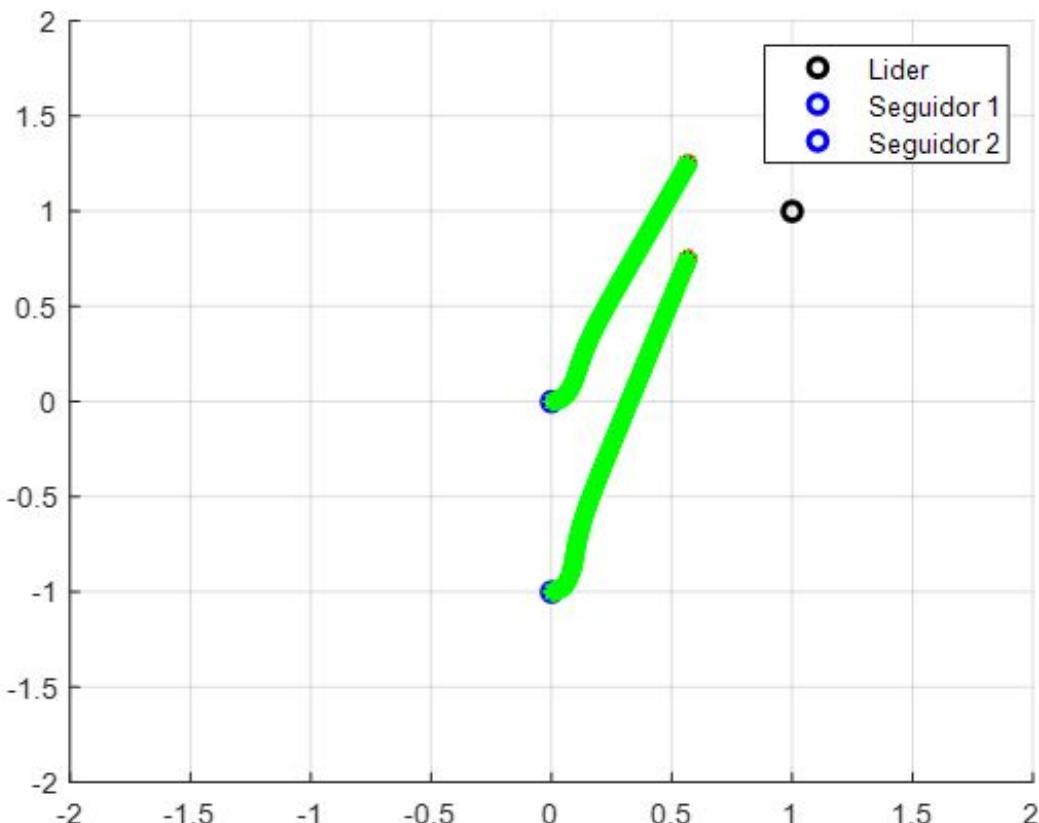


Figura 3.64: Navegación de los robots a los puntos de la formación delta.

En las gráficas de los errores, se puede ver que estos tienden a cero y llegan a su posición en menos de 5s, por lo que se puede concluir que se cumple con el tiempo especificado dentro de los requerimientos para realizar la formación inicial de los robots.

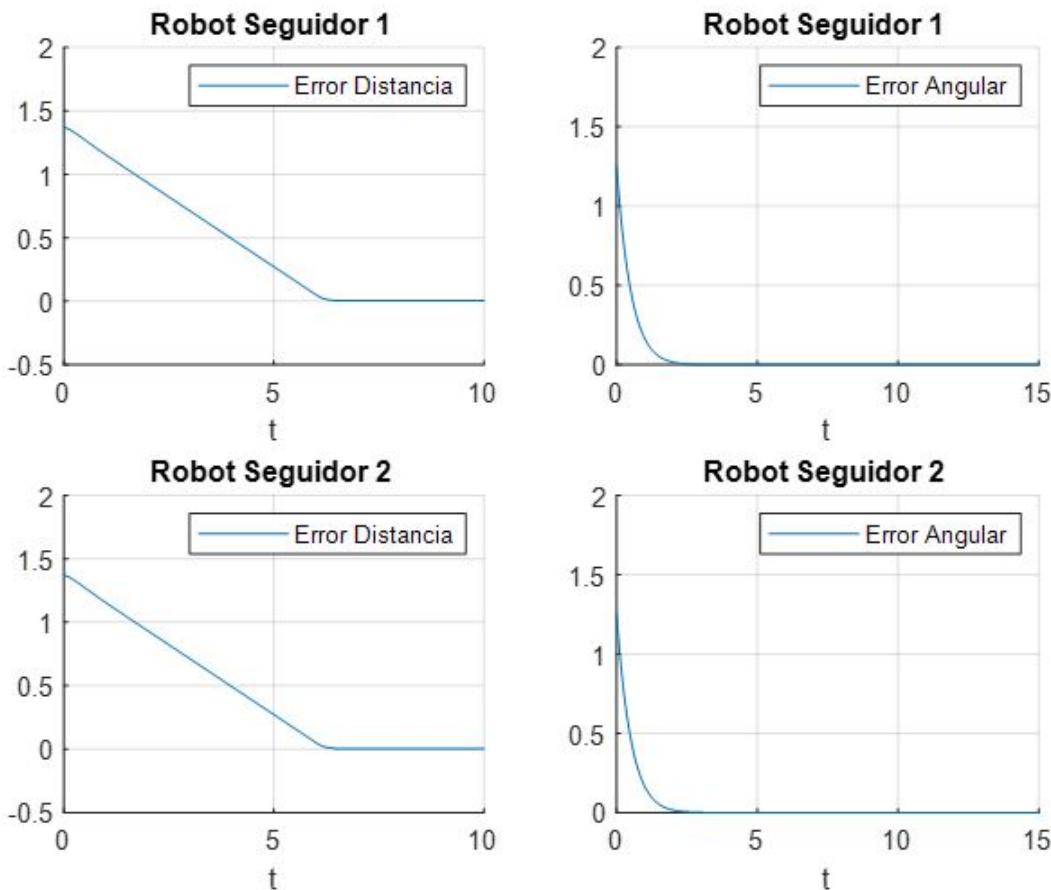


Figura 3.65: Gráficas del error angular y de distancia de los seguidores en la formación delta.

La segunda prueba se realizar para la formación en convoy, dando como posición inicial del líder el punto  $(0,1)$  con una orientación de  $0^\circ$ . Así, se calculan las posiciones deseadas para los seguidores:

| Seguidor 1 | Seguidor 2 |
|------------|------------|
| $x=-0.5$   | $x=-1$     |
| $y=1$      | $y=1$      |

En el desplazamiento de los robots, se puede observar que estos llegan a los puntos calculados, realizando la formación de manera adecuada.

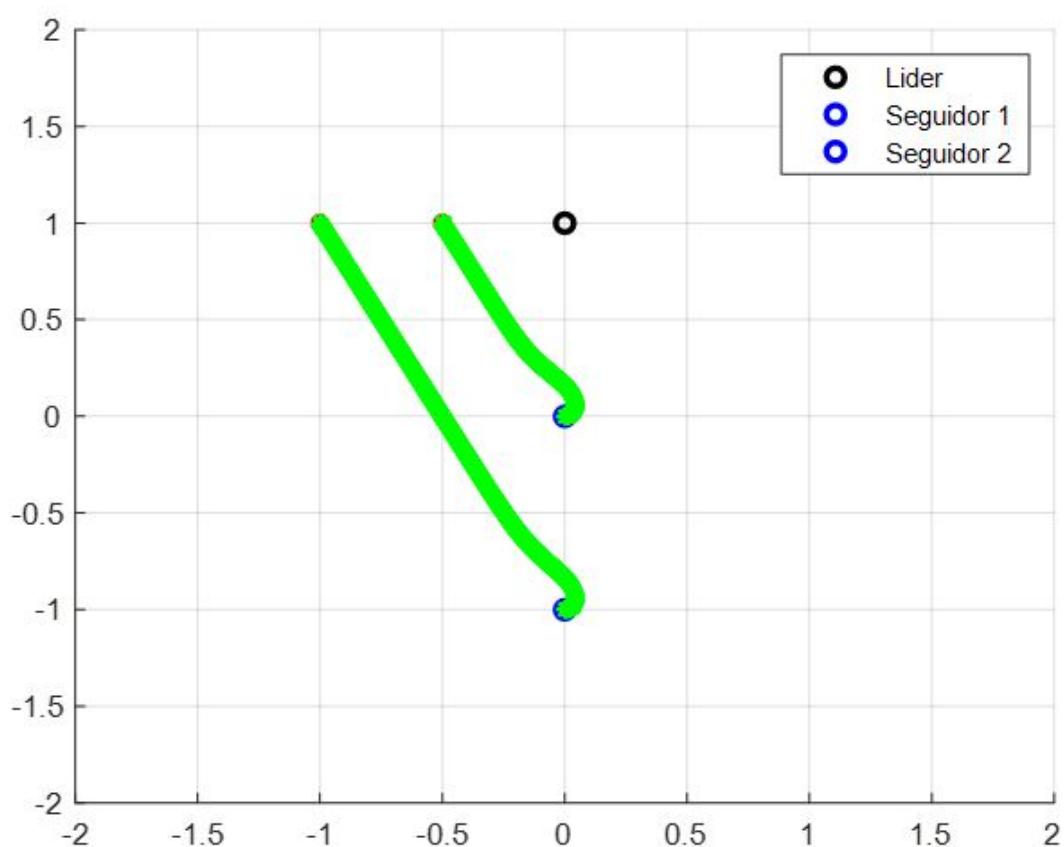


Figura 3.66: Navegación de los robots a los puntos de la formación convoy.

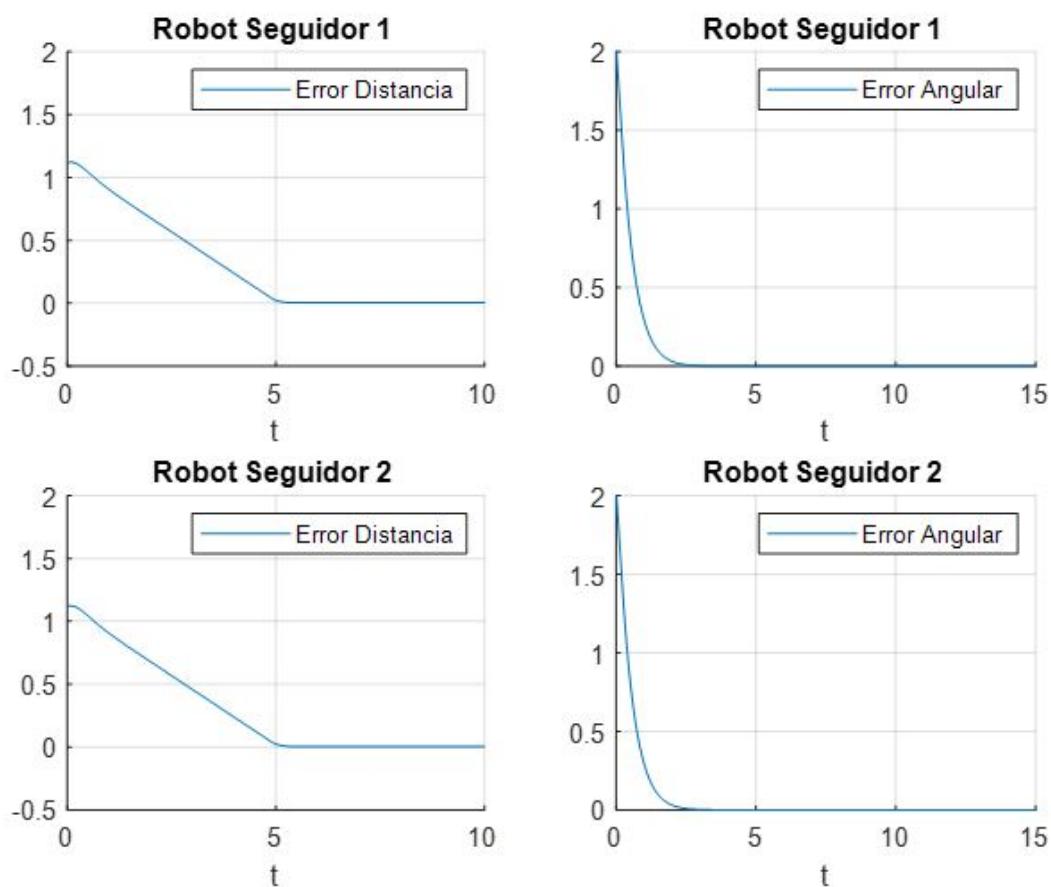


Figura 3.67: Gráficas del error angular y de distancia de los seguidores en la formación convoy.

Las gráficas de los errores para cada seguidor muestran que el error llega a cero poco más de 5s, lo que significa que la formación se realiza en menos de 5s, que al igual que en la primera prueba, demuestra que el tiempo mínimo especificado en los requerimientos se cumple. Por los resultados obtenidos en ambas pruebas, puede concluir que el método elegido funciona adecuadamente.

Para mantener la formación durante el trayecto, el robot líder debe asignar los puntos que deben seguir los robots seguidores para mantener la formación, lo cual implica la utilización de un controlador que realice el seguimiento de dichos puntos. Partiendo del controlador antes realizado para el seguimiento de trayectorias, se considera como el error de la distancia, la deseada entre el robot líder y cada seguidor menos la distancia actual que haya entre ellos y como el error del ángulo, la diferencia entre el ángulo  $\psi$ , formado desde la posición del robot seguidor  $(x_s, y_s)$  hasta el punto destino  $(x_d, y_d)$ , y la orientación del robot seguidor  $\theta_s$ .

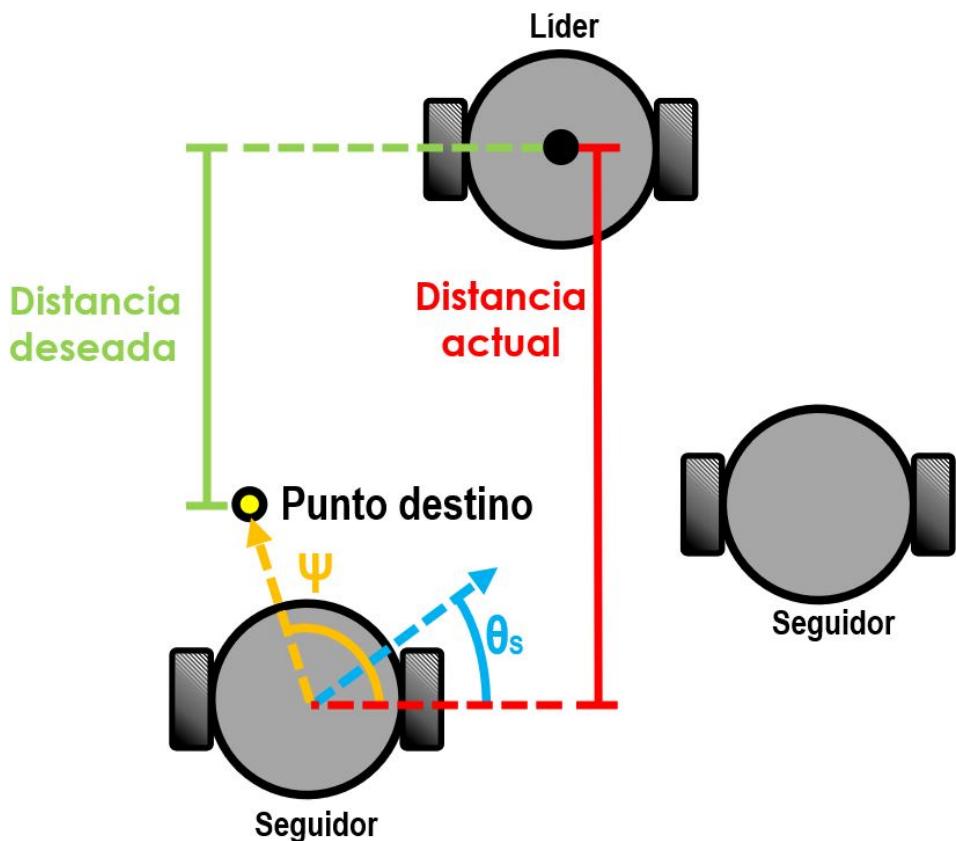


Figura 3.68: Representación del seguimiento al punto destino.

Así, para simular el comportamiento de este sistema, se utiliza el código realizado para el seguimiento de trayectorias con un controlador P, realizando solo las modificaciones mencionadas anteriormente.

El diagrama de flujo, que representa el funcionamiento del algoritmo para del código para este sistema, se muestra a continuación.

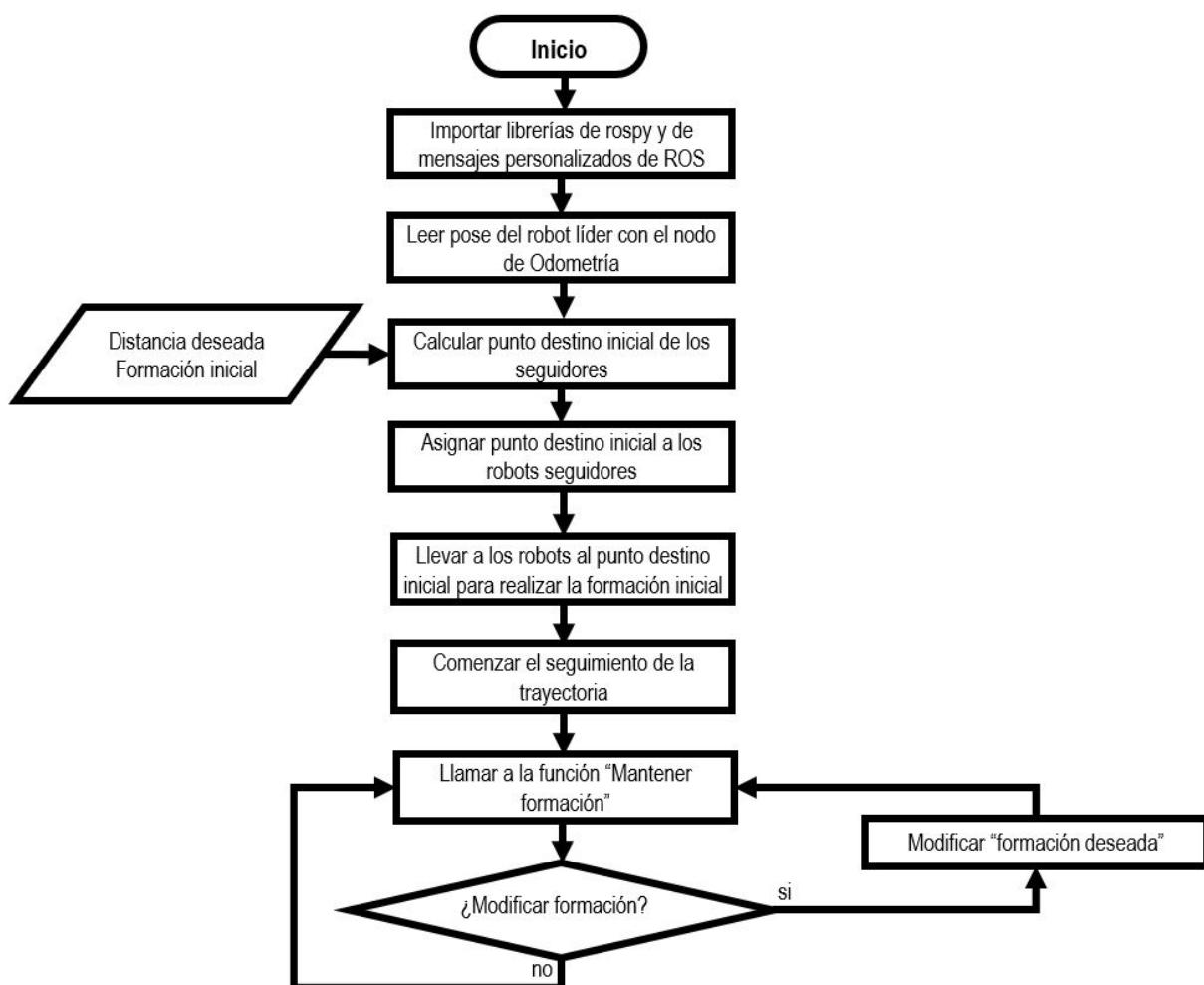


Figura 3.69: Diagrama de flujo del sistema de formación.

En donde la función “Mantener formación” esta descrita por el diagrama de flujo de la figura 3.70. Y consiste en actualizar los puntos a los que se deben dirigir los seguidores cuando el robot líder está en movimiento.

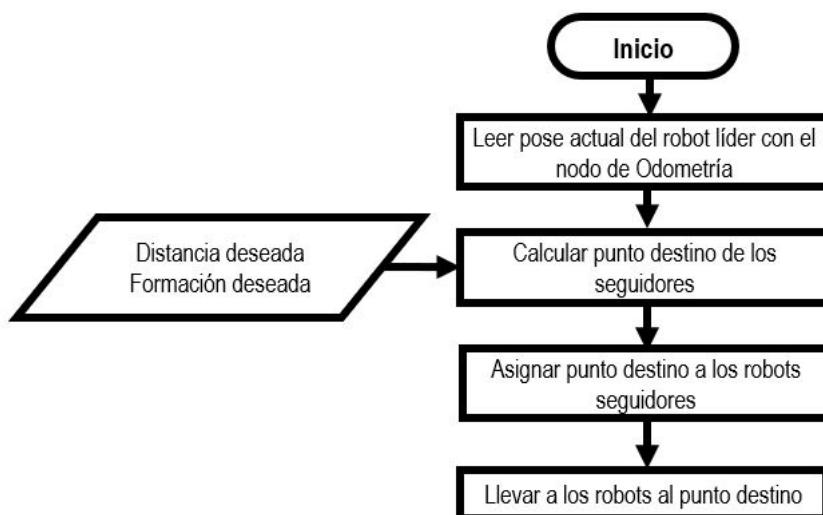


Figura 3.70: Diagrama de flujo de la función Mantener formación.

### 3.4.4. Área funcional 3: Movimiento

Esta área, esta conformada por los servomotores (Dynamixel XL430-W250-T), los cuales proporcionan al robot la capacidad de desplazarse en una superficie. Además, ayuda al sistema de percepción brindando información de odometría a través de su encoder para conocer la posición de los robots.

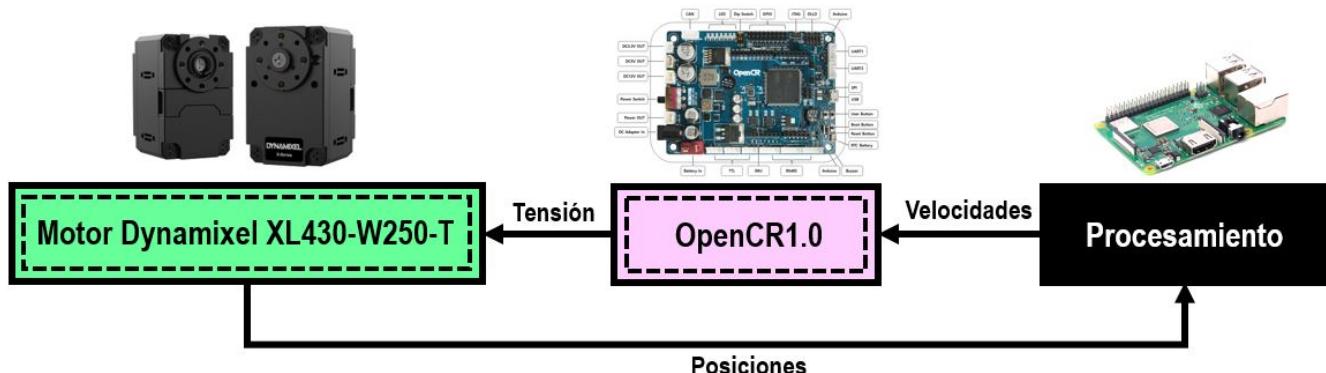


Figura 3.71: Diagrama área funcional Movimiento.

### Descripción de los componentes

De acuerdo con los requerimientos y la descripción de los robots, los componentes a emplear se describen a continuación.

#### Dynamixel XL430-W250-T

El XL-430-W250-T Dynamixel es un servo clasificado dentro de la familia de bajo costo de los servomotores. Gracias a su diseño pequeño y compacto, provee una gran flexibilidad para ensambles y para su uso [32].



Figura 3.72: Dynamixel XL430-W250-T [32].

A continuación, se presentan las características de este motor.

Tabla 3.7: Especificaciones motor Dynamixel XL430-W250-T [32].

| Elemento                      | Especificación  |
|-------------------------------|---|
| MCU                           | ARM CORTEX-M3 (72 MHz, 32Bit)   |
| Sensor de Posición            | Encoder (12 Bits, 360°)   |
| Motor                         | Bobinado  |
| Velocidad de baudeo           | 9,600 bps ~ 4.5 Mbps  |
| Algoritmo de control          | Control PID   |
| Resolución                    | 4096 pulsos/rev   |
| Peso                          | 57.2 g  |
| Dimensiones                   | 28.5 x 46.5 x 34 mm   |
| Relación de engrane           | 258.5:1   |
| Temperatura de Operación      | -5 ~ + 72 °C  |
| Modos de operación            | Modo Control de Velocidad   |
|                               | Modo Control de Posición (0° ~ 360°)  |
|                               | Modo Control Extendido de Posición (Multi-vueltas)  |
|                               | Modo Control PWM (Modo Control de Voltaje)  |
| Par Torsor                    | 1 Nm (a 9 V, 1 A)   |
|                               | 1.4 Nm (a 11.1 V, 1.3 A)  |
|                               | 1.5 Nm (a 12 V, 1.4 A)  |
| Velocidad sin carga           | 47 rev/min (a 9 V)  |
|                               | 57 rev/min (a 11.1 V)   |
|                               | 61 rev/min (a 12 V)   |
| Voltaje de entrada            | 6.5 ~ 12 V (Recomendado: 11.1 V)  |
| Tipo de Protocolo             | TTL (8bit, 1stop, sin paridad)  |
| Retroalimentación             | Posición, Velocidad, Carga, Tick en Tiempo Real, Trayectoria, Temperatura, Voltaje de Entrada, etc. |
| Corriente en tiempo de espera | 52 mA   |

### 3.5. Integración de Áreas Funcionales

A nivel estructural, la integración de las áreas funcionales está representada de la siguiente manera.



Figura 3.73: Integración de las áreas funcionales a nivel estructural.

En esta representación, la estructura del robot se encarga de sostener los componentes de las áreas funcionales, es decir, las llantas, los motores Dynamixel XL430-W250-T, el sensor LiDAR LDS-01, la tarjeta OpenCR1.0 y la Raspberry Pi 3 Model B+.

El área funcional de procesamiento, internamente, será la encargada de mantener comunicados los robots con la computadora central a través de un medio inalámbrico con buen alcance, como los módulos WiFi.

A nivel funcional, los sistemas que componen el área funcional de procesamiento, se integrarán en dos etapas:

1. La navegación del robot líder
2. La navegación de los seguidores

Teniendo como base el IDEF0 del sistema, se realizará el intercambio de información entre los distintos sistemas a través de señales. Sin embargo, unificar todos los sistemas dentro de un mismo código es una tarea difícil, pues hay procesos que necesitan llevarse a la par. Esta dificultad se afronta durante la navegación del robot líder, y la navegación de los robots seguidores, ya que mientras el robot procesa todo lo que necesita para llegar al punto destino, se debe ir asignando los puntos destino de los robots seguidores y su navegación hacia ellos. De forma visual, este problema se puede comprender más fácil con el siguiente diagrama.

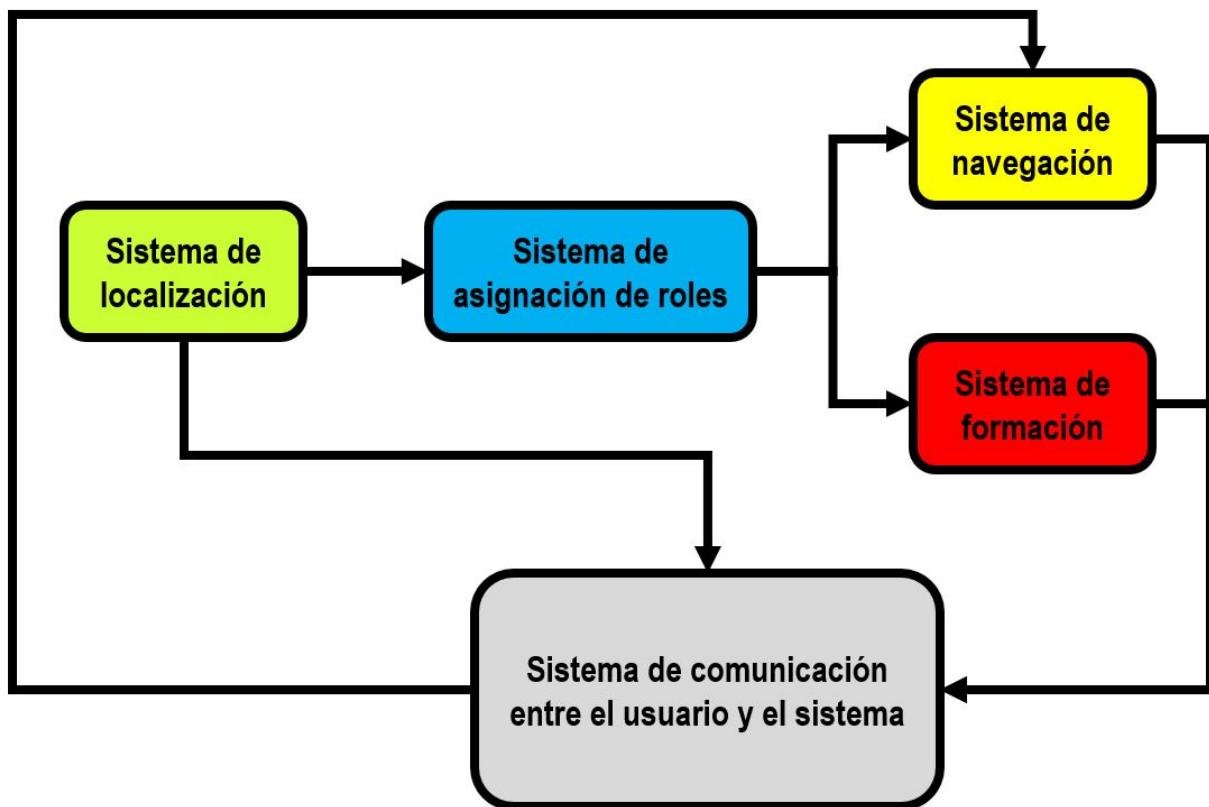


Figura 3.74: Integración de las sistemas a nivel funcional.

Como se puede ver, la navegación y la formación son procesos que deben llevarse a la par para que el sistema funcione. Afortunadamente, el lenguaje de programación Python, permite abordar este problema realizando subprocesos dentro del código, lo cual es básicamente realizar procesos distintos al mismo tiempo durante la ejecución del programa.

# 4

## Pruebas y Resultados

Para cada uno de los objetivos específicos del proyecto, deberán definirse los resultados esperados, de tal manera que pueda determinarse si han sido o no alcanzados (o en qué medida han sido alcanzados).

Las simulaciones realizadas, que comprenden todos los esquemas de control diseñados previamente, permiten visualizar su comportamiento y definir si el sistema es funcional o si requiere ser modificado para mejorar su desempeño.

## 4.1. Sistema de Localización

Se realizó la simulación del algoritmo en Gazebo, los resultados obtenidos se muestran a continuación. Diríjase a la sección de apéndices para conocer más acerca del código implementado.

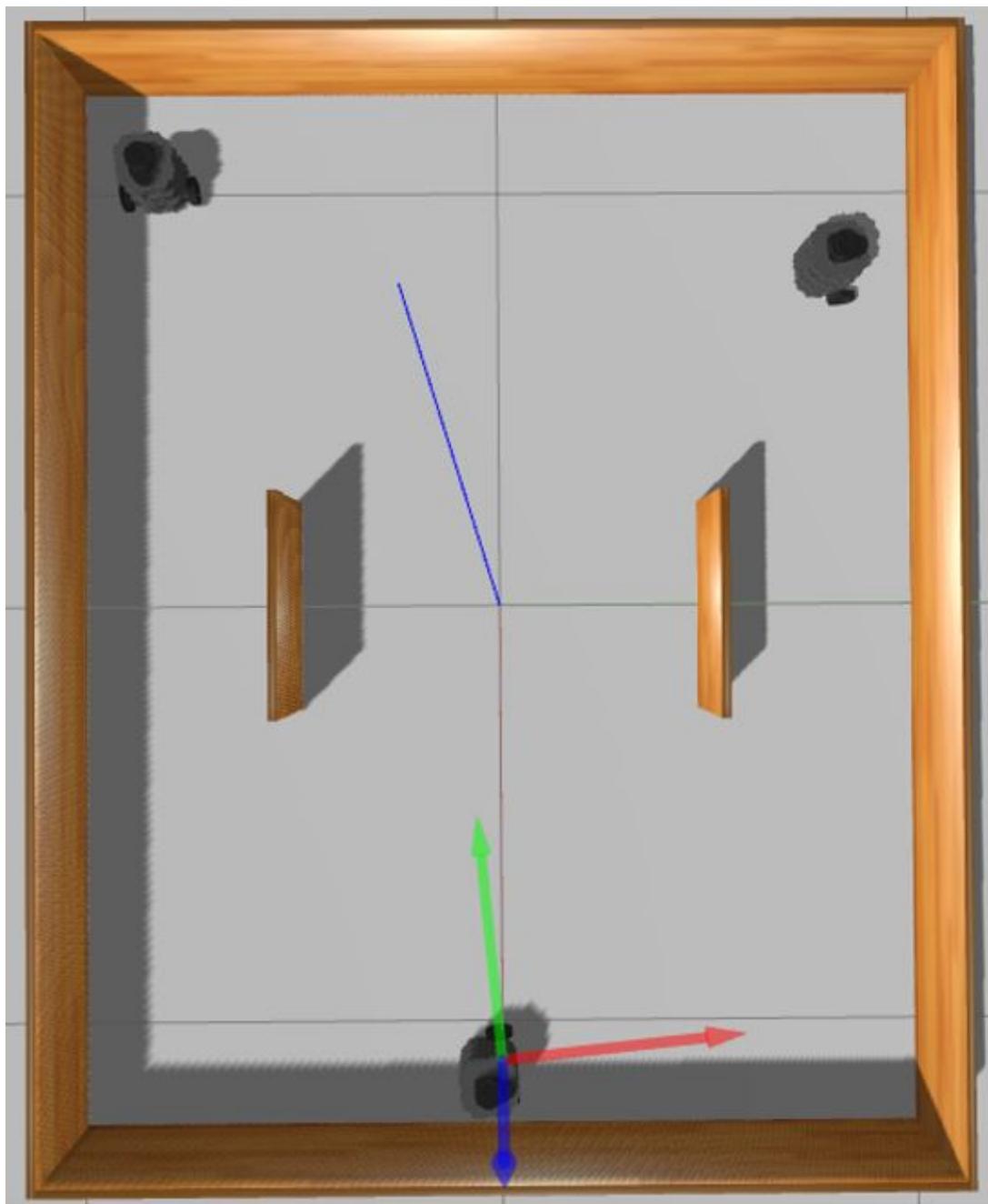
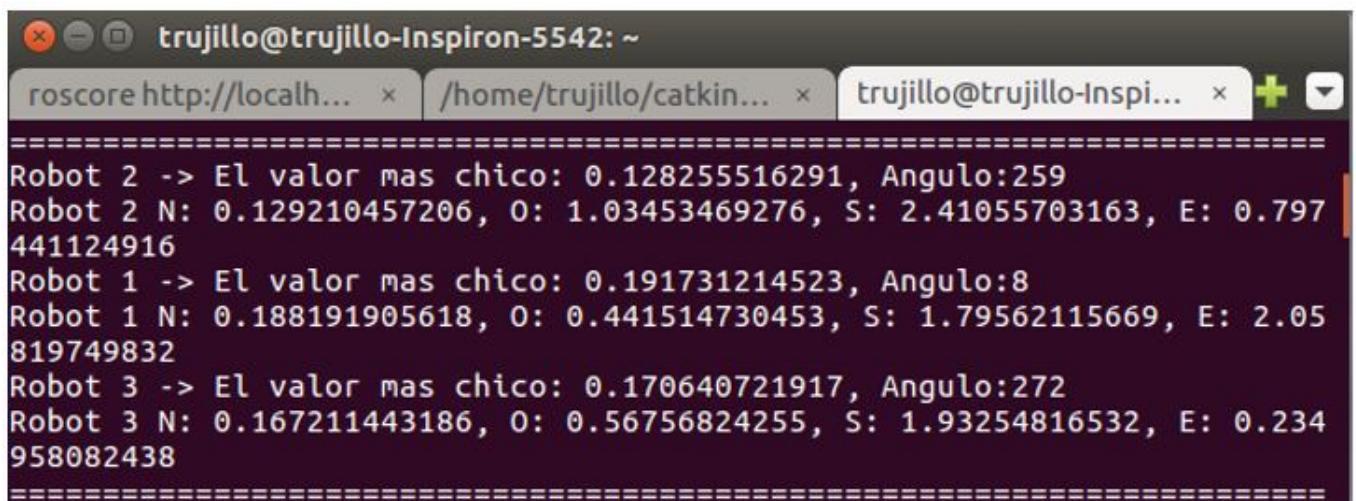


Figura 4.1: Simulación del sistema de localización.



```
trujillo@trujillo-Inspiron-5542: ~
roscore http://localhost:11311
/home/trujillo/catkin_ws/devel/lib/localization/localization_node
=====
Robot 2 -> El valor mas chico: 0.128255516291, Angulo:259
Robot 2 N: 0.129210457206, O: 1.03453469276, S: 2.41055703163, E: 0.797
441124916
Robot 1 -> El valor mas chico: 0.191731214523, Angulo:8
Robot 1 N: 0.188191905618, O: 0.441514730453, S: 1.79562115669, E: 2.05
819749832
Robot 3 -> El valor mas chico: 0.170640721917, Angulo:272
Robot 3 N: 0.167211443186, O: 0.56756824255, S: 1.93254816532, E: 0.234
958082438
=====
```

Figura 4.2: Mediciones del sistema de localización.

Como se observa en la figura 4.2, el algoritmo encuentra para cada robot la distancia más pequeña medida por el sensor LiDAR, que es el valor asignado al norte y las mediciones de los siguientes puntos cardinales son valores pertinentes si se realiza una inspección visual a la figura 4.1. Por esto, se considera que el avance propuesto para este sistema funciona de manera adecuada, aunque aún no está terminado debido a limitaciones del software de simulación. Es necesario trabajar con los robots en físico para poder concluir con este.

## 4.2. Sistema de Asignación de roles

A continuación se muestran los resultados de las pruebas realizadas en Gazebo. Diríjase a la sección de apéndices para conocer más acerca del código implementado.

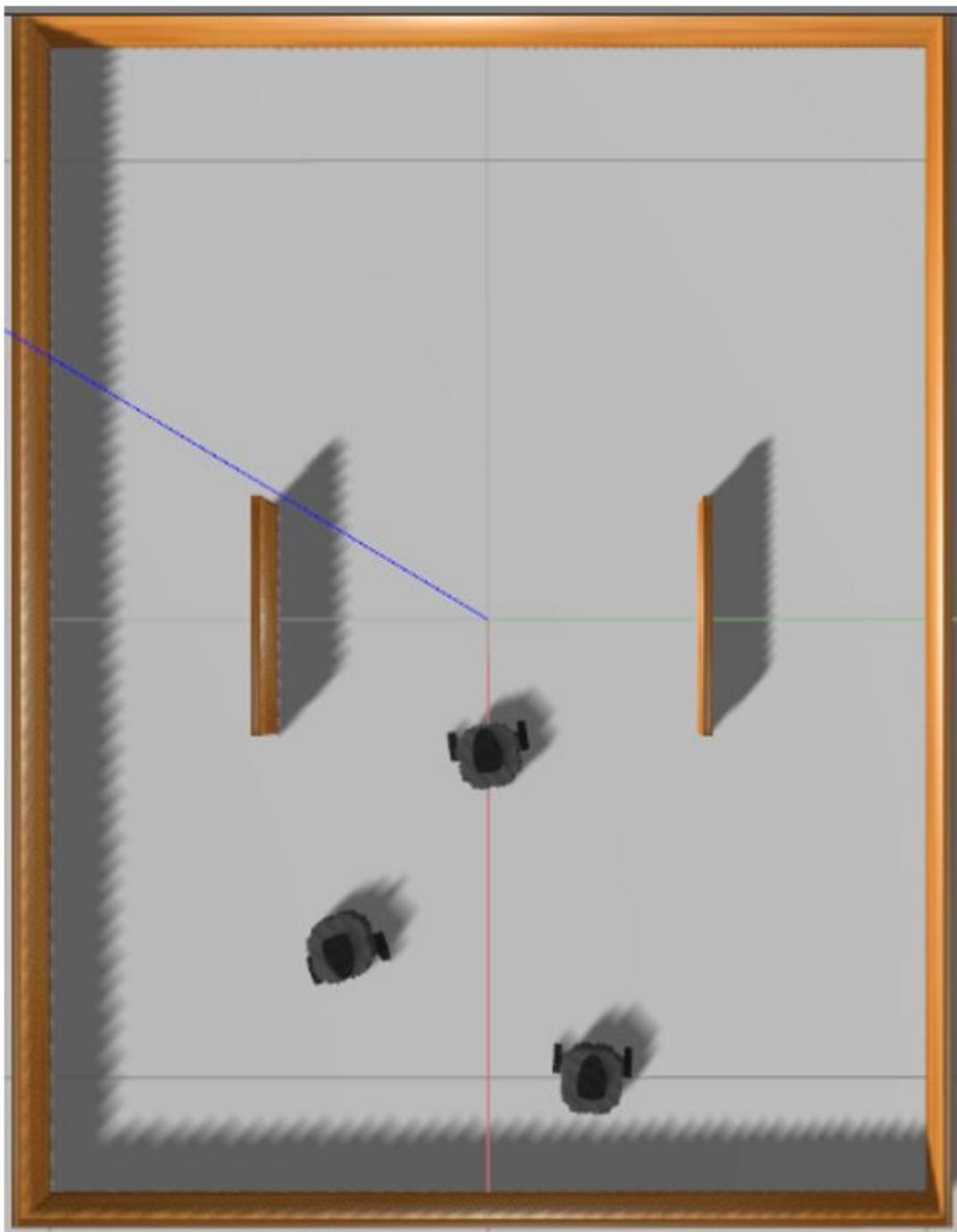
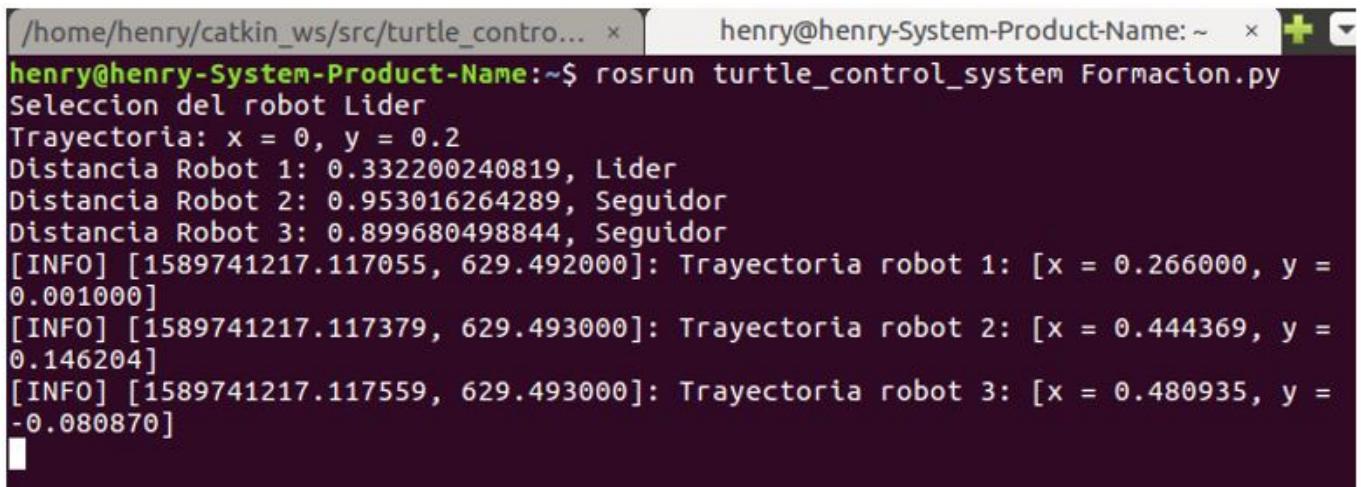


Figura 4.3: Simulación del sistema de asignación de roles.

## Pruebas y Resultados



```
/home/henry/catkin_ws/src/turtle_contro... x henry@henry-System-Product-Name:~ x + -> henry@henry-System-Product-Name:~$ rosrun turtle_control_system Formacion.py  
Seleccion del robot Lider  
Trayectoria: x = 0, y = 0.2  
Distancia Robot 1: 0.332200240819, Lider  
Distancia Robot 2: 0.953016264289, Seguidor  
Distancia Robot 3: 0.899680498844, Seguidor  
[INFO] [1589741217.117055, 629.492000]: Trayectoria robot 1: [x = 0.266000, y = 0.001000]  
[INFO] [1589741217.117379, 629.493000]: Trayectoria robot 2: [x = 0.444369, y = 0.146204]  
[INFO] [1589741217.117559, 629.493000]: Trayectoria robot 3: [x = 0.480935, y = -0.080870]
```

Figura 4.4: Resultados en consola del sistema de asignación de roles.

Los resultados demuestran un buen funcionamiento del sistema, ya que, asignando el punto destino y comparando las distancias que hay entre cada robot hacia este punto, si se selecciona el robot que tenga la distancia mínima como robot líder.

## 4.3. Sistema de Navegación

### Seguimiento de Trayectoria: Control P

A continuación, se presentan los resultados obtenidos de la simulación, diríjase a la sección de apéndices para conocer más acerca del código implementado.

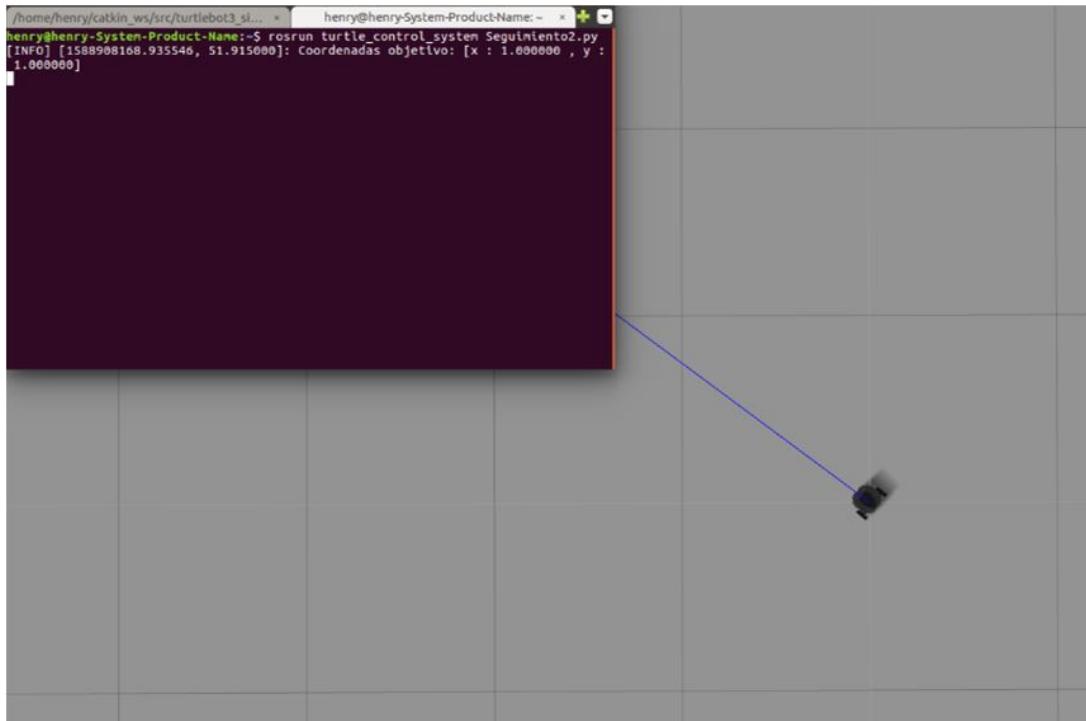


Figura 4.5: Simulación del sistema de seguimiento de trayectoria (Control P) Punto 1.

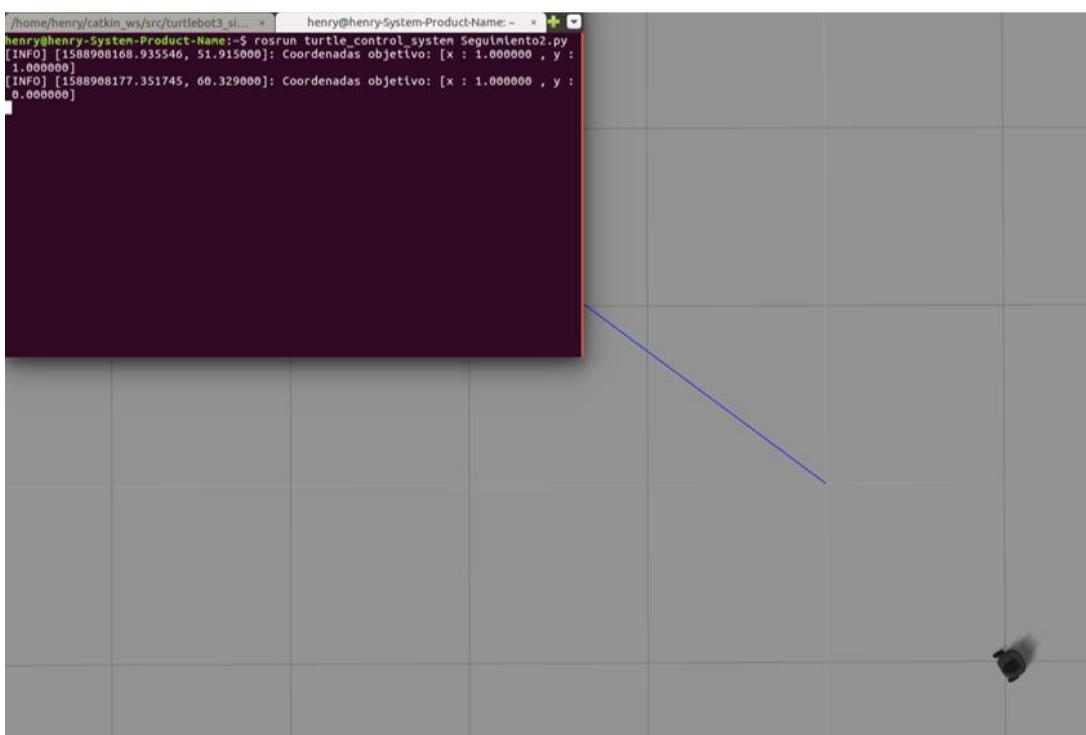


Figura 4.6: Simulación del sistema de seguimiento de trayectoria (Control P) Punto 2.

## Pruebas y Resultados

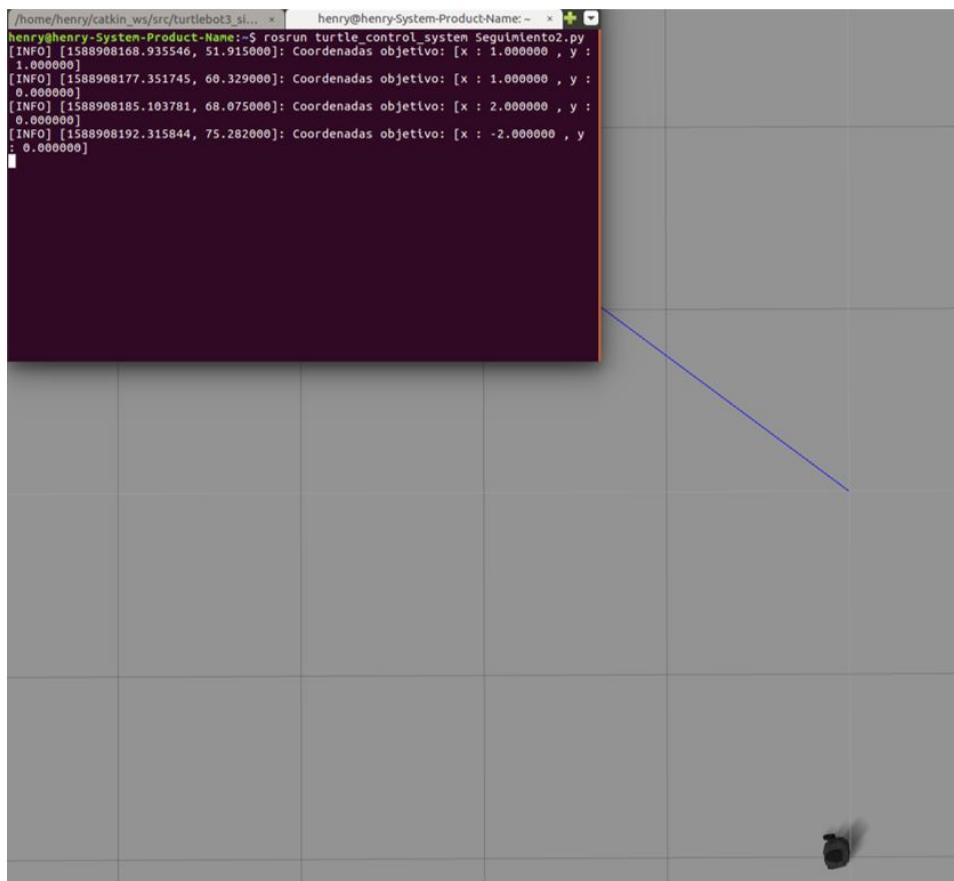


Figura 4.7: Simulación del sistema de seguimiento de trayectoria (Control P) Punto 3.

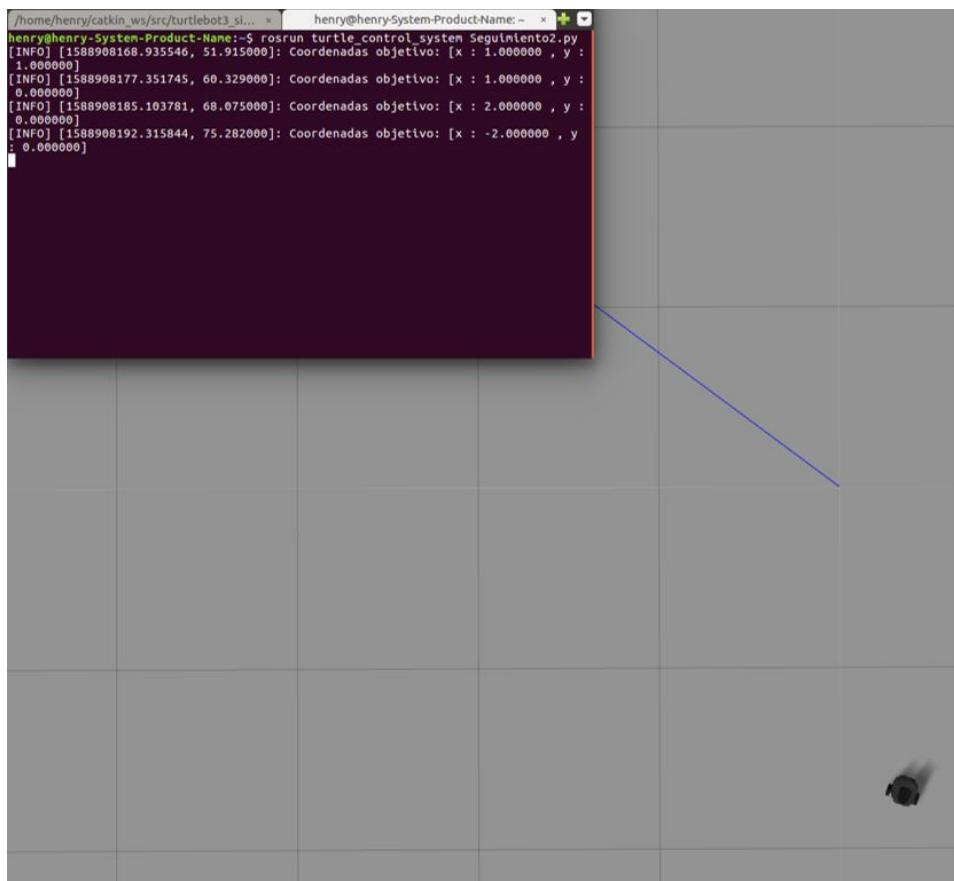


Figura 4.8: Simulación del sistema de seguimiento de trayectoria (Control P) Punto 4.

Al realizar las simulaciones, se observa un comportamiento correcto del sistema al momento de seguir una trayectoria dada. El controlador para la velocidad linear funciona sin ningún problema, pero con la velocidad angular se observaron detalles que hacían que el sistema fallara. El problema principal fue causado por la limitación que se tiene al momento de conocer la rotación del robot, la cual está dada en un rango de  $-\pi < \theta < \pi$ , donde si se daba el caso de que el robot tuviera que rotar a valores cercanos a los límites, el sistema podía desestabilizarse y empezar a rotar continuamente.

Para abordar esta problemática, se realizaron pruebas utilizando las transformaciones homogéneas para cambiar la posición que daban los robots del marco de referencia general, al marco de referencia relativo al robot. Las expresiones para este cambio de marco de referencia son las siguientes:

$$P = HP^* \quad (4.1)$$

$$P^* = H^{-1}P \quad (4.2)$$

Donde:

$P$ =Posición con respecto al marco de referencia fijo

$P^*$ =Posición con respecto al marco de referencia nuevo

$H$ =Matriz de transformación homogénea

La matriz  $H$  se obtiene al inicio de la simulación, con la rotación y posición que tenga inicialmente el robot. Con ella, es posible saber constantemente la posición del robot con respecto al marco de referencia relativo a este, y transformar el punto destino a dicho marco de referencia para poder calcular la distancia euclídea  $d$  y el ángulo  $\alpha$  dentro del mismo. Para efectos prácticos, la matriz  $H$  es calculada nuevamente cada vez que el robot llega al punto destino.

Realizando nuevamente las pruebas, se observa una reducción al problema que se tuvo, sin embargo, no fue una solución permanente, ya que este mismo problema se podría presentar. Al trabajar con transformaciones homogéneas, fue posible agregar una excepción ante este problema, el cual consiste en asignar una rotación con valor igual a  $\pi$  al marco de referencia que se está manejando. Diríjase a la sección de apéndices para conocer cómo se implementó.

## Pruebas y Resultados

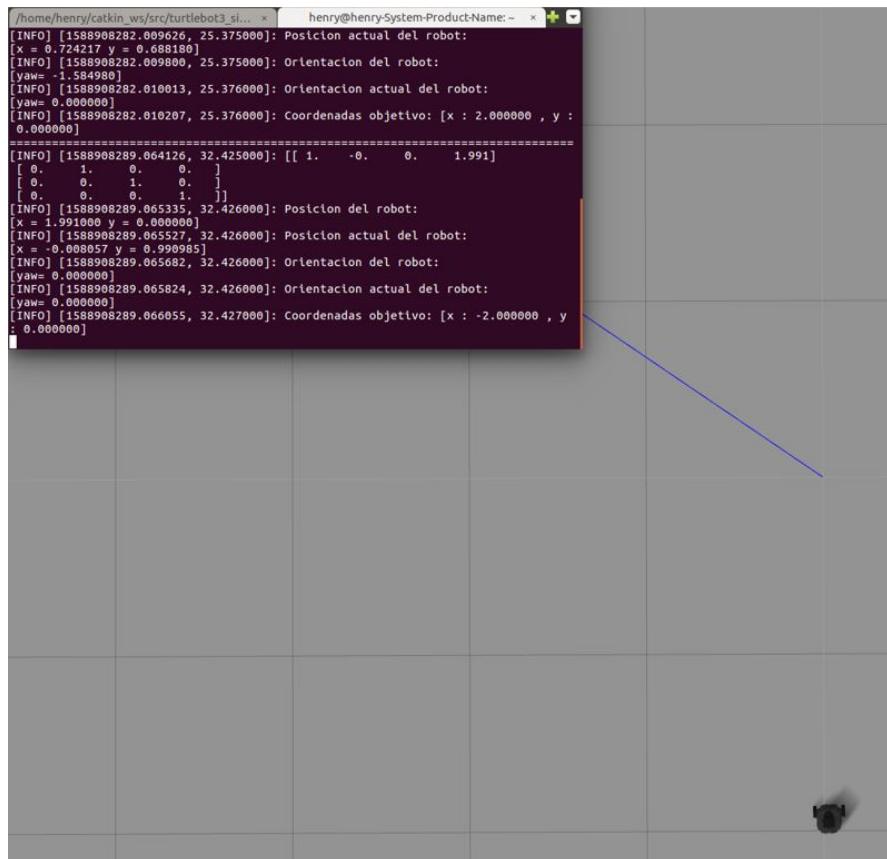


Figura 4.9: Simulación del sistema de seguimiento (Control P) modificado Punto 1.

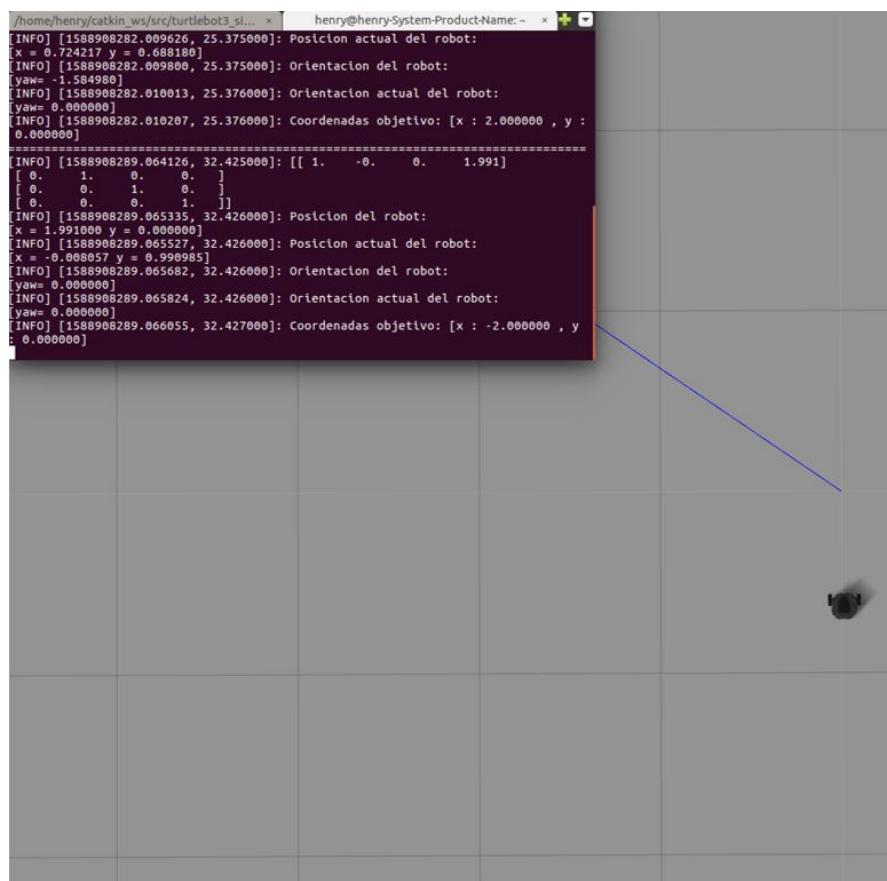


Figura 4.10: Simulación del sistema de seguimiento (Control P) modificado Punto 2.

```
[x = -0.008057 y = 0.990985] henry@henry-System-Product-Name:~ + 
[INFO] [1588908289.065682, 32.426000]: Orientacion del robot:
[yaw= 0.00000]
[INFO] [1588908289.065824, 32.426000]: Orientacion actual del robot:
[yaw= 0.00000]
[INFO] [1588908289.066055, 32.427000]: Coordenadas objetivo: [x : -2.000000 , y : 0.000000]
=====
[INFO] [1588908312.204293, 55.547000]: [[ -9.99800020e-01 1.99980002e-02 0.0000000e+00 -1.99400000e+00]
[ -1.9998002e-02 -9.9980020e-01 0.00000000e+00 1.00000000e-03]
[ 0.00000000e+00 0.00000000e+00 1.00000000e+00 0.00000000e+00]
[ 0.00000000e+00 0.00000000e+00 0.00000000e+00 1.00000000e+00]]
[INFO] [1588908312.205513, 55.548000]: Posicion del robot:
[x = -1.994000 y = 0.001000]
[INFO] [1588908312.205726, 55.549000]: Posicion actual del robot:
[x = 2.747000 y = 0.015000]
[INFO] [1588908312.205876, 55.549000]: Orientacion del robot:
[yaw= -3.121593]
[INFO] [1588908312.206059, 55.549000]: Orientacion actual del robot:
[yaw= 0.000000]
[INFO] [1588908312.206251, 55.549000]: Coordenadas objetivo: [x : -1.000000 , y : 1.000000]
```

Figura 4.11: Simulación del sistema de seguimiento (Control P) modificado Punto 3.

## Pruebas y Resultados

---

### Seguimiento de Trayectoria: Control Difuso

Se realizó la simulación del algoritmo con ayuda de Gazebo, utilizando un escenario vacío, las coordenadas del punto al que se quiere llegar es (1,1), las siguientes imágenes muestran el desplazamiento del robot.

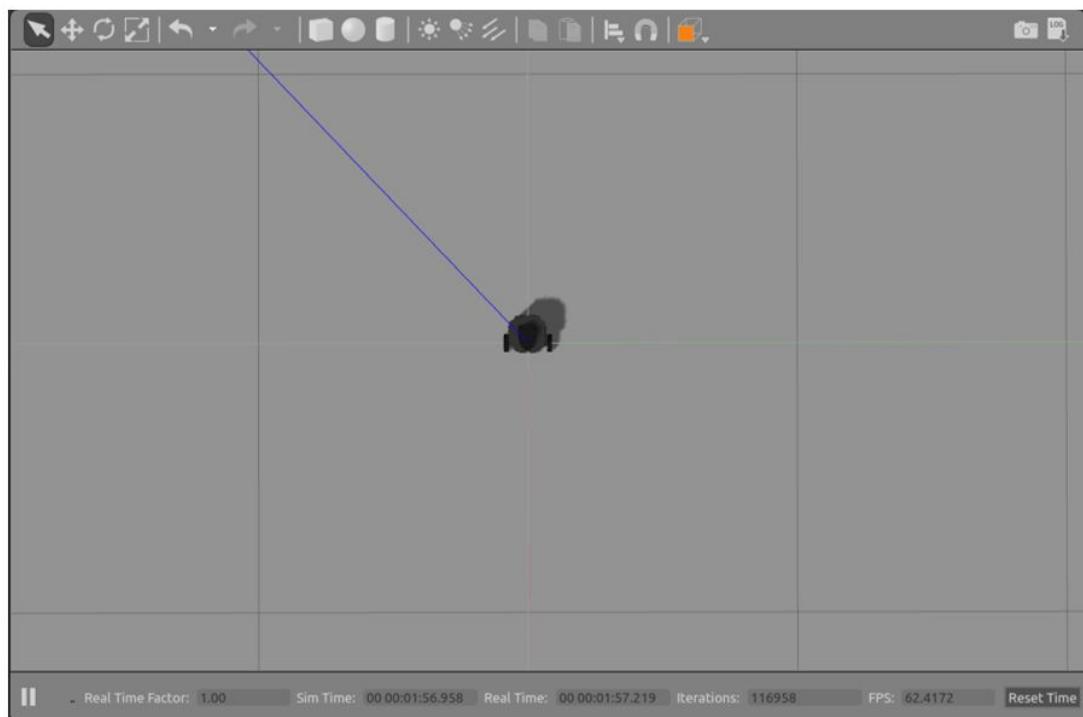


Figura 4.12: Simulación del sistema de seguimiento (Control Difuso) 1a prueba.

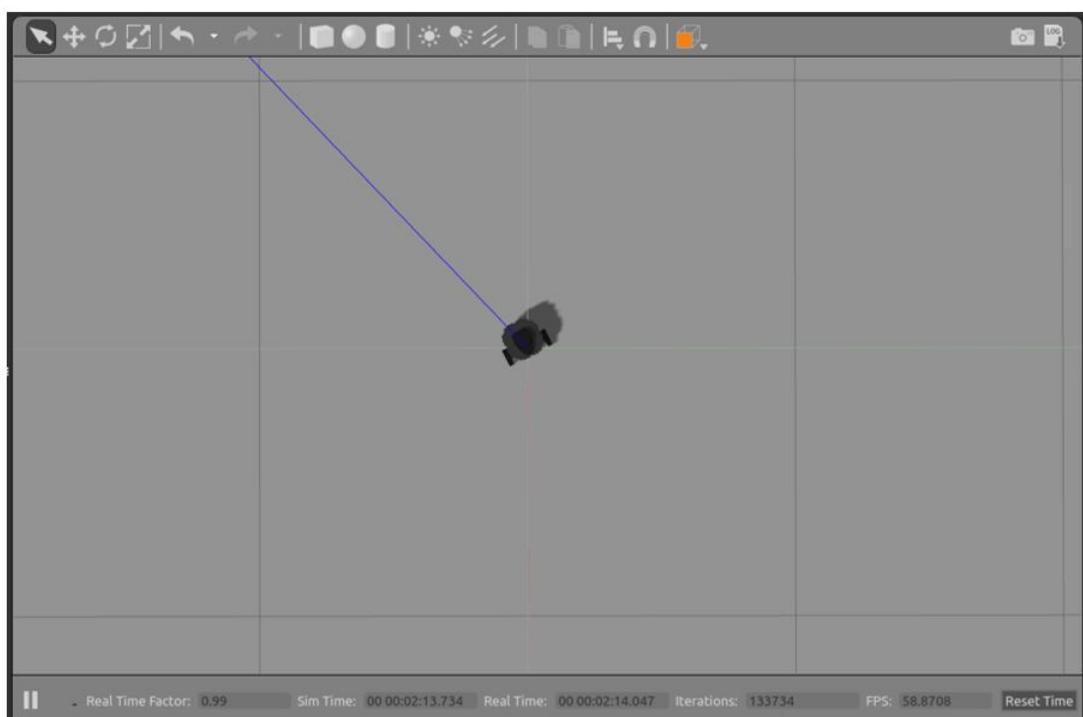


Figura 4.13: Simulación del sistema de seguimiento (Control Difuso) 1a prueba.

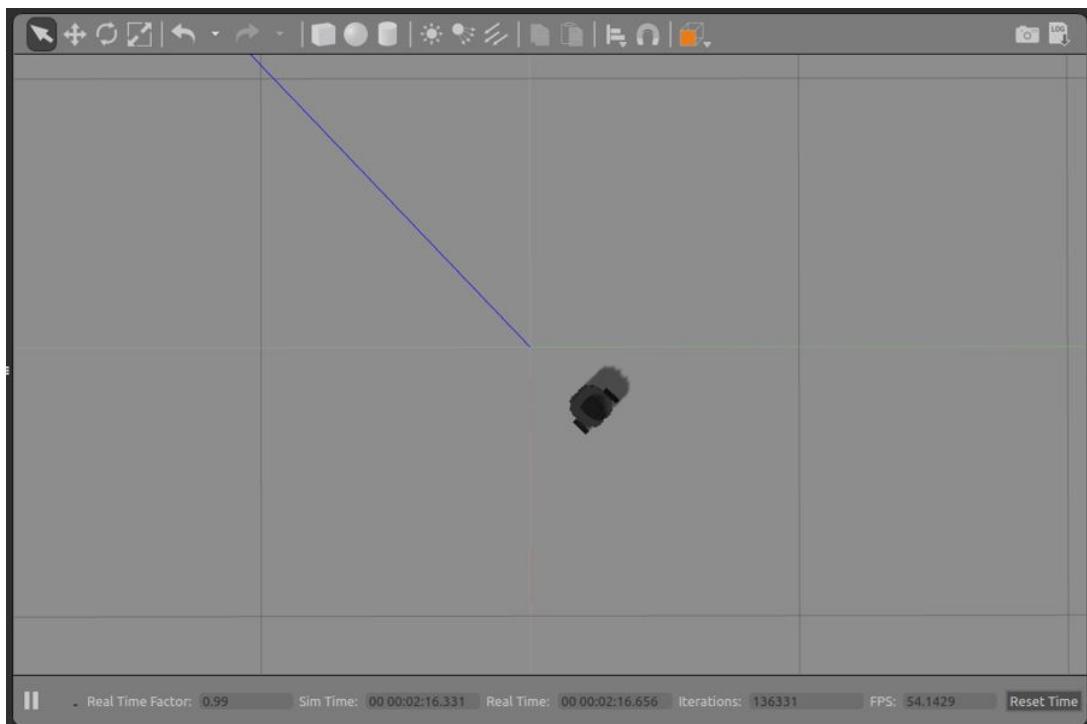


Figura 4.14: Simulación del sistema de seguimiento (Control Difuso) 1a prueba.

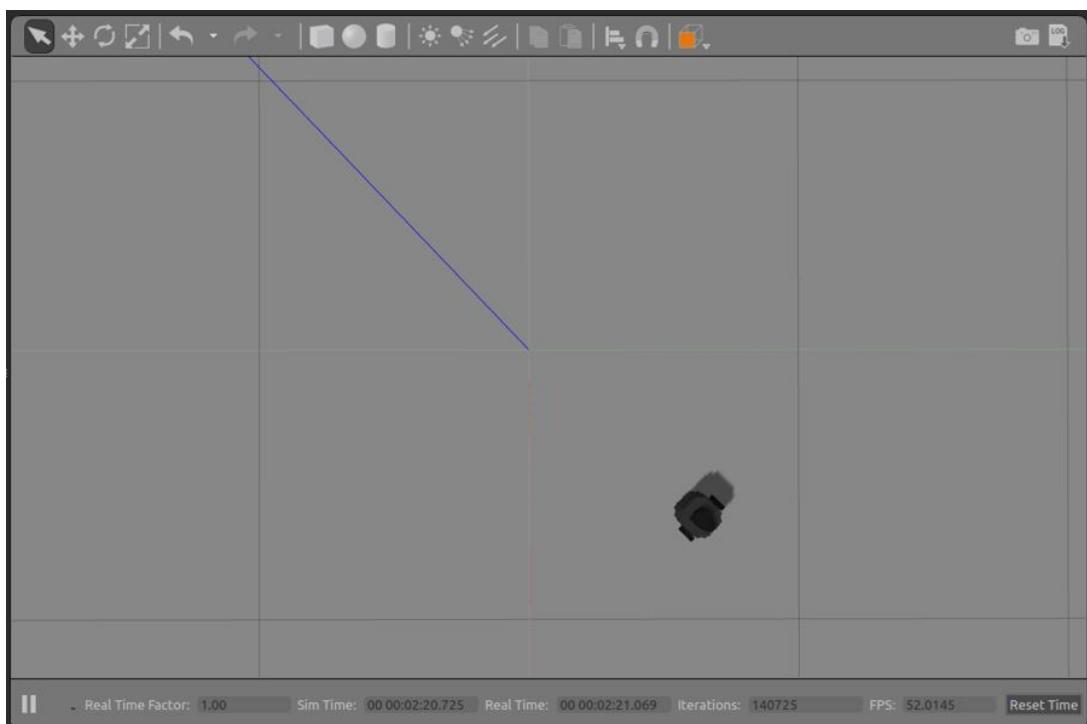


Figura 4.15: Simulación del sistema de seguimiento (Control Difuso) 1a prueba.

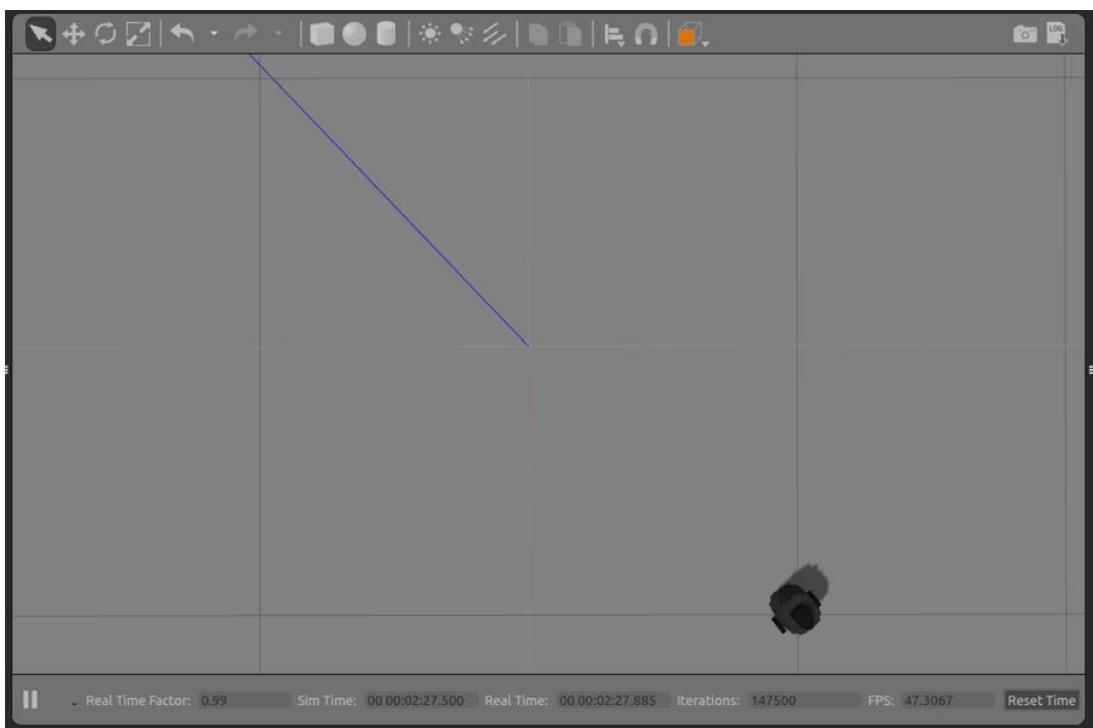


Figura 4.16: Simulación del sistema de seguimiento (Control Difuso) 1a prueba.

En la gráfica del error, se puede observar que al inicio el error de distancia es positivo y tarda alrededor de 15 segundos en hacerse cero. En cambio, en el error del ángulo, su comportamiento es oscilatorio alrededor del cero.

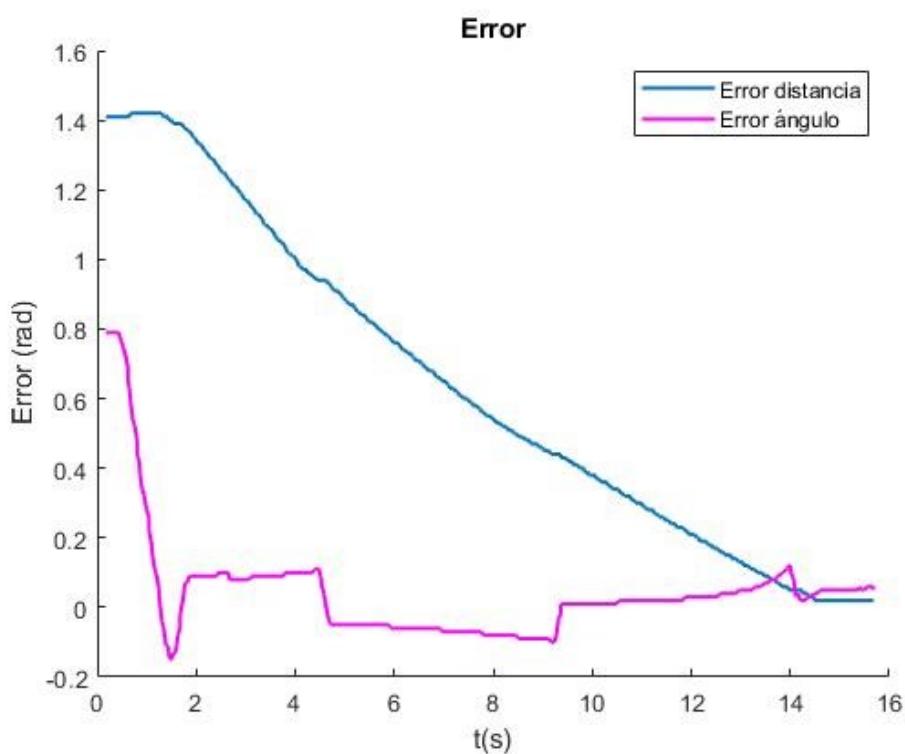


Figura 4.17: Gráfica de los errores del sistema de seguimiento (Control Difuso) 1a prueba.

También se puede ver la gráfica de la posición, en ambos ejes se alcanza la posición deseada al mismo tiempo.

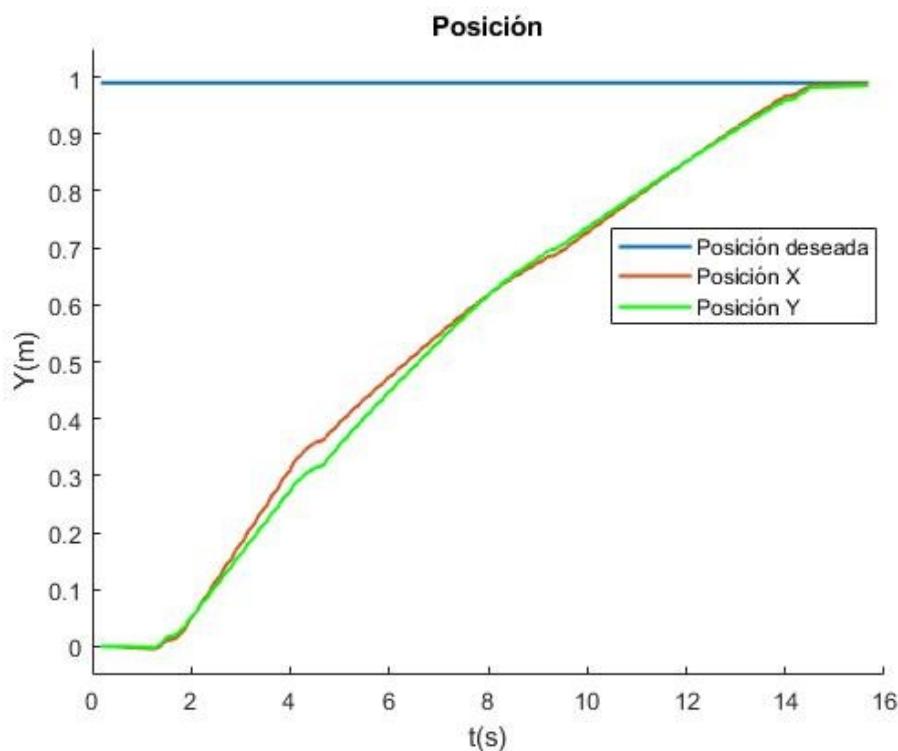


Figura 4.18: Gráfica de la posición del sistema de seguimiento (Control Difuso) 1a prueba.

De igual forma, se realizó la simulación del algoritmo en el escenario pruebas propuesto, las coordenadas del punto al que se quiere llegar es (0.8, -0.8), las siguientes imágenes muestran el desplazamiento del robot. Así como las gráficas de error y la posición del robot durante la simulación.

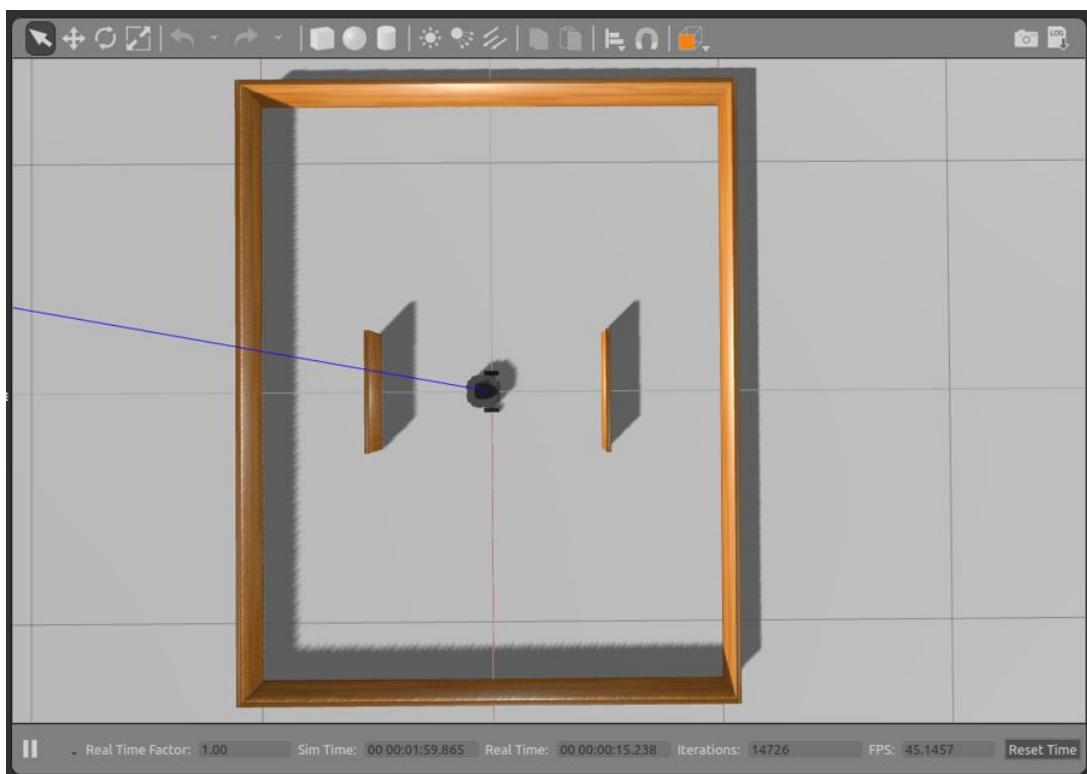


Figura 4.19: Simulación del sistema de seguimiento (Control Difuso) 2a prueba.

## Pruebas y Resultados

.....

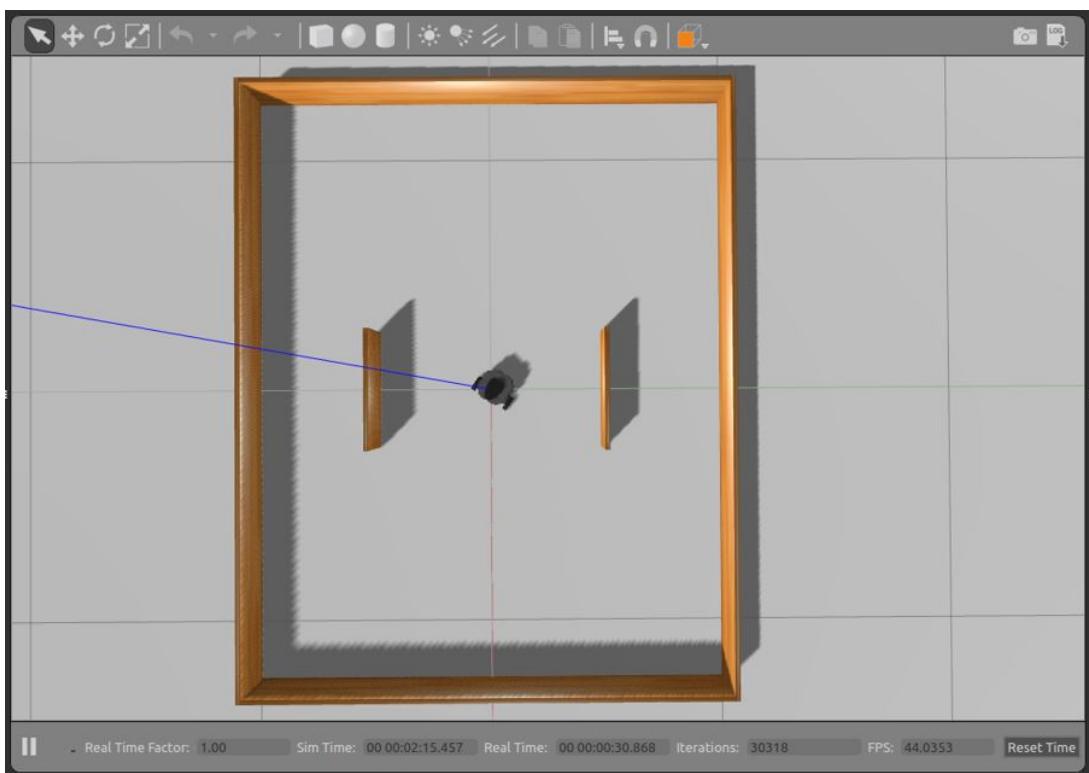


Figura 4.20: Simulación del sistema de seguimiento (Control Difuso) 2a prueba.

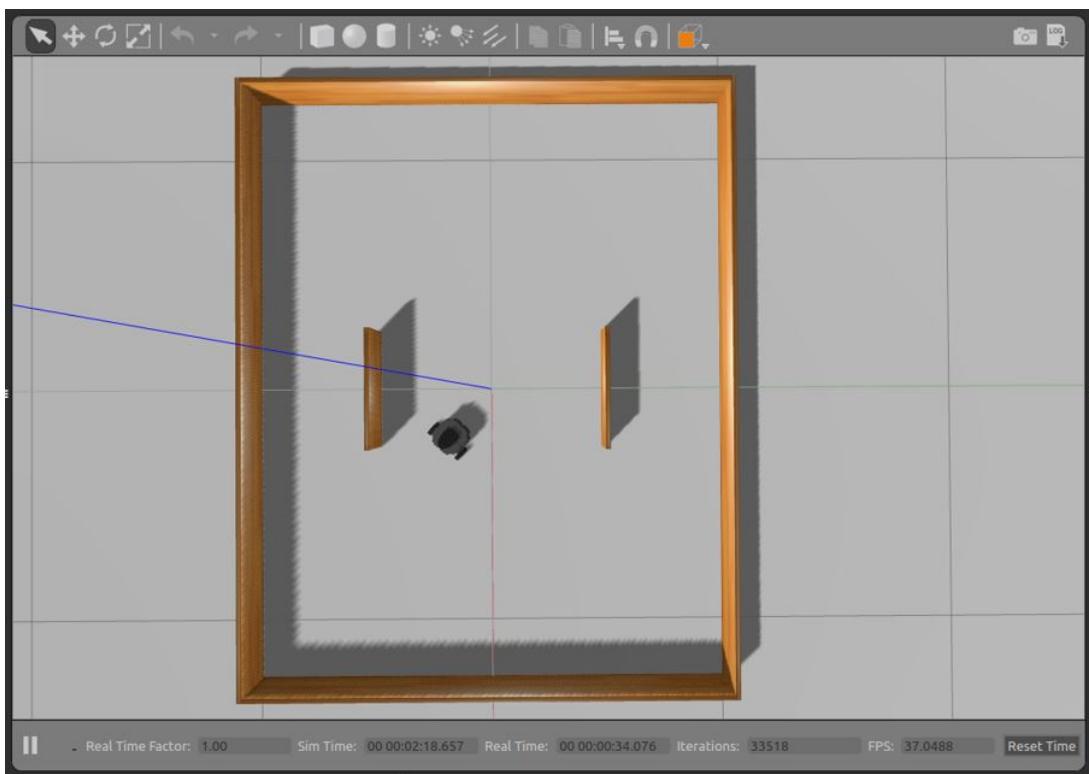


Figura 4.21: Simulación del sistema de seguimiento (Control Difuso) 2a prueba.

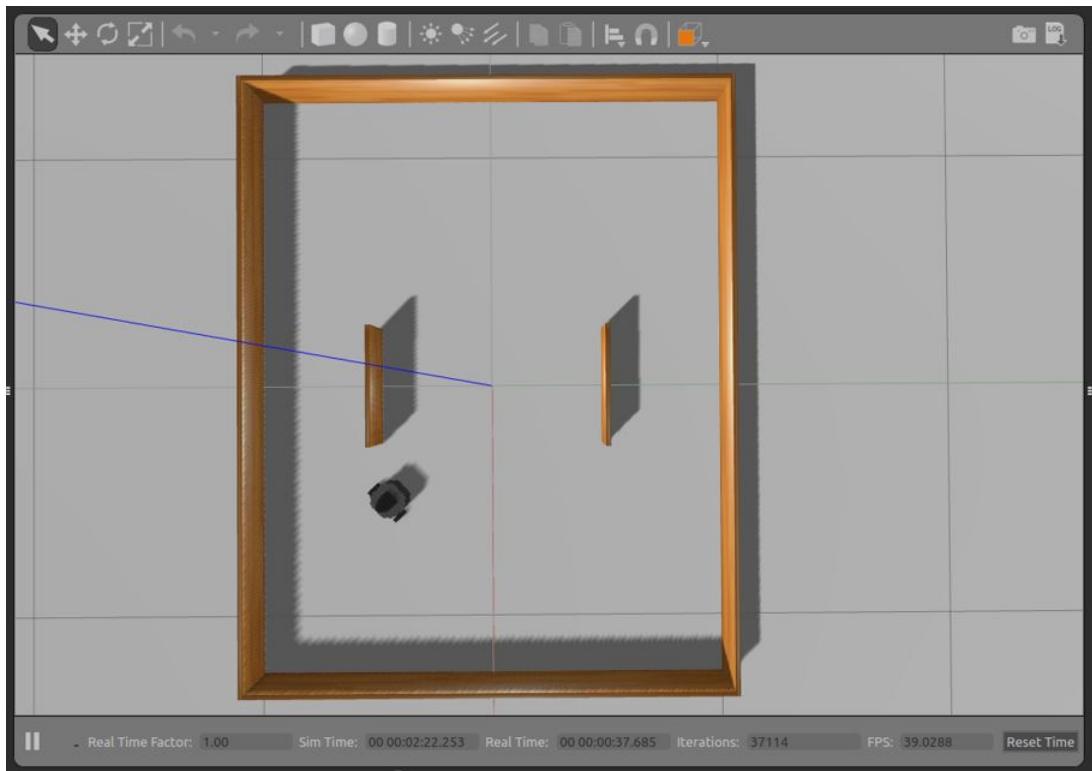


Figura 4.22: Simulación del sistema de seguimiento (Control Difuso) 2a prueba.

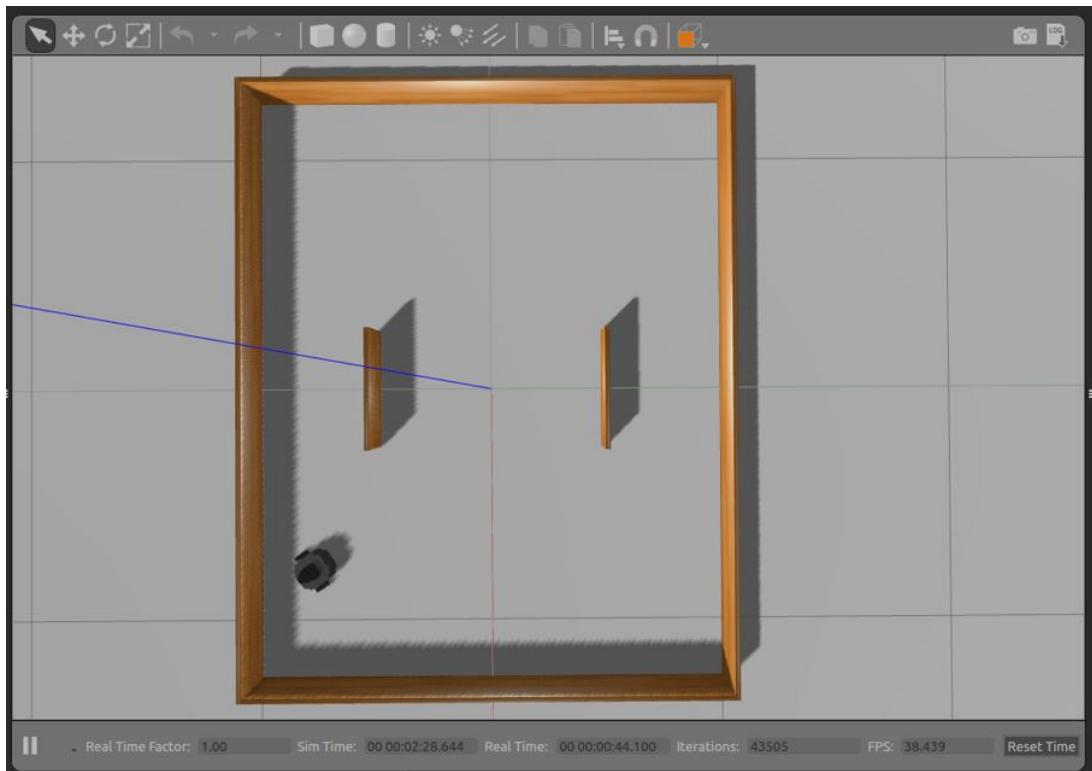


Figura 4.23: Simulación del sistema de seguimiento (Control Difuso) 2a prueba.

## Pruebas y Resultados

---

En la gráfica de error, se puede ver que el error de ángulo presenta una respuesta oscilatoria respecto al cero, mientras que el error de distancia tiene una respuesta gradual.

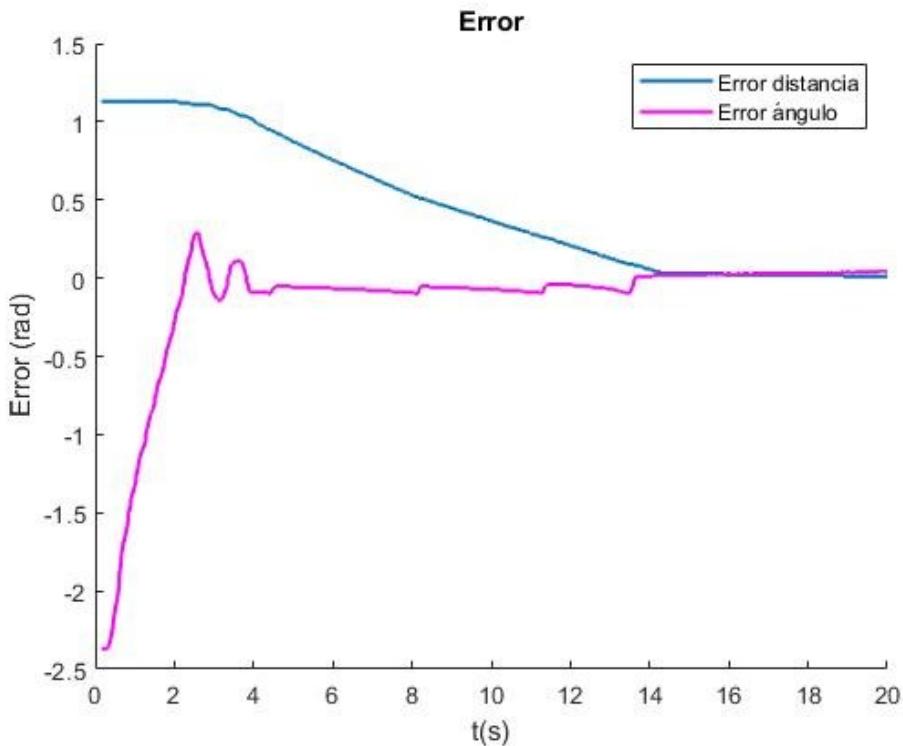


Figura 4.24: Gráfica de los errores del sistema de seguimiento (Control Difuso) 2a prueba.

En la gráfica de la posición del robot, en ambos ejes se llega a la posición deseada al mismo tiempo.

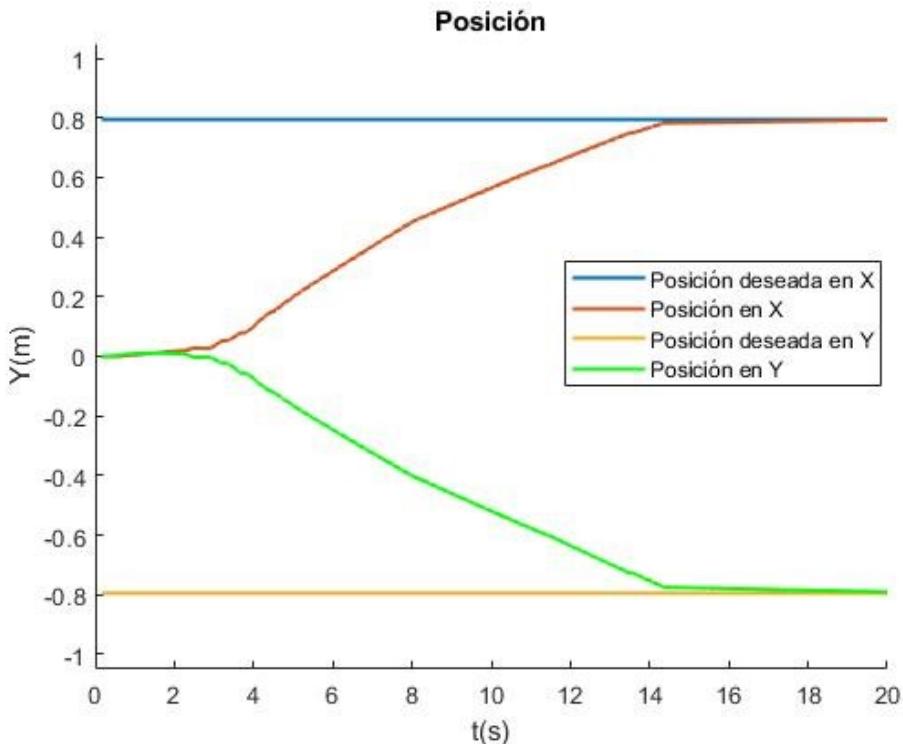


Figura 4.25: Gráfica de la posición del sistema de seguimiento (Control Difuso) 2a prueba.

Como se observó, el algoritmo de control funciona como se esperaba, aun se puede mejorar agregando una ganancia para que se reduzca el tiempo en que llega al objetivo, o se puede probar cambiando algunos parámetros de las funciones de membresía para que el desplazamiento del robot sea más rápido.

## Pruebas y Resultados

---

### Evasión de Obstáculos

Se realizó la simulación del algoritmo con ayuda de Gazebo, utilizando el escenario de pruebas propuesto, las siguientes imágenes muestran el desplazamiento del robot. Diríjase a la sección de apéndices para visualizar el código.

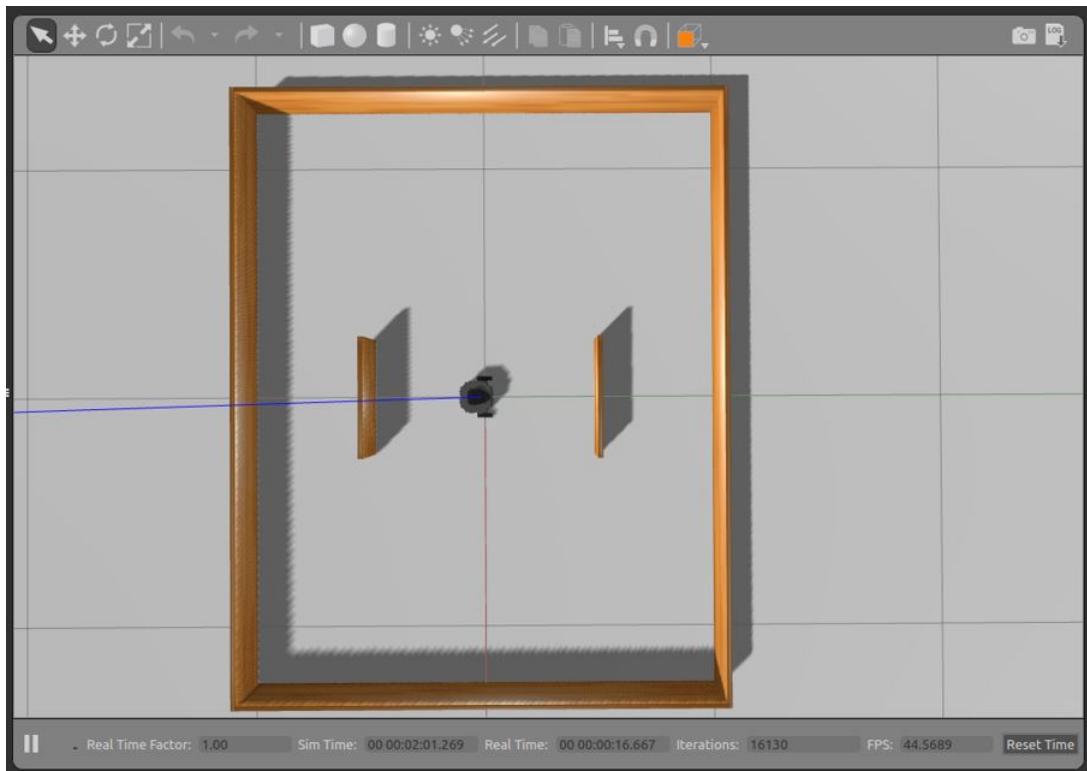


Figura 4.26: Simulación sistema de evasión.

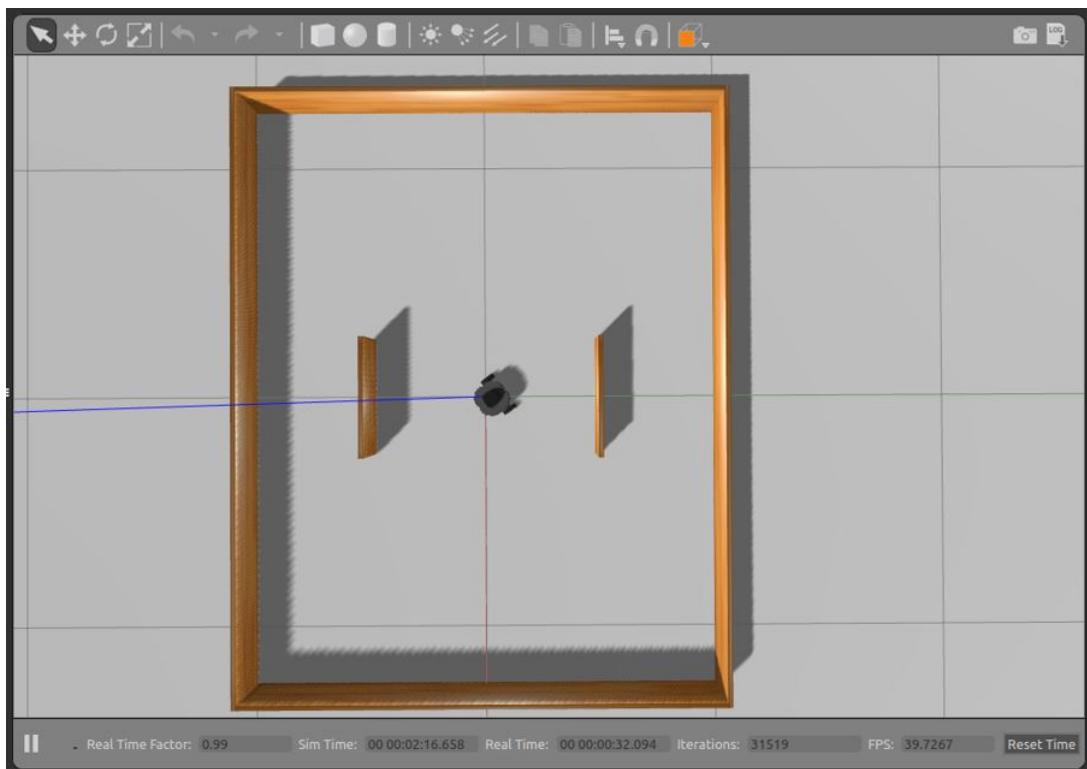


Figura 4.27: Simulación sistema de evasión.

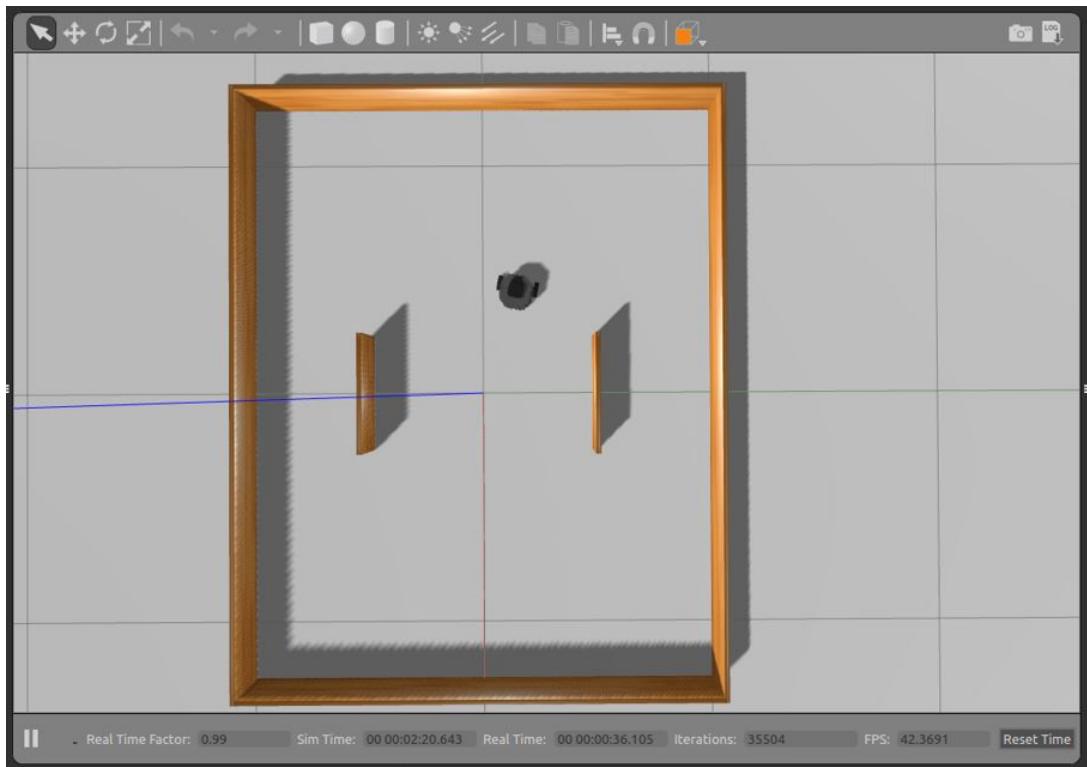


Figura 4.28: Simulación sistema de evasión.

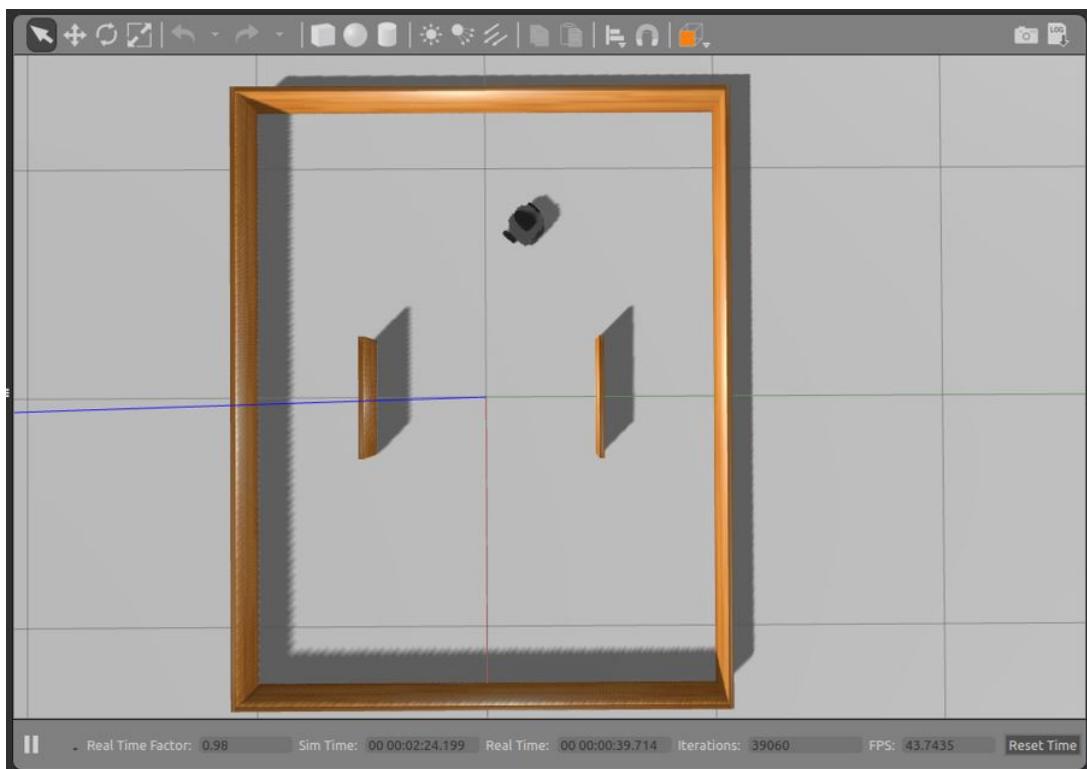


Figura 4.29: Simulación sistema de evasión.

## Pruebas y Resultados

.....

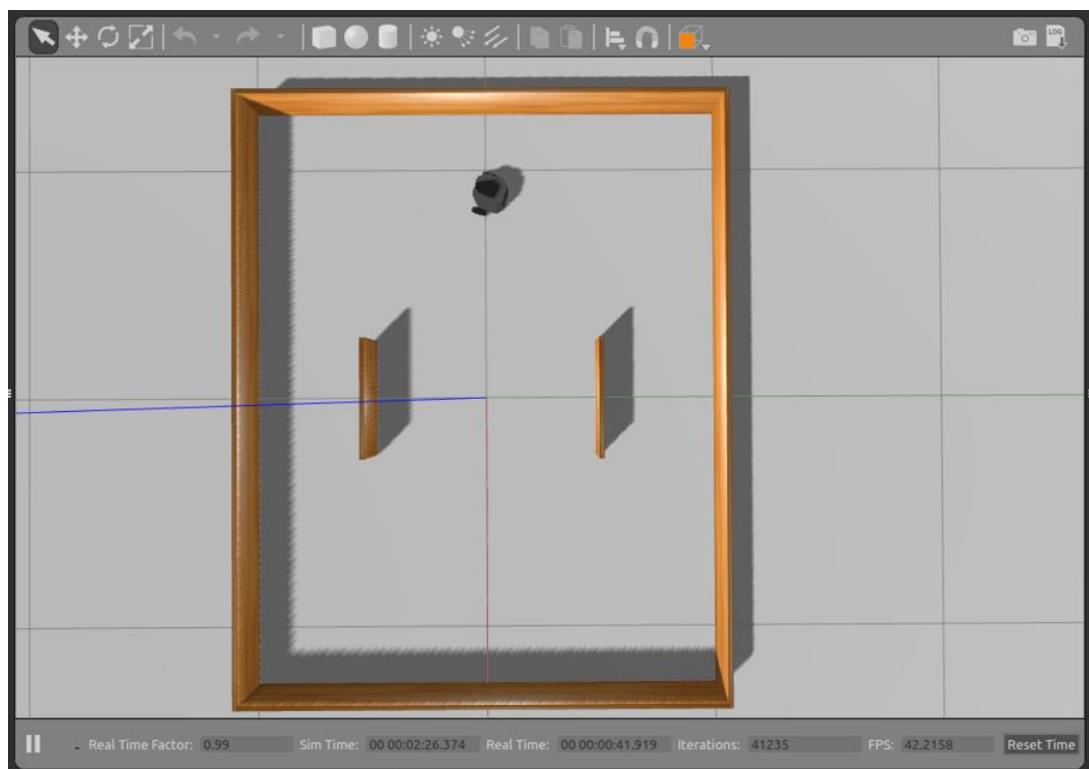


Figura 4.30: Simulación sistema de evasión.

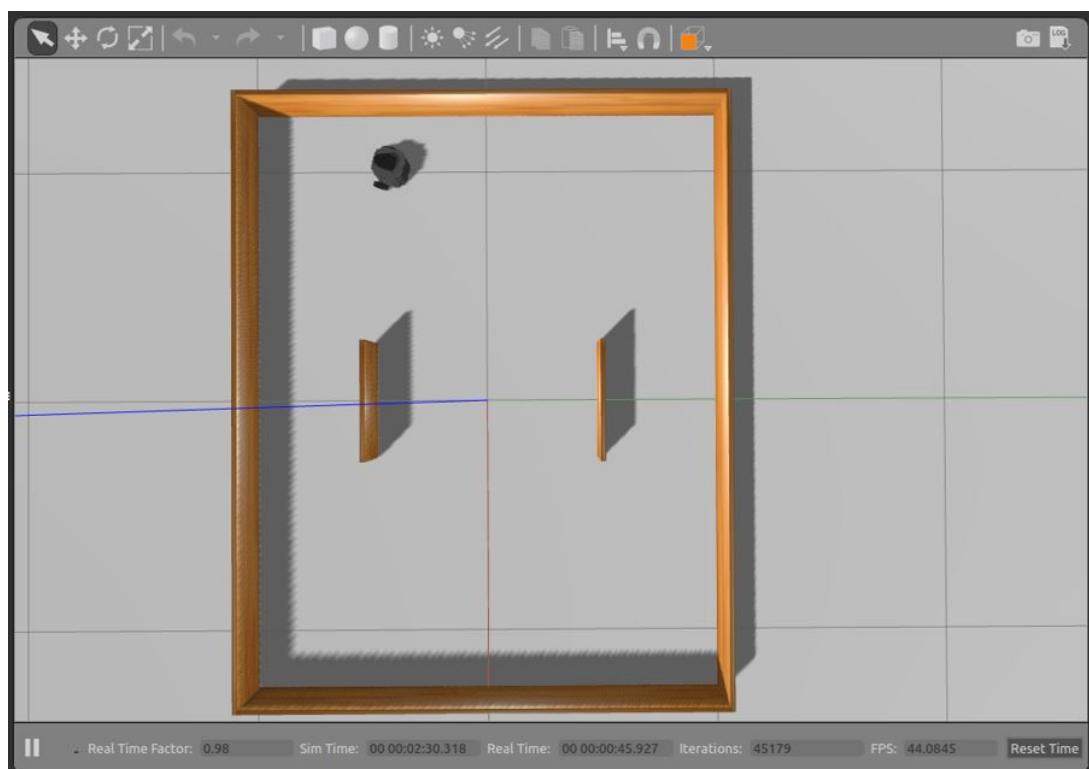


Figura 4.31: Simulación sistema de evasión.

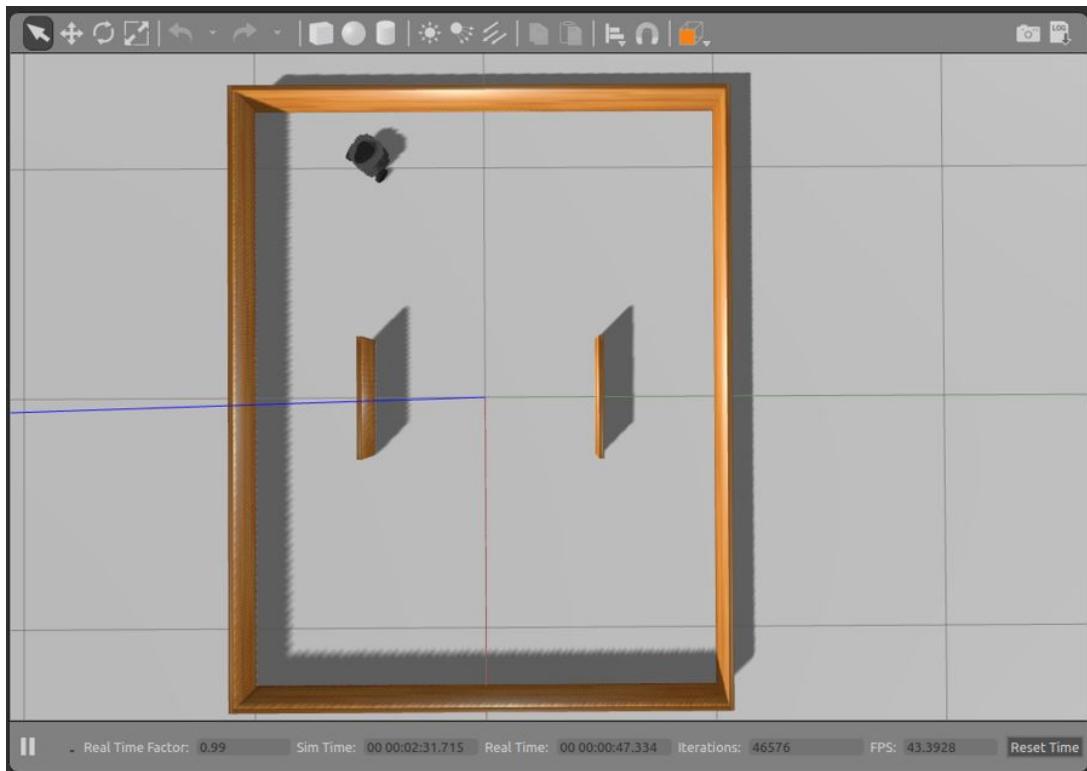


Figura 4.32: Simulación sistema de evasión.

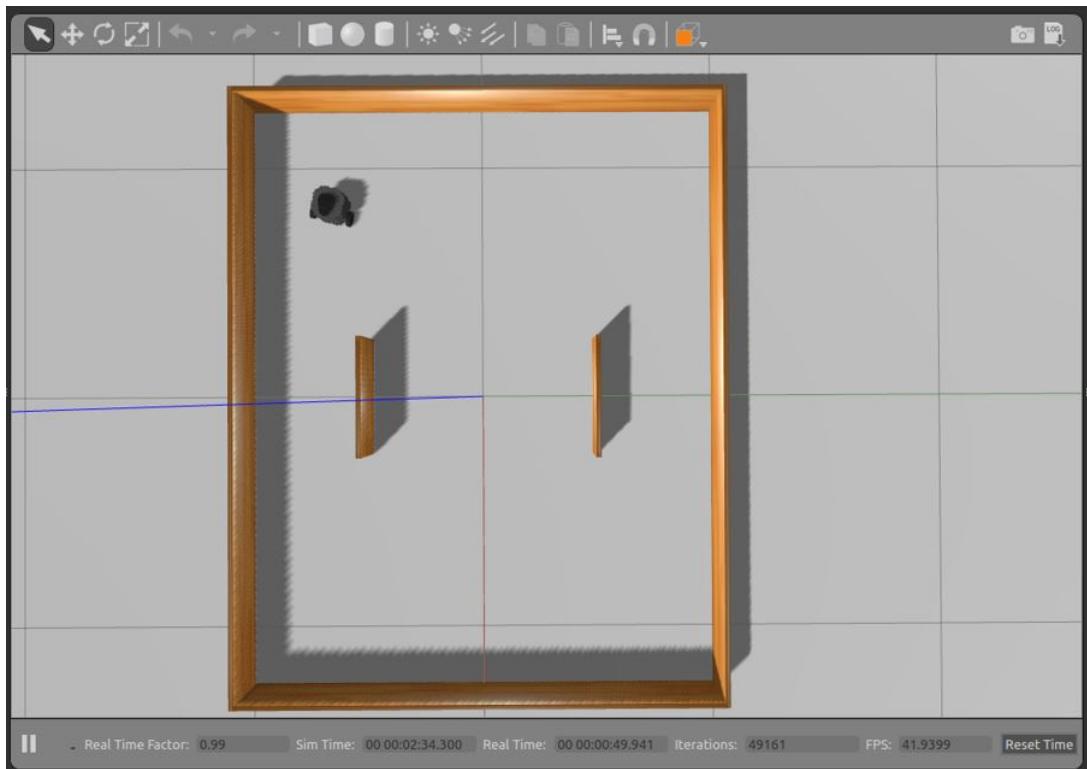


Figura 4.33: Simulación sistema de evasión.

## Pruebas y Resultados

---

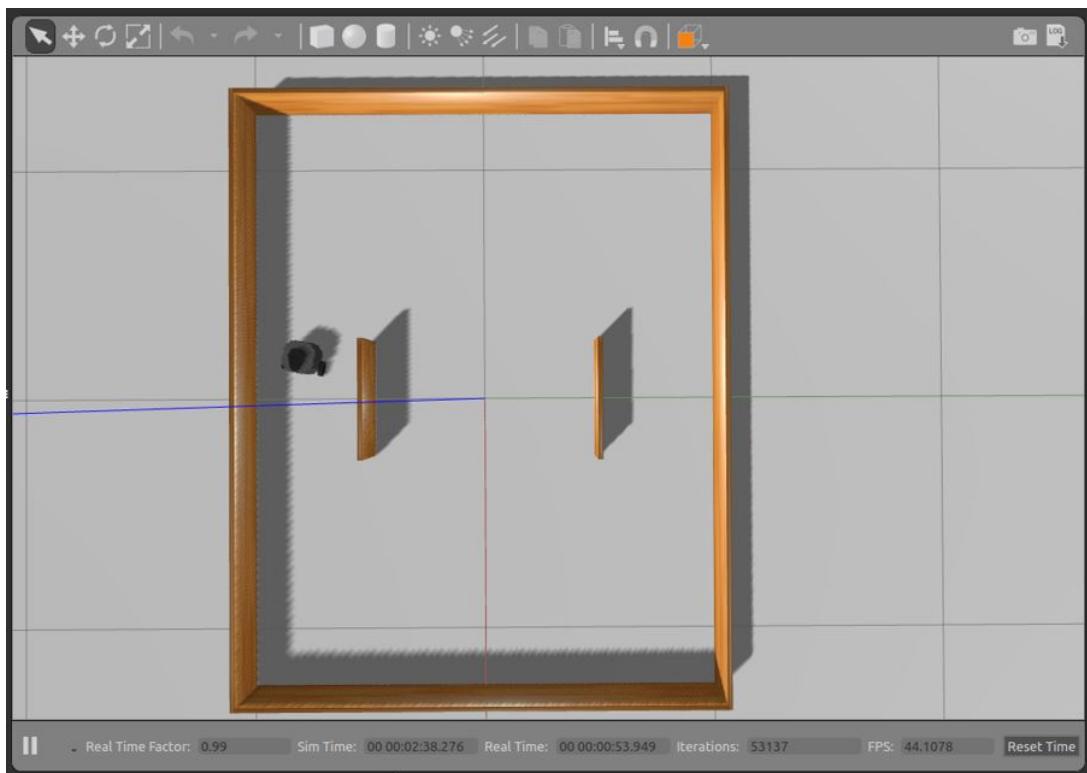


Figura 4.34: Simulación sistema de evasión.

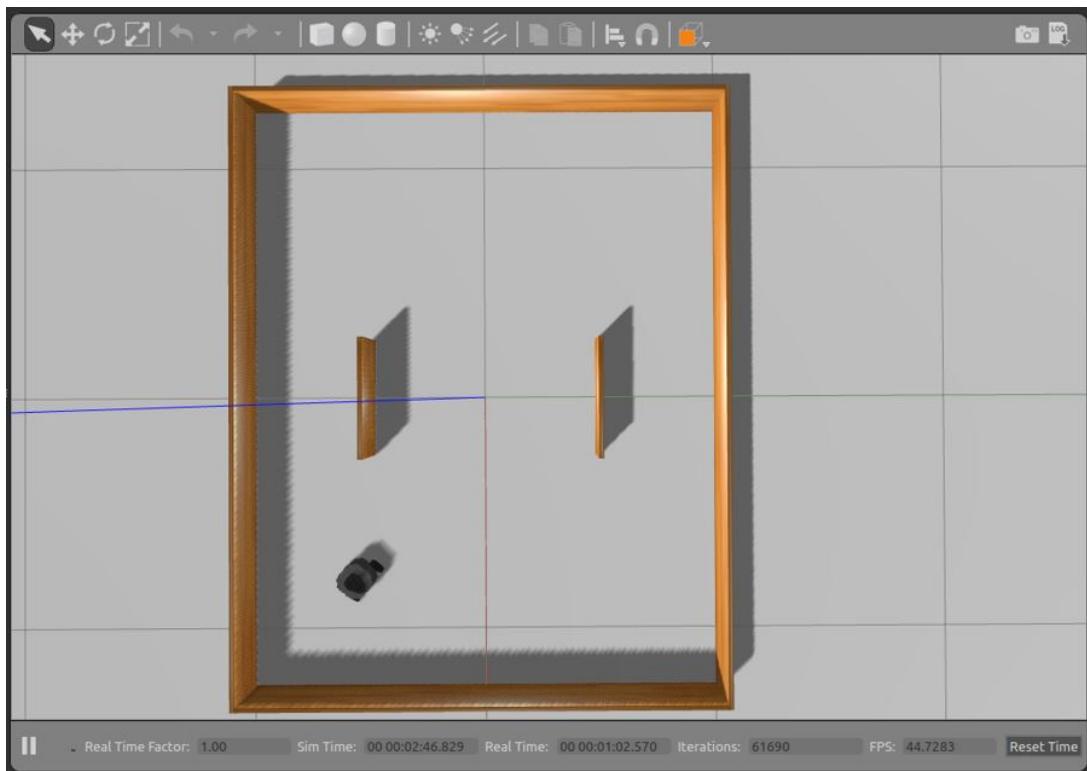


Figura 4.35: Simulación sistema de evasión.

Se puede observar que el resultado obtenido es el esperado, el algoritmo cumple con su objetivo, pero aun se puede mejorar, ya que al no considerar un amplio rango de detección por parte del sensor LIDAR puede que el algoritmo no elija la mejor forma de evadir un obstáculo.

Después se hizo una comparación entre los valores de salida del controlador implementado en MATLAB y el implementado en Python. A continuación se muestra, en la tabla 3.7, los resultados.

Tabla 4.1: Comparación de los valores del controlador de evasión.

| Norte | Este | Oeste | Lineal |        | Angular  |         | Formación |        |
|-------|------|-------|--------|--------|----------|---------|-----------|--------|
|       |      |       | MATLAB | Python | MATLAB   | Python  | MATLAB    | Python |
| 1     | 1    | 1     | 0.11   | 0.11   | -3.3e-17 | 2.1e-16 | 0.817     | 0.8974 |
| 0.3   | 0.3  | 0.3   | 0.0226 | 0.0233 | 1.95     | 1.92    | 4.18      | 3.4    |
| 0.3   | 0.7  | 0.3   | 0.0303 | 0.0308 | -1.65    | -1.633  | 1.09      | 1.2183 |
| 1     | 0.3  | 0.3   | 0.11   | 0.11   | -3.3e-17 | 2.1e-16 | 4.18      | 3.4    |

Se puede observar que existen algunas variaciones entre los resultados obtenidos por MATLAB y los del algoritmo implementado en Python, esto puede deberse a la precisión utilizada, pero en general los valores son muy parecidos.

## 4.4. Sistema de Formación

A continuación, se muestran los resultados de las simulaciones en Gazebo, en las cuales se realizan ambas formaciones propuestas. Diríjase a la sección de apéndices para conocer más acerca del código implementado.

### Formación en delta

Para empezar, se muestra la posición inicial de los robots dentro del escenario de pruebas.

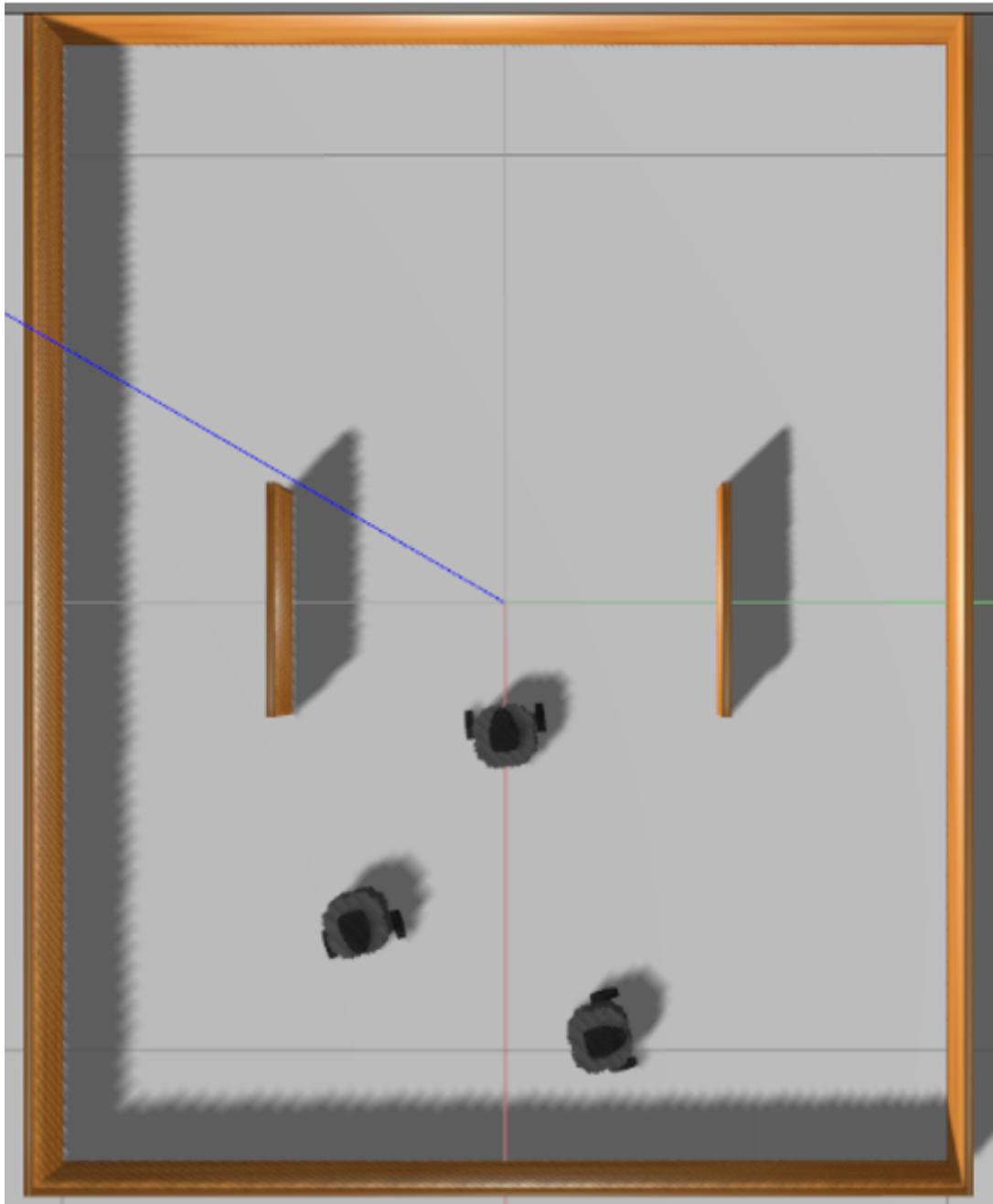


Figura 4.36: Posición inicial de los robots en el escenario para la formación en delta.

Después, se presentan los puntos calculados a los cuales los robots seguidores deben dirigirse para realizar la formación.

```
/home/henry/catkin_ws/src/turtle_contro... x      henry@henry-System-Product-Name:~ x + 
henry@henry-System-Product-Name:~$ rosrun turtle_control_system Formacion.py
Seleccion del robot Lider
Trayectoria: x = 0, y = 0.2
Distancia Robot 1: 0.332200240819, Lider
Distancia Robot 2: 0.953016264289, Seguidor
Distancia Robot 3: 0.899680498844, Seguidor
[INFO] [1589741217.117055, 629.492000]: Trayectoria robot 1: [x = 0.266000, y =
0.001000]
[INFO] [1589741217.117379, 629.493000]: Trayectoria robot 2: [x = 0.444369, y =
0.146204]
[INFO] [1589741217.117559, 629.493000]: Trayectoria robot 3: [x = 0.480935, y =
-0.080870]
```

Figura 4.37: Puntos calculados de los seguidores para la formación delta.

Luego, se muestra el desplazamiento de los robots a su posición inicial.

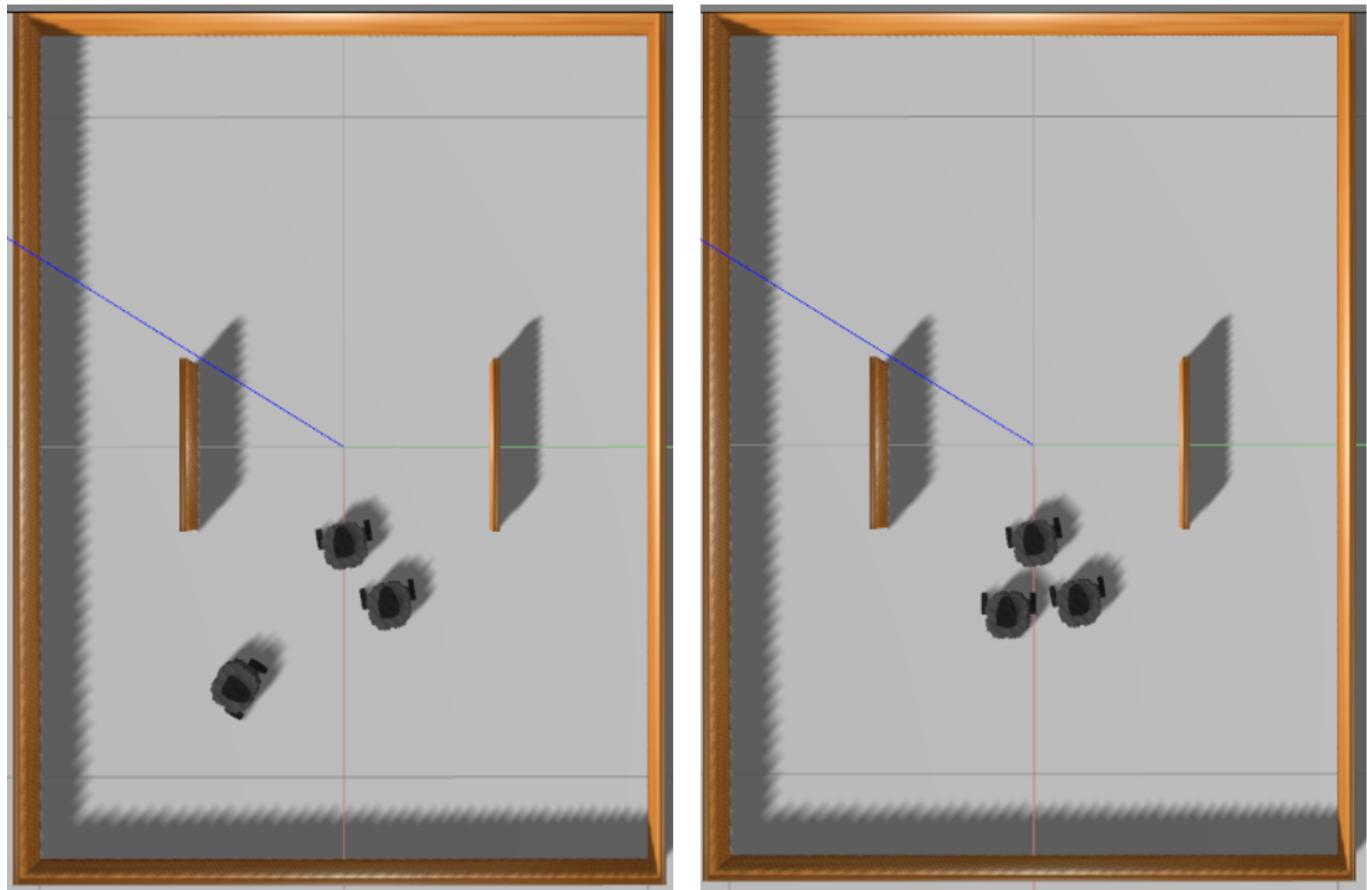


Figura 4.38: Realización de la formación delta.

Por último, se muestra el desplazamiento de los robots en formación.

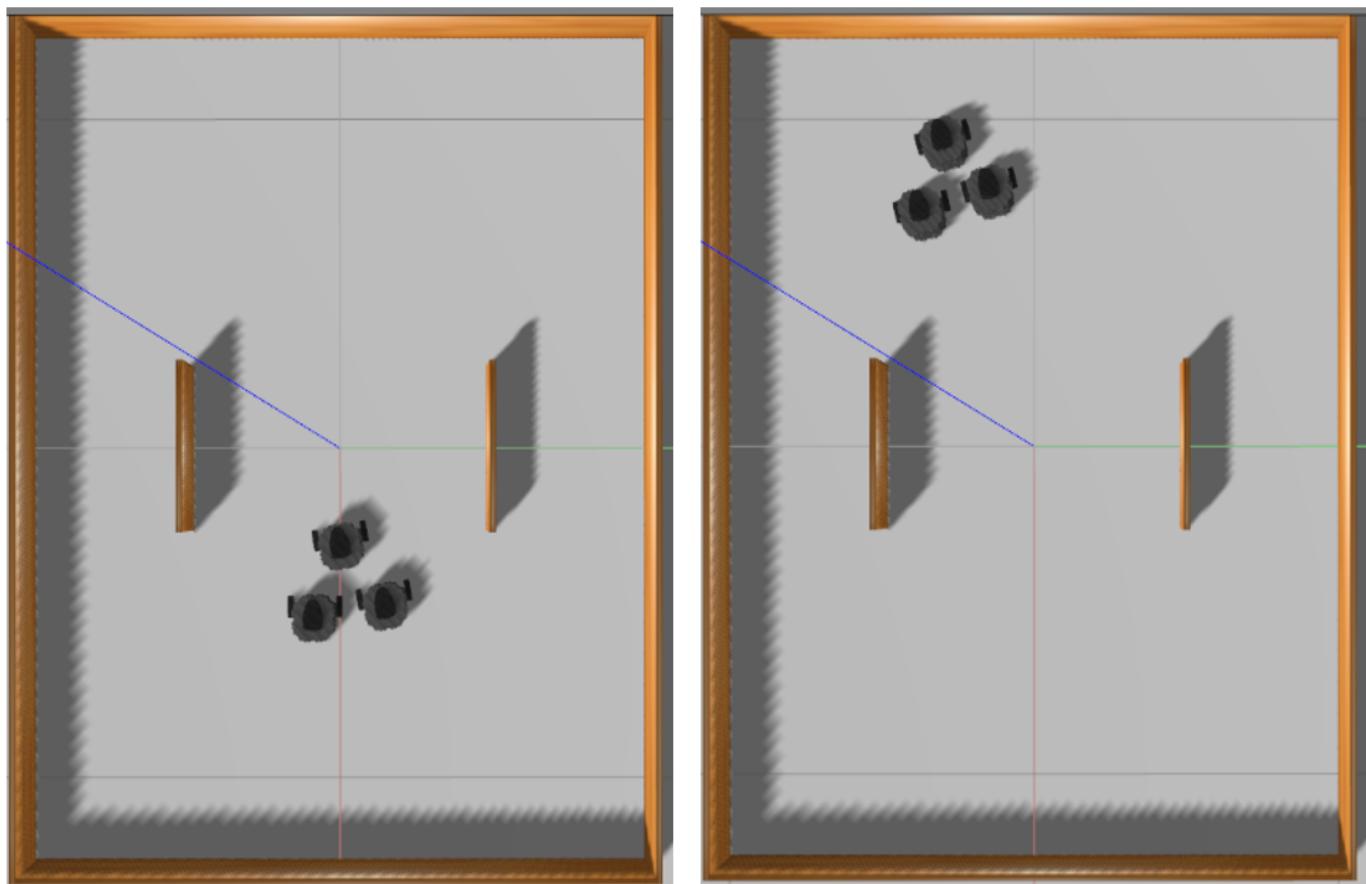


Figura 4.39: Recorrido en formación delta.

## Formación en convoy

Para empezar, se muestra la posición inicial de los robots dentro del escenario de pruebas.

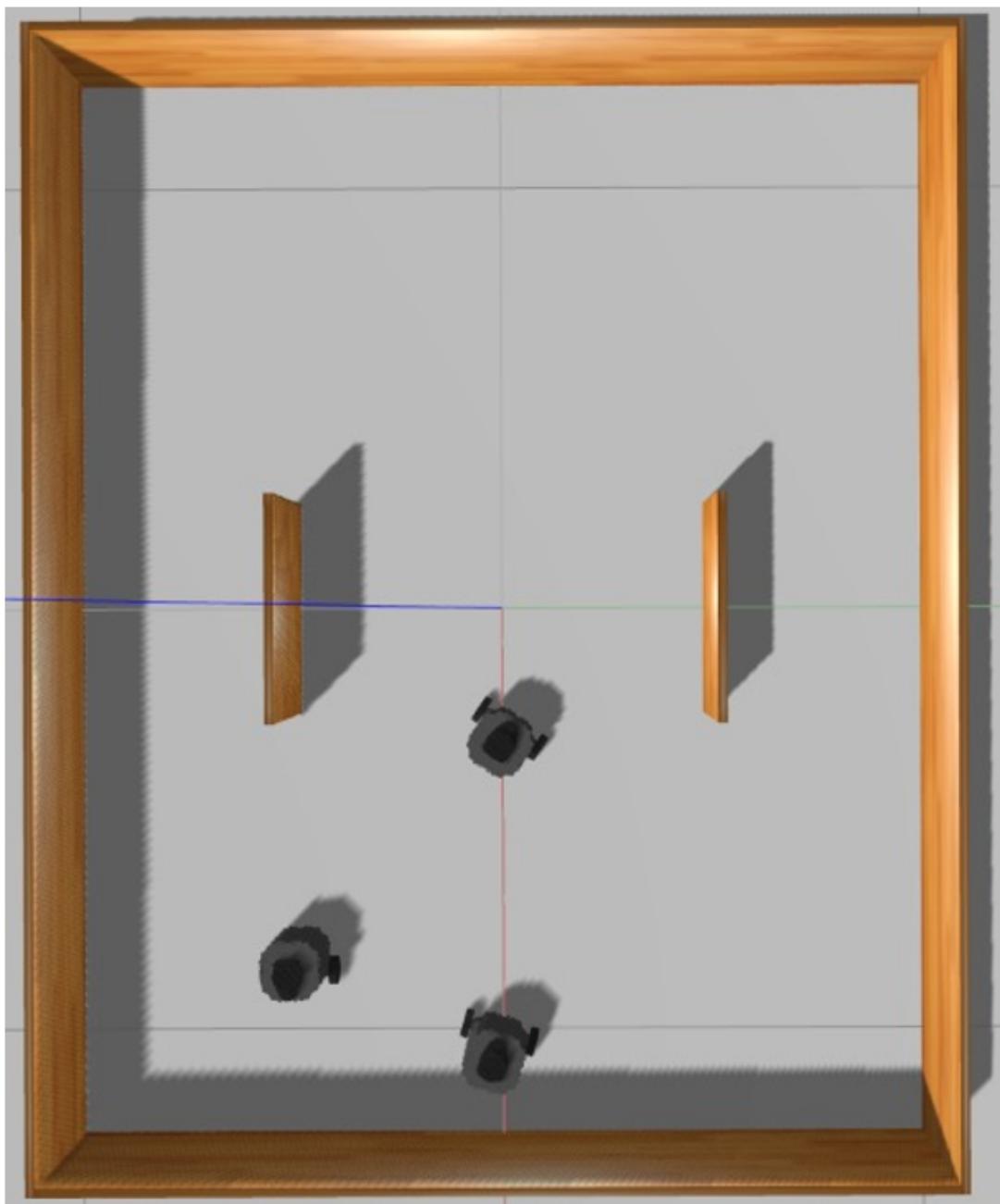


Figura 4.40: Posición inicial de los robots en el escenario para la formación en convoy.

## Pruebas y Resultados

Después, se presentan los puntos calculados a los cuales los robots seguidores deben dirigirse para realizar la formación.

```
/home/enrique/catkin_ws/src/turtle_cont... x enrique@enrique-Inspiron-7472: ~ x + -  
enrique@enrique-Inspiron-7472:~$ rosrun turtle_control_system Formacion.py  
Selección del robot Líder  
Trayectoria: x = -1, y = 0  
Distancia Robot 1: 1.28512606386, Líder  
Distancia Robot 2: 1.999, Seguidor  
Distancia Robot 3: 1.91803258575, Seguidor  
[INFO] [1591661266.356754, 175.638000]: Trayectoria robot 1: [x = 0.285000, y = 0.018000]  
[INFO] [1591661266.363820, 175.641000]: Trayectoria robot 2: [x = 0.476533, y = -0.109338]  
[INFO] [1591661266.366442, 175.642000]: Trayectoria robot 3: [x = 0.668067, y = -0.236676]
```

Figura 4.41: Puntos calculados de los seguidores para la formación convoy.

Luego, se muestra el desplazamiento de los robots a su posición inicial.

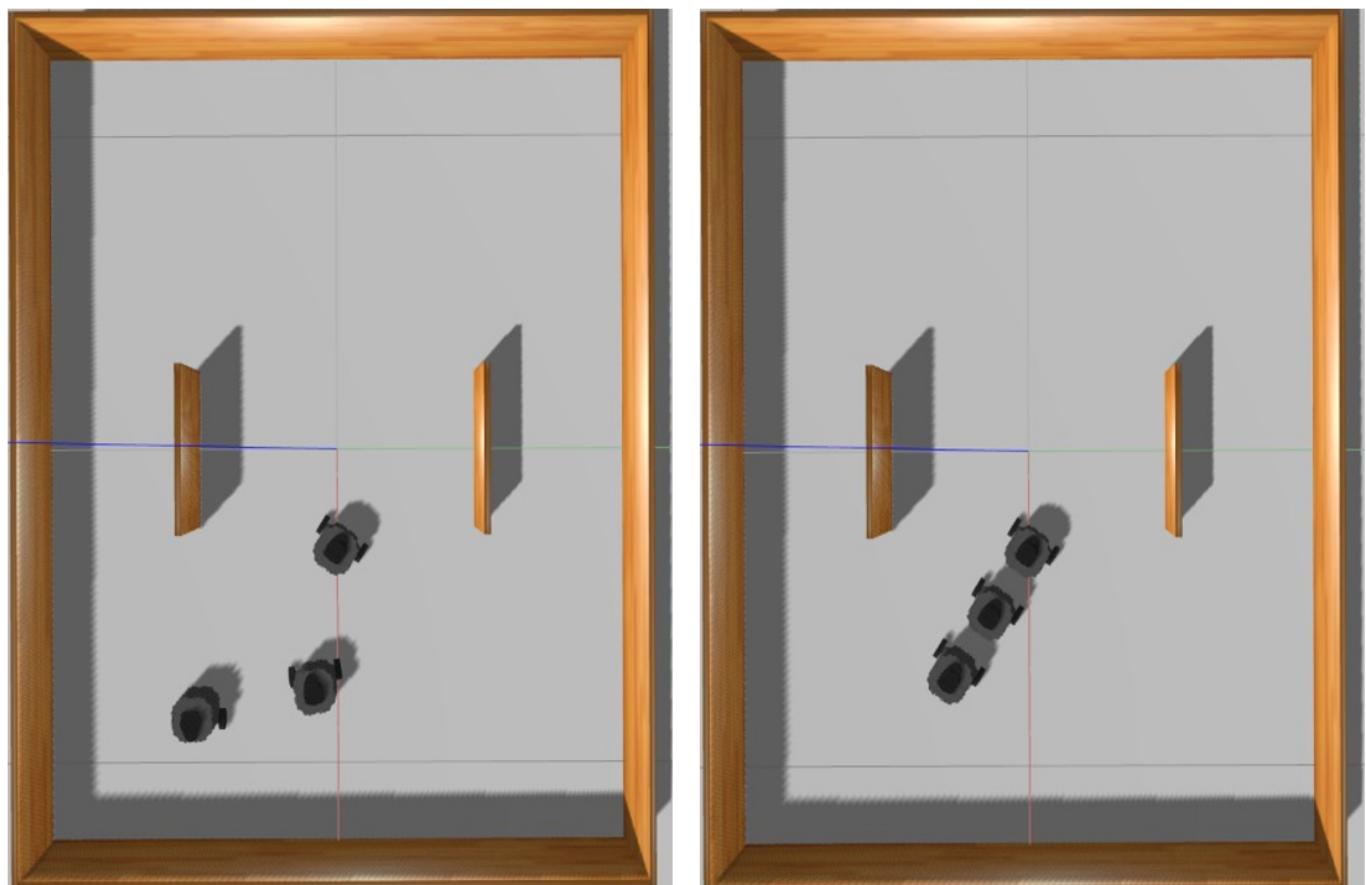


Figura 4.42: Realización de la formación convoy.

Por último, se muestra el desplazamiento de los robots en formación.

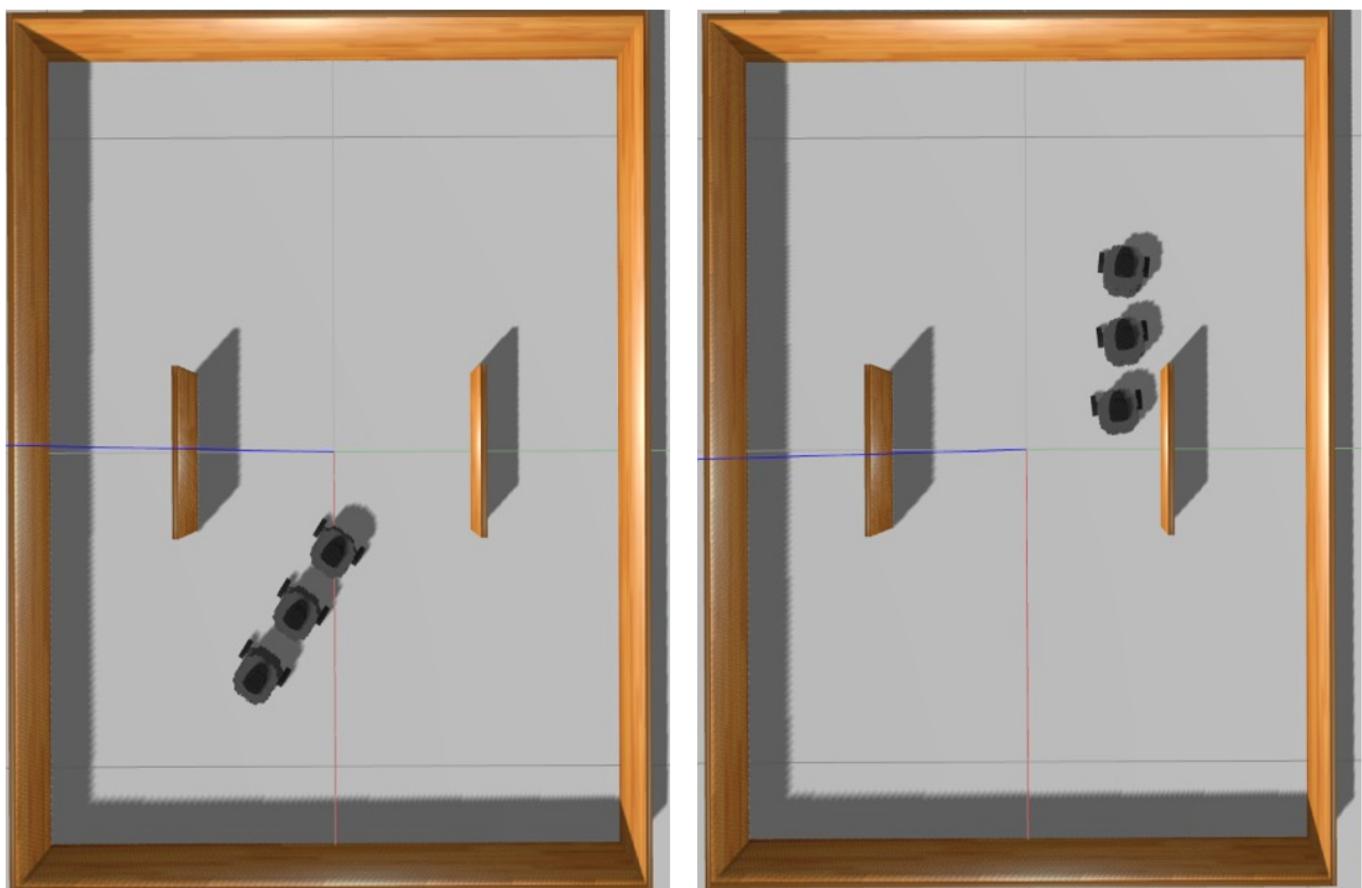


Figura 4.43: Recorrido en formación convoy.

De los resultados obtenidos, se puede concluir que el sistema tiene un buen comportamiento con el controlador proporcional. Cabe destacar que, para este resultado, las ganancias  $k$  tuvieron que calcularse de manera iterativa, ya que, con ganancias grandes, el sistema tiende a desestabilizarse.

### Cambio de formación

Además de estos resultados, se probó como funciona el cambio de formación durante el trayecto de los robots. Debido a que aun no se tiene integrado el sistema, es necesario simular la señal que indica cuando se requiere modificar la formación debido a que hay una reducción en el espacio. Así, se optó por declarar que ante valores de posición del robot líder en el rango  $-0,5 \leq x \leq 0,5$ , la formación se realizaría en convoy, mientras que, para valores fuera de ese rango, la formación se realizaría en delta.

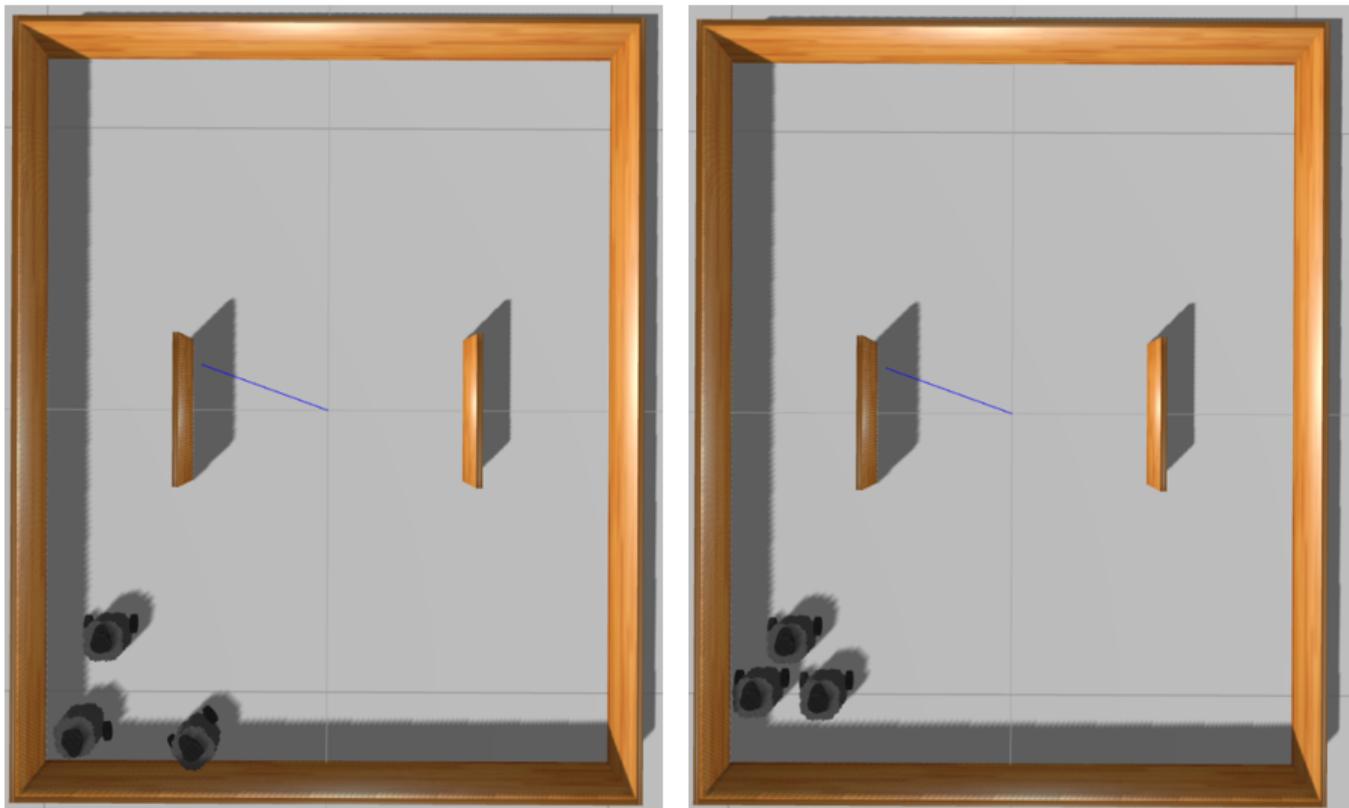


Figura 4.44: Cambio de formación de delta a convoy.

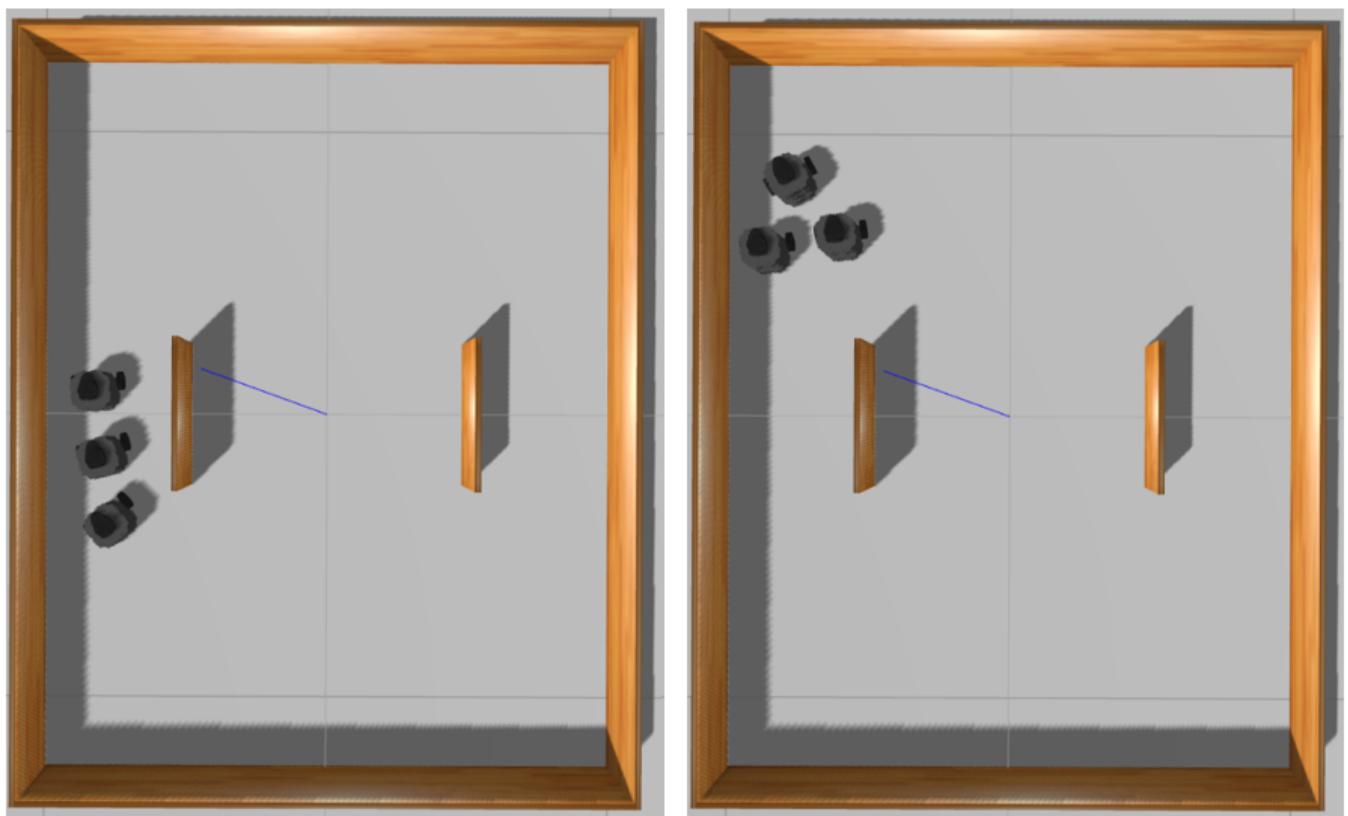


Figura 4.45: Cambio de formación de convoy a delta.

Los resultados muestran que el cambio de formación de los robots durante el recorrido del robot líder se realiza de forma correcta. Si bien, la modificación de formación se realiza durante un movimiento continuo, las pruebas finales se realizarán cuando todos los sistemas estén integrados para observar su desempeño y determinar si es necesario optar por la alternativa en la cual se detiene el grupo y se realiza el cambio para posteriormente continuar con el desplazamiento.

# Conclusiones

Los resultados obtenidos muestran un funcionamiento adecuado en los sistemas diseñados, que, en conjunto, conformarían el proyecto. Sin embargo, aun hay detalles a considerar durante la integración e implementación.

Comenzando por la localización de los robots, el progreso obtenido por el momento está limitado a las herramientas de simulación. Sin embargo, aunque es imposible visualizar el comportamiento real, con los resultados obtenidos se puede concluir que la lógica planteada funciona.

Siguiendo con la navegación de un robot dentro del espacio de trabajo, se logró el seguimiento de una trayectoria, teniendo mejores resultados ante la utilización de un controlador clásico debido a que el controlador difuso tiene una reacción más lenta en comparación con este, es decir, el tiempo que le toma al robot desplazarse de un punto a otro, mostró ser mayor (10s contra 15s).

A pesar de que el controlador difuso no requiera del modelo matemático para ser diseñado, si se necesita tener un amplio conocimiento sobre el funcionamiento del sistema. Además, el tiempo de procesamiento y los recursos utilizados fueron mayores en comparación con el controlador clásico.

Una vez que se definió la utilización de un controlador clásico, es posible agregar la acción de control integral al controlador proporcional diseñado buscando reducir el error en estado estacionario que se pudo observar en las gráficas obtenidas durante la validación.

Si bien, el controlador propuesto para la evasión de obstáculos trabaja adecuadamente, aun se debe buscar la forma de incorporarlo al seguimiento de trayectorias de lo contrario, se necesitará seleccionar un método diferente.

Por último, para la formación del conjunto de robots, se utiliza un controlador clásico como consecuencia de las observaciones realizadas al controlador difuso empleado para el seguimiento de trayectorias.

En general, se puede concluir que se satisfacen los requerimientos del diseño propuestos y que se cumplen los objetivos planteados para esta etapa del proyecto. No obstante, aun es necesario comprobar el desempeño de todo el sistema en conjunto, debido a que actualmente solo se cuenta con los sistemas funcionando de manera individual.

## Referencias

- [1] L. Solaque, M. Molina y E. Rodríguez, “Seguimiento de trayectorias con un robot móvil de configuración diferencial”, Ing. USBMed., vol. 5, pp. 26-34, 2014.
- [2] C. Soria, R. Carelli, R. Kelly y J. Ibarra, “Control de robots Cooperativos por medio de visión artificial” en Congreso de la Sociedad Chilena de Control Automático, Santiago, Chile, 2004, pp. 1-6 [En línea]. Disponible en: [http://ebanov.inaut.unsj.edu.ar/publicaciones/Ca1661\\_04.pdf](http://ebanov.inaut.unsj.edu.ar/publicaciones/Ca1661_04.pdf)
- [3] J. Chen, D. Sun, J. Yang y H. Chen, “Leader-follower formation control of multiple non-holonomic mobile robots incorporating a receding-horizon scheme”, pp. 727-746, 2010
- [4] C.E. Bugarin y A. Y. Aguilar, “Control visual para la formación de robots móviles tipo uniciclo bajo el esquema líder-seguidor”, Ingeniería investigación y tecnología, vol. 15, pp. 593-602, 2013.
- [5] J. D. Gallegos, R. Castro y M. Velasco, “Control de formación líder-seguidor robusto de un conjunto de robots móviles diferenciales”, pp. 1-6, 2018.
- [6] V. Barrientos, J. García y R. Silva, “Robots Móviles: Evolución y Estado del Arte”, Polibits, vol. 17, no. 35, pp. 12-17, 2007.
- [7] A. Bazoula, M. Djouadi y H. Maaref, “Formation control of multi-robots via fuzzy logic technique”, International Journal of Computers, Communications & Control (IJCCC), pp. 179-184, 2008.
- [8] M. Sánchez, D. Toro y L. Zarazua, “Coordinación de dos robots móviles terrestres bajo un esquema líder seguidor”, Trabajo Terminal, Mecatrónica, IPN, Ciudad de México, CDMX, 2019.
- [9] A. Domínguez y P. Medina, “Control de sistema robótico multi-agente de navegación terrestre”, Trabajo Terminal, Mecatrónica, IPN, Ciudad de México, CDMX, 2019.

- [10] D. Báez, J. Hernández y I. Martínez, “Sistema Multi-Robot descentralizado para la navegación vial autónoma en un escenario controlado”, Trabajo Terminal, Mecatrónica, IPN, Ciudad de México, CDMX, 2019.
- [11] C. A. Villar, “Control inteligente aplicado a robótica colaborativa”, Tesis de maestría, Sistemas digitales, IPN, Tijuana, B.C., 2016.
- [12] Q. Han, S. Sun y H. Lang, “Leader-follower formation control of multi-robots based on bearing-only observations”, pp. 120-129, 2019.
- [13] “Conceptos básicos de la robótica”, apuntes del curso Autómatas Industriales, Academia de Mecatrónica, IPN, 2020.
- [14] J. Lentin, *Robot Operating System (ROS) for Absolute Beginners*. 1ra edición. India: Apress, 2018.
- [15] Y. Pyo, H. Cho, R. Jung y T. Lim, *ROS Robot Programming: From the basic concept to practical programming and robot application*. 1ra edición. República de Corea: ROBOTIS Co., Ltd., 2017.
- [16] A. Barrientos, L. Peñin, C. Balaguer y R. Aracil, *Fundamentos de Robótica*. 2da edición. España: Mc Graw Hill, 2007.
- [17] I. Bronshtein, K. Semendiaev, *Manual de matemáticas para ingenieros y estudiantes*. 2da Edición. Moscú: Editorial Mir, 1973.
- [18] V. Mazzone, *Controladores PID*. Universidad Nacional de Quilmes, 2002.
- [19] A. Pérez, *Resumen de las principales características de controladores PID*. Ministerio de Educación, Universidad Nacional de San Juan.
- [20] C. González, *Lógica difusa*. España: Universidad de Castilla La Mancha, 2011.
- [21] F. Ortiz, “Modelado y control PD-Difuso en tiempo real para el sistema barra-esfera”, Tesis de maestría, Control Automático, IPN, Ciudad de México, CDMX, 2004.
- [22] T. Ross, *Fuzzy logic with engineering applications*. 2da edición. Estados Unidos: John Wiley & Son, Ltd, 2004.
- [23] Z. Kovacic y S. Bogdan, *Fuzzy controller design*. Estados Unidos: Taylor & Francis Group, 2006.

## Referencias

---

- [24] A. Freddi, M. Salmon, *Design Principles and Methodologies*. 1ra edición. Bologna, Italia: Springer, 2018.
- [25] U. Peñuelas, “Metodología para diseño mecatrónico”, Tesis de maestría, Facultad de ingeniería, UNAM, Ciudad de México, CDMX, 2007.
- [26] Open Robotics (2020), Robots E-manual TurtleBot3 [En línea]. Disponible en: <http://emanual.robotis.com/docs/en/platform/turtlebot3/specifications/>
- [27] Open Robotics (2020), Robots E-manual TurtleBot3 [En línea]. Disponible en: [http://emanual.robotis.com/docs/en/platform/turtlebot3/appendix\\_ids\\_01/](http://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_ids_01/)
- [28] Open Robotics (2020), Robots E-manual TurtleBot3 [En línea]. Disponible en: [http://emanual.robotis.com/docs/en/platform/turtlebot3/appendix\\_opencr1\\_0/](http://emanual.robotis.com/docs/en/platform/turtlebot3/appendix_opencr1_0/)
- [29] Raspberrypi, Raspberry Pi 3 Model B+ [En línea]. Disponible es: <https://www.raspberrypi.org/products/raspberry-pi-3-model-b-plus/>
- [30] “Control de Movimientos”, IA y Robótica. Grupo MINA. Universidad de la República. Disponible en: <https://www.fing.edu.uy/inco/cursos/robotica/>
- [31] T. Fraichard, ”Trajectory planning in a dynamic workspace: a 'Statetime space' approach”, Advanced Robotics, INRIA Rhône Alpes, Francia, 1999.
- [32] Open Robotics (2020), Robots E-manual TurtleBot3 [En línea]. Disponible en: <http://emanual.robotis.com/docs/en/dxl/x/xl430-w250/>

# **Apéndices**

## A. Códigos

### A.1. Sistema de Localización

```
1 #!/usr/bin/env python
2 from tf.transformations import euler_from_quaternion, quaternion_from_euler
3 import time
4 import rospy
5 from sensor_msgs.msg import LaserScan
6
7 distancias1 = []
8 nuevo1 = []
9 dist1 = []
10 distancias2 = []
11 nuevo2 = []
12 dist2 = []
13 distancias3 = []
14 nuevo3 = []
15 dist3 = []
16
17 class Robot:
18     #Variables para Posicion
19     x=0
20     y=0
21
22     #Variables Orientacion
23     #Quaternion
24     qx=0
25     qy=0
26     qz=0
27     qw=0
28     #Euler
29     roll=0
30     pitch=0
31     yaw=0
32
33     #Variables distancia Laser
34     ranges = []
35
36 class TurtleBotSystem:
37     def __init__(self):
```

```
38    rospy.init_node('turtle_control', anonymous=False)
39
40    self.tb1=Robot()
41    self.tb2=Robot()
42    self.tb3=Robot()
43
44    self.rate=rospy.Rate(10)
45    #Nodos de los robots
46    #Robot 1
47    self.scan_subscriber_tb1=rospy.Subscriber("/Robot1/scan", LaserScan, self.
48    scan_callback_tb1)
49    #Robot 2
50    self.scan_subscriber_tb2=rospy.Subscriber("/Robot2/scan", LaserScan, self.
51    scan_callback_tb2)
52    #Robot 3
53    self.scan_subscriber_tb3=rospy.Subscriber("/Robot3/scan", LaserScan, self.
54    scan_callback_tb3)
55
56    def scan_callback_tb1(self,msg):
57        distancias1=msg.ranges
58        a1=sorted(enumerate(distancias1), key=lambda x: x[1], reverse=True)
59        index1, value1 = a1[359]
60        print("Robot 1 -> El valor mas chico: {}, Angulo:{}" .format(value1, index1))
61        nuevo1.extend(distancias1[index1: ])
62        nuevo1.extend(distancias1[ :index1-1])
63        Norte1=nuevo1[0]
64        Oeste1=nuevo1[90]
65        Sur1=nuevo1[180]
66        Este1=nuevo1[270]
67        print("Robot 1 N: {}, O: {}, S: {}, E: {}" .format(Norte1,Oeste1,Sur1,Este1))
68
69    def scan_callback_tb2(self,msg):
70        distancias2=msg.ranges
71        a2=sorted(enumerate(distancias2), key=lambda x: x[1], reverse=True)
72        index2, value2 = a2[359]
73        print("Robot 2 -> El valor mas chico: {}, Angulo:{}" .format(value2, index2))
74        nuevo2.extend(distancias2[index2: ])
75        nuevo2.extend(distancias2[ :index2-1])
76        Norte2=nuevo2[0]
77        Oeste2=nuevo2[90]
78        Sur2=nuevo2[180]
```

```
76 Este2=nuevo2[270]
77 print("Robot 2 N: {}, O: {}, S: {}, E: {}".format(Norte2,Oeste2,Sur2,Este2))
78
79 def scan_callback_tb3(self,msg):
80     distancias3=msg.ranges
81     a3=sorted(enumerate(distancias3), key=lambda x: x[1], reverse=True)
82     index3, value3 = a3[359]
83     print("Robot 3 -> El valor mas chico: {}, Angulo:{} ".format(value3, index3))
84     nuevo3.extend(distancias3[index3:])
85     nuevo3.extend(distancias3[:index3-1])
86     Norte3=nuevo3[0]
87     Oeste3=nuevo3[90]
88     Sur3=nuevo3[180]
89     Este3=nuevo3[270]
90     print("Robot 3 N: {}, O: {}, S: {}, E: {}".format(Norte3,Oeste3,Sur3,Este3))
91     print"====="
92
93 def start_system(self):
94
95     while not rospy.is_shutdown():
96         self.rate.sleep()
97
98 if __name__=='__main__':
99     try:
100         t1=TurtleBotSystem()
101         t1.start_system()
102     except rospy.ROSInterruptException:
103         pass
```

## A.2. Sistema de Navegación

### Seguimiento de Trayectorias: Control P

```

1 #!/usr/bin/env python
2
3 from tf.transformations import euler_from_quaternion, quaternion_from_euler
4 import move_p2p
5 import lineal_transformations
6 from numpy import array
7 from math import pi
8 import time
9
10 #Importamos rospy
11 import rospy
12
13 #Llamamos los nodos que vamos a usar
14 from geometry_msgs.msg import Twist
15 from nav_msgs.msg import Odometry
16 =====
17 #Variables globales
18 Kv=5 #Ganancia Velocidad Linear
19 Ka=2 #Ganancia Velocidad Angular
20 tol=0.01 #Tolerancia permitida
21 =====
22 #Trayectoria temporal para pruebas
23 tx=[0.8, 0.8, -0.8, -0.8, 0]
24 ty=[0, 0.8, 0.8, -0.8, 0]
25 =====
26 class Robot:
27     #Variables para Posicion
28     x=0
29     y=0
30
31     #Variables Orientacion
32     #Quaternion
33     qx=0
34     qy=0
35     qz=0
36     qw=0
37     #Euler
38     roll=0

```

## Apéndices

```
39     pitch=0
40     yaw=0
41
42 #Variables distancia Laser
43 ranges=[]
44
45 #Variables proximo punto de trayectoria
46 gx=0
47 gy=0
48
49 #Rol del Robot
50 role='None'
51
52 #Variables de apoyo para uso de coordenadas absolutas
53 H=[]
54 angle=0
55 curyaw=0
56 curx=0
57 cury=0
58 curgx=0
59 curgy=0
60 =====
61 class TurtleBotSystem:
62     def __init__(self):
63         #Nodo maestro
64         rospy.init_node('turtle_control', anonymous=False)
65
66         self.tb1=Robot()
67
68         #Nodos de lectura
69         #Robot 1
70         self.pose_subscriber_tb1=rospy.Subscriber('Robot1/odom', Odometry, self.
71             pose_callback_tb1)
72
73         #Nodos de escritura
74         #Robot 1
75         self.velocity_publisher=rospy.Publisher('Robot1/cmd_vel', Twist, queue_size=10)
76
77         self.rate=rospy.Rate(10) #10Hz
78
79     def pose_callback_tb1(self, msg):
```

```
79     self.tb1.qx=round(msg.pose.pose.orientation.x,2)
80     self.tb1.qy=round(msg.pose.pose.orientation.y,2)
81     self.tb1.qz=round(msg.pose.pose.orientation.z,2)
82     self.tb1.qw=round(msg.pose.pose.orientation.w,2)
83
84     (self.tb1.roll ,self.tb1.pitch ,self.tb1.yaw)=euler_from_quaternion([self.tb1.qx ,
85                           self.tb1.qy ,self.tb1.qz ,self.tb1.qw])
86
87     self.tb1.x=round(msg.pose.pose.position.x,3)
88     self.tb1.y=round(msg.pose.pose.position.y,3)
89 #=====
90 #Sección donde inicia todo el movimiento del sistema
91 def start_system(self):
92     global tx ,ty
93
94     self.sys_start=True
95     self.state='Rotar '
96     self.wait=True
97
98     vel_msg=Twist()
99     rospy.on_shutdown(self.shutdown)
100
101    while not rospy.is_shutdown():
102        #Asignamos un timer para que el sistema pueda comenzar a leer los datos de
103        #los sensores
104        if self.wait==True:
105            time.sleep(1)
106            #Obtenemos el angulo en el que el robot se encuentra inicialmente
107            self.tb1.angle=self.tb1.yaw
108            self.transf_matrix_H()
109            self.wait=False
110
111            #Actualizamos la posición y rotación de las coordenadas absolutas
112            self.cur_position()
113            self.cur_yaw()
114
115            #Iniciamos el sistema
116            if self.sys_start==True:
117                for i in range(len(tx)):
118                    #Tomamos los puntos de la trayectoria
119                    self.tb1.gx=tx[i]
```

## Apéndices

```
118         self.tb1.gy=ty[i]
119
120         #Actualizamos las coordenadas objetivo al sistema coordenado absoluto
121         self.cur_goal()
122         self.state='Rotar'
123
124         #Imprimimos las coordenadas objetivo
125         rospy.loginfo('Coordenadas objetivo: [x : %f , y : %f]',tx[i],ty[i])
126
127         #Entrando a la maquina de estados
128         while self.state != 'Fin':
129             self.turtle_state()
130
131         #Obteniendo el nuevo marco de referencia cuando termina de llegar al
132         #punto
133         if self.state == 'Fin':
134             self.tb1.angle=self.tb1.yaw
135             self.tb1.curyaw=0
136             self.transf_matrix_H()
137             self.show_info()
138             #Apagamos el sistema
139             self.sys_start=False
140             self.rate.sleep()
141 #=====
142     #En esta sección entramos a los diferentes casos para el control del robot líder,
143     #el desarrollo para los robots seguidores tendría que ser diferente
144     def turtle_state(self):
145         if self.state=='Rotar':
146             vel_msg=Twist()
147
148             while move_p2p.angle_error(self.tb1.curgx,self.tb1.curgy,self.tb1.curx,self.
149             tb1.cury,self.tb1.curyaw) >= tol:
150                 if move_p2p.angle_error(self.tb1.curgx,self.tb1.curgy,self.tb1.curx,self.
151                 tb1.cury,self.tb1.curyaw) > 2:
152                     if self.tb1.angle + pi > pi:
153                         self.tb1.angle=self.tb1.angle - pi
154                     else:
155                         self.tb1.angle=self.tb1.angle + pi
156                         self.transf_matrix_H()
157                         self.cur_goal()
158                         self.update_pose()
```

```

155     vel_msg.linear.x=0
156     vel_msg.linear.y=0
157     vel_msg.linear.z=0
158
159     vel_msg.angular.x=0
160     vel_msg.angular.y=0
161     vel_msg.angular.z=move_p2p.angular_vel(self.tb1.curgx,self.tb1.curgy,self.
162                                               tb1.curx,self.tb1.cury,Ka,self.tb1.curyaw)
163     self.velocity_publisher.publish(vel_msg)
164
165     self.state='Avanzar'
166     self.rate.sleep()
167 elif self.state=='Avanzar':
168     vel_msg=Twist()
169
170     while move_p2p.euclidean_distance(self.tb1.curgx,self.tb1.curgy,self.tb1.curx,
171                                         self.tb1.cury) >= tol:
172         if move_p2p.angle_error(self.tb1.curgx,self.tb1.curgy,self.tb1.curx,self.
173                               tb1.cury,self.tb1.curyaw) > 2:
174             if self.tb1.angle + pi > pi:
175                 self.tb1.angle=self.tb1.angle - pi
176             else:
177                 self.tb1.angle=self.tb1.angle + pi
178             self.transf_matrix_H()
179             self.cur_goal()
180             self.update_pose()
181             vel_msg.linear.x=move_p2p.linear_vel(self.tb1.curgx,self.tb1.curgy,self.tb1
182 .curx,self.tb1.cury,Kv)
183             vel_msg.linear.y=0
184             vel_msg.linear.z=0
185
186             vel_msg.angular.x=0
187             vel_msg.angular.y=0
188             vel_msg.angular.z=move_p2p.angular_vel(self.tb1.curgx,self.tb1.curgy,self.
189                                               tb1.curx,self.tb1.cury,Ka,self.tb1.curyaw)
190             self.velocity_publisher.publish(vel_msg)
191
192             self.velocity_publisher.publish(Twist())
193
194             self.state='Fin'
195             self.rate.sleep()

```

```

191     else:
192         rospy.loginfo("Error, intentando de nuevo... ")
193         self.tb1.state='Inicio'
194 #####
195 #Sección de transformación de coordenadas
196 def transf_matrix_H(self):
197     #Se obtiene la matriz de transformación H actual
198     self.tb1.H=lineal_transformations.cur_H(self.tb1.angle, self.tb1.x, self.tb1.y)
199
200 def cur_position(self):
201     #Transformamos las coordenadas actuales del robot del marco de referencia fijo
202     #al del robot
203     (self.tb1.curx, self.tb1.cury)=lineal_transformations.cur_point(self.tb1.H, self.
204     tb1.x, self.tb1.y)
205
206 def cur_yaw(self):
207     #Transformamos la rotación del robot del marco de referencia fijo al actual
208     dif_tb1=self.tb1.yaw-self.tb1.angle
209     if dif_tb1 > pi:
210         self.tb1.curyaw=self.tb1.yaw-self.tb1.angle-2*pi
211     elif dif_tb1 < -pi:
212         self.tb1.curyaw=self.tb1.yaw-self.tb1.angle+2*pi
213     else:
214         self.tb1.curyaw=self.tb1.yaw-self.tb1.angle
215
216 def cur_goal(self):
217     #Transformamos las coordenadas objetivo del marco de referencia fijo al actual
218     (self.tb1.curgx, self.tb1.curgy)=lineal_transformations.cur_point(self.tb1.H,
219     self.tb1.gx, self.tb1 gy)
220
221 #####
222 def update_pose(self):
223     self.cur_position()
224     self.cur_yaw()
225 #####
226 def show_info(self):
227     print("====")
228     rospy.loginfo(self.tb1.H)
229     rospy.loginfo("Posicion del robot:\n[x = %f y = %f]", self.tb1.x, self.tb1.y)
230     rospy.loginfo("Posicion actual del robot:\n[x = %f y = %f]", self.tb1.curx, self.
231     tb1.cury)
232     rospy.loginfo("Orientacion del robot:\n[yaw= %f]", self.tb1.yaw)

```

```
228     rospy.loginfo("Orientacion actual del robot:\n[yaw= %f]",self.tb1.cur yaw)
229
230     def shutdown(self):
231         rospy.loginfo("Stopping TurtleBot...")
232         self.velocity_publisher.publish(Twist())
233         rospy.sleep(1)
234 #=====
235 if __name__=='__main__':
236     try:
237         t1=TurtleBotSystem()
238         t1.start_system()
239     except rospy.ROSInterruptException:
240         pass
```

## Seguimiento de Trayectorias: Control Difuso

```
1 #!/usr/bin/env python
2 from tf.transformations import euler_from_quaternion, quaternion_from_euler
3 from numpy import array
4 from math import pi, pow, atan2, sqrt
5 import rospy
6 import numpy as np
7 import skfuzzy as fuzz
8 from skfuzzy import control as ctrl
9 import matplotlib.pyplot as plt
10 import time
11 from geometry_msgs.msg import Twist
12 from nav_msgs.msg import Odometry
13
14 class Position:
15     x=[]
16     y=[]
17
18 class Robot:
19     #Variables para Posicion
20     pose=Position()
21     pose.x=0
22     pose.y=0
23
24     #Variables Orientacion
25     qx=0
26     qy=0
27     qz=0
28     qw=0
29
30     roll=0
31     pitch=0
32     yaw=0
33
34     goal=Position()
35     goal.x=0
36     goal.y=0
37
38     #Variables actuales del sistema de referencia
39     H=[]
40     angle=0
```

```
41 curyaw=0
42 curpose=Position()
43 curpose.x=0
44 curpose.y=0
45 curgoal=Position()
46 curgoal.x=0
47 curgoal.y=0
48 eangulo=0
49 dist=0
50 edist=0
51
52 def controlador(eangulo , dist):
53     eang = np.arange(-3.15,3.15,0.01)
54     pos = np.arange(0,3.2,0.01)
55     angular = np.arange(-2.84,2.84,0.01)
56     lineal = np.arange(0,0.22,0.01)
57
58     eang_gdneg = fuzz.trimf(eang, [-3.15,-3.15,-1])
59     eang_chneg = fuzz.trimf(eang, [-2,-1,-0.1])
60     eang_cero = fuzz.trimf(eang, [-0.1,0,0.1])
61     eang_chpos = fuzz.trimf(eang, [0.1,1,2])
62     eang_gdpos = fuzz.trimf(eang, [1,3.15,3.15])
63
64     pos_cero = fuzz.trimf(pos, [0,0,0.05])
65     pos_ch = fuzz.trimf(pos, [0.05,0.5,1])
66     pos_gd = fuzz.trapmf(pos, [0.5,1,3.2,3.2])
67
68     angular_gder = fuzz.trimf(angular, [-2.84,-2.84,-1])
69     angular_chder = fuzz.trimf(angular, [-2,-1,-0.01])
70     angular_cero = fuzz.trimf(angular, [-0.01,0,0.01])
71     angular_chizq = fuzz.trimf(angular, [0.01,1,2])
72     angular_gizq = fuzz.trimf(angular, [1,2.84,2.84])
73
74     lineal_cero = fuzz.trimf(lineal, [0,0,0.005])
75     lineal_lento = fuzz.trimf(lineal, [0.005,0.11,0.15])
76     lineal_rapido = fuzz.trimf(lineal, [0.13,0.22,0.22])
77
78     x = int(rob.eangulo*100+315)
79     y = int(rob.dist*100+1)
80
81 R1=np.fmin(eang_gdneg[x],pos_cero[y])
```

## Apéndices

---

```
82 R2=np.fmin(eang_gdneg[x],pos_ch[y])
83 R3=np.fmin(eang_gdneg[x],pos_gd[y])
84
85 R4=np.fmin(eang_chneg[x],pos_cero[y])
86 R5=np.fmin(eang_chneg[x],pos_ch[y])
87 R6=np.fmin(eang_chneg[x],pos_gd[y])
88
89 R7=np.fmin(eang_cero[x],pos_cero[y])
90 R8=np.fmin(eang_cero[x],pos_ch[y])
91 R9=np.fmin(eang_cero[x],pos_gd[y])
92
93 R10=np.fmin(eang_chpos[x],pos_cero[y])
94 R11=np.fmin(eang_chpos[x],pos_ch[y])
95 R12=np.fmin(eang_chpos[x],pos_gd[y])
96
97 R13=np.fmin(eang_gdpos[x],pos_cero[y])
98 R14=np.fmin(eang_gdpos[x],pos_ch[y])
99 R15=np.fmin(eang_gdpos[x],pos_gd[y])
100
101 #MAX salida V. angular
102 CM1=np.fmax(R1,R4)
103 CM2=np.fmax(R7,R8)
104 CM3=np.fmax(R9,R10)
105 CM4=np.fmax(CM1,CM2)
106 CM5=np.fmax(CM3,R13)
107 CMF=np.fmax(CM4,CM5)
108 GDF=np.fmax(R2,R3)
109 CDF=np.fmax(R5,R6)
110 CIF=np.fmax(R11,R12)
111 GIF=np.fmax(R14,R15)
112
113 #Cutlines
114 cut1=np.fmin(angular_cero,CMF)
115 cut2=np.fmin(angular_gder,GDF)
116 cut3=np.fmin(angular_chder,CDF)
117 cut4=np.fmin(angular_chizq,CIF)
118 cut5=np.fmin(angular_gizq,GIF)
119
120 #MAX cutlines
121 macut1=np.fmax(cut1,cut2)
122 macut2=np.fmax(cut3,cut4)
```

```
123     macut3=np.fmax(macut1,macut2)
124     macut4=np.fmax(cut5,macut3)
125
126     angular_value = fuzz.defuzz(angular,macut4, 'centroid')
127
128 #MAX salida V. lineal
129 ZM1=np.fmax(R1,R2)
130 ZM2=np.fmax(R3,R4)
131 ZM3=np.fmax(R5,R6)
132 ZM4=np.fmax(R7,R10)
133 ZM5=np.fmax(R11,R12)
134 ZM6=np.fmax(R13,R14)
135 ZM7=np.fmax(ZM1,ZM2)
136 ZM8=np.fmax(ZM3,ZM4)
137 ZM9=np.fmax(ZM5,ZM6)
138 ZM10=np.fmax(ZM7,ZM8)
139 ZM11=np.fmax(ZM9,R15)
140 ZMF=np.fmax(ZM10,ZM11)
141
142 #Cutlines
143 cut1l=np.fmin(lineal_cero,ZMF)
144 cut2l=np.fmin(lineal_lento,R8)
145 cut3l=np.fmin(lineal_rapido,R9)
146
147 #MAX cutlines
148 macut1l=np.fmax(cut1l,cut2l)
149 macut2l=np.fmax(cut3l,macut1l)
150
151 lineal_value = fuzz.defuzz(lineal,macut2l, 'centroid')
152 return angular_value,lineal_value
153
154 def pose_callback(msg):
155     rob.pose.x=round(msg.pose.pose.position.x,4)
156     rob.pose.y=round(msg.pose.pose.position.y,4)
157     rob.qx=round(msg.pose.pose.orientation.x,2)
158     rob.qy=round(msg.pose.pose.orientation.y,2)
159     rob.qz=round(msg.pose.pose.orientation.z,2)
160     rob.qw=round(msg.pose.pose.orientation.w,2)
161     (rob.roll,rob.pitch,rob.yaw)=euler_from_quaternion([rob.qx,rob.qy,rob.qz,rob.qw])
162 ])
```

## Apéndices

```
163 #Puntos
164     rob.goal.x = 0.8
165     rob.goal.y = -0.8
166 #Angulo de giro
167     rob.angle = atan2(rob.goal.y-rob.pose.y,rob.goal.x-rob.pose.x)
168 #Error angulo
169     rob.eangulo = round(rob.angle - rob.yaw,2)
170 #Distancia euclidiana
171     rob.dist = round(sqrt(pow((rob.goal.x-rob.pose.x),2)+pow((rob.goal.y-rob.pose.y),2)),2)
172
173     a,l=controlador(rob.eangulo,rob.dist)
174     move.linear.x = l
175     move.angular.z = a
176     pub.publish(move)
177
178 initTime=time.time()
179 move = Twist()
180 rospy.init_node('odometria_data')
181 pub = rospy.Publisher('/Robot1/cmd_vel', Twist, queue_size=10)
182 pose_subscriber=rospy.Subscriber('/Robot1/odom', Odometry, pose_callback)
183 rob=Robot()
184 rospy.spin()
```

## Evasión de Obstáculos

```

1 #!/usr/bin/env python
2 import rospy
3 import numpy as np
4 import skfuzzy as fuzz
5 from skfuzzy import control as ctrl
6 import matplotlib.pyplot as plt
7 from geometry_msgs.msg import Twist
8 from sensor_msgs.msg import LaserScan
9
10 def callback(msg):
11
12     if (msg.ranges[0] < 0.5) and (msg.ranges[7] > 0.5) and (msg.ranges[353] > 0.5) :
13         entradanorte = msg.ranges[0]
14     elif (msg.ranges[0] > 0.5) and (msg.ranges[7] < 0.5) and (msg.ranges[353] > 0.5)
15         :
16         entradanorte = msg.ranges[7]
17     elif (msg.ranges[0] > 0.5) and (msg.ranges[7] > 0.5) and (msg.ranges[353] < 0.5)
18         :
19         entradanorte = msg.ranges[353]
20     else :
21         entradanorte = msg.ranges[0]
22
23     entradaoeste = msg.ranges[90]
24     entradaeste = msg.ranges[270]
25
26     #Conjuntos difusos
27     #ENTRADAS
28     norte=ctrl.Antecedent(np.arange(0, 2.5, 0.01), 'norte')
29     este=ctrl.Antecedent(np.arange(0, 2.5, 0.01), 'este')
30     oeste=ctrl.Antecedent(np.arange(0, 2.5, 0.01), 'oeste')
31     #SALIDAS
32     lineal=ctrl.Consequent(np.arange(0,0.22,0.01), 'lineal')
33     angular=ctrl.Consequent(np.arange(-2.84,2.84,0.01), 'angular')
34     formacion=ctrl.Consequent(np.arange(0,5,1), 'formacion')
35     #Funciones de membresia
36     #ENTRADAS
37     norte[ 'cerca' ] = fuzz.trapmf(norte.universe, [0,0,0.4,0.6])
38     norte[ 'medio' ] = fuzz.trimf(norte.universe, [0.6,1,1.45])
39     norte[ 'lejos' ] = fuzz.trapmf(norte.universe, [1,1.45,2.5,2.5])

```

```

39 este[ 'cerca' ] = fuzz.trapmf(este.universe , [0,0,0.4,0.6])
40 este[ 'medio' ] = fuzz.trimf(este.universe , [0.6,1,1.45])
41 este[ 'lejos' ] = fuzz.trapmf(este.universe , [1,1.45,2.5,2.5])
42
43 oeste[ 'cerca' ] = fuzz.trapmf(oeste.universe , [0,0,0.4,0.6])
44 oeste[ 'medio' ] = fuzz.trimf(oeste.universe , [0.6,1,1.45])
45 oeste[ 'lejos' ] = fuzz.trapmf(oeste.universe , [1,1.45,2.5,2.5])
46
47 #SALIDAS
48 lineal[ 'lento' ] = fuzz.trimf(lineal.universe , [0,0,0.07])
49 lineal[ 'medio' ] = fuzz.trimf(lineal.universe , [0.07,0.11,0.15])
50 lineal[ 'rapido' ] = fuzz.trimf(lineal.universe , [0.13,0.22,0.22])
51
52 angular[ 'derecha' ] = fuzz.trimf(angular.universe , [-2.84,-2.84,-0.1])
53 angular[ 'cero' ] = fuzz.trimf(angular.universe , [-0.1,0,0.1])
54 angular[ 'izquierda' ] = fuzz.trimf(angular.universe , [0.1,2.84,2.84])
55
56 formacion[ 'constante' ] = fuzz.trimf(formacion.universe , [0,0,2.5])
57 formacion[ 'cambio' ] = fuzz.trimf(formacion.universe , [2.5,5,5])
58
59 #Reglas difusas Salida 1
60 rule0 = ctrl.Rule(antecedent=((norte[ 'cerca' ] & este[ 'cerca' ] & oeste[ 'cerca' ]) | (norte[ 'cerca' ] & este[ 'cerca' ] & oeste[ 'medio' ]) | (norte[ 'cerca' ] & este[ 'cerca' ] & oeste[ 'lejos' ]) | (norte[ 'cerca' ] & este[ 'medio' ] & oeste[ 'cerca' ]) | (norte[ 'cerca' ] & este[ 'medio' ] & oeste[ 'medio' ]) | (norte[ 'cerca' ] & este[ 'medio' ] & oeste[ 'lejos' ]) | (norte[ 'cerca' ] & este[ 'lejos' ] & oeste[ 'cerca' ]) | (norte[ 'cerca' ] & este[ 'lejos' ] & oeste[ 'medio' ]) | (norte[ 'cerca' ] & este[ 'lejos' ] & oeste[ 'lejos' ])), consequent=lineal[ 'lento' ], label='rule lento')
61 rule1 = ctrl.Rule(antecedent=((norte[ 'medio' ] & este[ 'cerca' ] & oeste[ 'cerca' ]) | (norte[ 'medio' ] & este[ 'cerca' ] & oeste[ 'medio' ]) | (norte[ 'medio' ] & este[ 'cerca' ] & oeste[ 'lejos' ]) | (norte[ 'medio' ] & este[ 'medio' ] & oeste[ 'cerca' ]) | (norte[ 'medio' ] & este[ 'medio' ] & oeste[ 'medio' ]) | (norte[ 'medio' ] & este[ 'medio' ] & oeste[ 'lejos' ]) | (norte[ 'medio' ] & este[ 'lejos' ] & oeste[ 'cerca' ]) | (norte[ 'medio' ] & este[ 'lejos' ] & oeste[ 'medio' ]) | (norte[ 'medio' ] & este[ 'lejos' ] & oeste[ 'lejos' ])), consequent=lineal[ 'medio' ], label='rule medio')
62 rule2 = ctrl.Rule(antecedent=((norte[ 'lejos' ] & este[ 'cerca' ] & oeste[ 'cerca' ]) | (norte[ 'lejos' ] & este[ 'cerca' ] & oeste[ 'medio' ]) | (norte[ 'lejos' ] & este[ 'cerca' ] & oeste[ 'lejos' ]) | (norte[ 'lejos' ] & este[ 'medio' ] & oeste[ 'cerca' ]) | (norte[ 'lejos' ] & este[ 'medio' ] & oeste[ 'medio' ]) | (norte[ 'lejos' ] & este[ 'medio' ] & oeste[ 'lejos' ]) | (norte[ 'lejos' ] & este[ 'lejos' ] & oeste[ 'cerca' ]) | (norte[ 'lejos' ] & este[ 'lejos' ] & oeste[ 'medio' ]) | (norte[ 'lejos' ] & este[ 'lejos' ] & oeste[ 'lejos' ])), consequent=lineal[ 'rapido' ], label='rule rapido')

```

```

.....oeste[ 'lejos '))), consequent=lineal[ 'rapido '], label='rule rapido ')
63 #Reglas difusas Salida 2
64 rule3 = ctrl.Rule(antecedent=((norte[ 'cerca '] & este[ 'cerca '] & oeste[ 'cerca ']) |
65   (norte[ 'cerca '] & este[ 'cerca '] & oeste[ 'medio ']) | (norte[ 'cerca '] & este[ 'cerca '] & oeste[ 'lejos ']) | (norte[ 'cerca '] & este[ 'medio '] & oeste[ 'medio ']) |
66   (norte[ 'cerca '] & este[ 'medio '] & oeste[ 'lejos ']) | (norte[ 'cerca '] & este[ 'lejos '] & oeste[ 'lejos '])), consequent=angular[ 'izquierda '], label='rule izquierda ')
67 rule4 = ctrl.Rule(antecedent=((norte[ 'cerca '] & este[ 'medio '] & oeste[ 'cerca ']) |
68   (norte[ 'cerca '] & este[ 'lejos '] & oeste[ 'cerca ']) | (norte[ 'cerca '] & este[ 'lejos '] & oeste[ 'medio '])), consequent=angular[ 'derecha '], label='rule derecha ')
69 rule5 = ctrl.Rule(antecedent=((norte[ 'medio '] & este[ 'cerca '] & oeste[ 'cerca ']) |
70   (norte[ 'medio '] & este[ 'cerca '] & oeste[ 'medio ']) | (norte[ 'medio '] & este[ 'cerca '] & oeste[ 'lejos ']) | (norte[ 'medio '] & este[ 'medio '] & oeste[ 'cerca ']) | (norte[ 'medio '] & este[ 'medio '] & oeste[ 'medio ']) | (norte[ 'medio '] & este[ 'lejos '] & oeste[ 'cerca ']) | (norte[ 'medio '] & este[ 'lejos '] & oeste[ 'medio ']) | (norte[ 'medio '] & este[ 'lejos '] & oeste[ 'lejos ']) | (norte[ 'lejos '] & este[ 'cerca '] & oeste[ 'cerca ']) | (norte[ 'lejos '] & este[ 'cerca '] & oeste[ 'medio ']) | (norte[ 'lejos '] & este[ 'cerca '] & oeste[ 'lejos ']) | (norte[ 'lejos '] & este[ 'medio '] & oeste[ 'cerca ']) | (norte[ 'lejos '] & este[ 'medio '] & oeste[ 'medio ']) | (norte[ 'lejos '] & este[ 'medio '] & oeste[ 'lejos ']) | (norte[ 'lejos '] & este[ 'lejos '] & oeste[ 'cerca ']) | (norte[ 'lejos '] & este[ 'lejos '] & oeste[ 'medio ']) | (norte[ 'lejos '] & este[ 'lejos '] & oeste[ 'lejos '])), consequent=angular[ 'cero '], label='rule cero ')
71 #Reglas difusas Formacion
72 rule7 = ctrl.Rule(antecedent=(este[ 'cerca '] & oeste[ 'cerca ']), consequent=
73   formacion[ 'cambio '], label='rule cambio ')
74 rule8 = ctrl.Rule(antecedent=((este[ 'cerca '] & oeste[ 'medio ']) | (este[ 'cerca '] &
75   oeste[ 'lejos ']) | (este[ 'medio '] & oeste[ 'cerca ']) | (este[ 'medio '] & oeste[ 'medio ']) |
76   (este[ 'medio '] & oeste[ 'lejos ']) | (este[ 'lejos '] & oeste[ 'cerca ']) | (este[ 'lejos '] & oeste[ 'medio ']) |
77   (este[ 'lejos '] & oeste[ 'lejos '])), consequent=
78   formacion[ 'constante '], label='rule constante ')
#Salida 1
system1 = ctrl.ControlSystem(rules=[rule0 , rule1 , rule2 ])
simulacion1 = ctrl.ControlSystemSimulation(system1)
#Valores entradas
simulacion1.input[ 'norte ']= entradanorte
simulacion1.input[ 'este ']= entradaeste
simulacion1.input[ 'oeste ']= entradaoeste
simulacion1.compute()

```

```
79
80 #Salida 2
81 system2 = ctrl.ControlSystem(rules=[rule3, rule4, rule5])
82 simulacion2 = ctrl.ControlSystemSimulation(system2)
83 #Valores entradas
84 simulacion2.input['norte']= entradanorte
85 simulacion2.input['este'] = entradaeste
86 simulacion2.input['oeste'] = entradaoeste
87 simulacion2.compute()
88
89 #Salida 3
90 system3 = ctrl.ControlSystem(rules=[rule7, rule8])
91 simulacion3 = ctrl.ControlSystemSimulation(system3)
92 #Valores entradas
93 simulacion3.input['este'] = entradaeste
94 simulacion3.input['oeste'] = entradaoeste
95 simulacion3.compute()
96
97 move.linear.x = float(simulacion1.output['lineal'])
98 move.angular.z = float(simulacion2.output['angular'])
99 pub.publish(move)
100
101 move = Twist()
102 rospy.init_node('obstacle_avoidance_node')
103 pub = rospy.Publisher("/Robot1/cmd_vel", Twist, queue_size=10)
104 sub = rospy.Subscriber("/Robot1/scan", LaserScan, callback)
105 rospy.spin()
```

### A.3. Sistema de Formación

```
1 #!/usr/bin/env python
2
3 from tf.transformations import euler_from_quaternion, quaternion_from_euler
4 import move_p2p
5 import lineal_transformations
6 import formation as form
7 from numpy import array
8 from math import pi, cos, sin
9 import time
10
11 #Importamos rospy
12 import rospy
13
14 #Llamamos los nodos que vamos a usar
15 from geometry_msgs.msg import Twist
16 from nav_msgs.msg import Odometry
17 from sensor_msgs.msg import LaserScan
18 =====
19 #Variables globales
20 Kv=5 #Ganancia Velocidad Linear
21 Ka=2 #Ganancia Velocidad Angular
22 tol=0.01 #Tolerancia permitida
23 =====
24 #Trayectoria temporal para pruebas
25 tx=0
26 ty=0
27 =====
28 class Robot:
29     #Variables para Posicion
30     x=0
31     y=0
32
33     #Variables Orientacion
34     #Quaternion
35     qx=0
36     qy=0
37     qz=0
38     qw=0
39     #Euler
```

## Apéndices

```
40     roll=0
41     pitch=0
42     yaw=0
43
44 #Variables distancia Laser
45 ranges=[]
46
47 #Variables actuales del sistema de referencia
48 H=[]
49 angle=0
50 curyaw=0
51 curx=0
52 cury=0
53 curgx=0
54 curgy=0
55
56 role='None'
57 =====
58 class TurtleBot:
59     def __init__(self):
60         #Nodo maestro
61         rospy.init_node('turtle_control', anonymous=False)
62
63         self.tb1=Robot()
64         self.tb2=Robot()
65         self.tb3=Robot()
66
67     #Nodos de lectura
68     #Robot 1
69         self.pose_subscriber_tb1=rospy.Subscriber('/Robot1/odom', Odometry, self.
70             pose_callback_tb1)
71         self.detection_subscriber_tb1=rospy.Subscriber('/Robot1/scan', LaserScan, self.
72             scan_callback_tb1)
73     #Robot 2
74         self.pose_subscriber_tb2=rospy.Subscriber('/Robot2/odom', Odometry, self.
75             pose_callback_tb2)
76         self.detection_subscriber_tb2=rospy.Subscriber('/Robot2/scan', LaserScan, self.
77             scan_callback_tb2)
78     #Robot 3
79         self.pose_subscriber_tb3=rospy.Subscriber('/Robot3/odom', Odometry, self.
80             pose_callback_tb3)
```

```
76     self.detection_subscriber_tb3=rospy.Subscriber('/Robot3/scan',LaserScan,self.  
77         scan_callback_tb3)  
78  
79     #Nodos de escritura  
80     #Robot 1  
81     self.velocity_publisher_tb1=rospy.Publisher('/Robot1/cmd_vel',Twist,queue_size  
82         =10)  
83     #Robot 2  
84     self.velocity_publisher_tb2=rospy.Publisher('/Robot2/cmd_vel',Twist,queue_size  
85         =10)  
86     #Robot 3  
87     self.velocity_publisher_tb3=rospy.Publisher('/Robot3/cmd_vel',Twist,queue_size  
88         =10)  
89  
90     self.rate=rospy.Rate(10) #10Hz  
91  
92     def pose_callback_tb1(self,msg):  
93         self.tb1.qx=round(msg.pose.pose.orientation.x,2)  
94         self.tb1.qy=round(msg.pose.pose.orientation.y,2)  
95         self.tb1.qz=round(msg.pose.pose.orientation.z,2)  
96         self.tb1.qw=round(msg.pose.pose.orientation.w,2)  
97  
98         (self.tb1.roll,self.tb1.pitch,self.tb1.yaw)=euler_from_quaternion([self.tb1.qx,  
99             self.tb1.qy,self.tb1.qz,self.tb1.qw])  
100  
101    def scan_callback_tb1(self,msg):  
102        self.tb1.ranges=msg.ranges  
103  
104    def pose_callback_tb2(self,msg):  
105        self.tb2.qx=round(msg.pose.pose.orientation.x,2)  
106        self.tb2.qy=round(msg.pose.pose.orientation.y,2)  
107        self.tb2.qz=round(msg.pose.pose.orientation.z,2)  
108        self.tb2.qw=round(msg.pose.pose.orientation.w,2)  
109  
110        (self.tb2.roll,self.tb2.pitch,self.tb2.yaw)=euler_from_quaternion([self.tb2.qx,  
111             self.tb2.qy,self.tb2.qz,self.tb2.qw])
```

## Apéndices

```
111
112     self.tb2.x=round(msg.pose.pose.position.x,3)
113     self.tb2.y=round(msg.pose.pose.position.y,3)
114
115 def scan_callback_tb2(self,msg):
116     self.tb2.ranges=msg.ranges
117
118 def pose_callback_tb3(self,msg):
119     self.tb3.qx=round(msg.pose.pose.orientation.x,2)
120     self.tb3.qy=round(msg.pose.pose.orientation.y,2)
121     self.tb3.qz=round(msg.pose.pose.orientation.z,2)
122     self.tb3.qw=round(msg.pose.pose.orientation.w,2)
123
124     (self.tb3.roll,self.tb3.pitch,self.tb3.yaw)=euler_from_quaternion([self.tb3.qx,
125                         self.tb3.qy,self.tb3.qz,self.tb3.qw])
126
127     self.tb3.x=round(msg.pose.pose.position.x,3)
128     self.tb3.y=round(msg.pose.pose.position.y,3)
129
130     def scan_callback_tb3(self,msg):
131         self.tb3.ranges=msg.ranges
132 #=====
133 #Sección donde inicia todo el movimiento del sistema
134 def start_robot(self):
135     self.sys_start=True
136     self.wait=True
137
138     vel_msg=Twist()
139
140     self.tb1.role='Lider'
141     self.tb2.role='Seguidor'
142     self.tb3.role='Seguidor'
143
144     self.state='Inicio'
145     self.form='Delta'
146
147     while not rospy.is_shutdown():
148
149         #Asignamos un timer para que el sistema pueda comenzar a leer los datos de
150         #los sensores
151         if self.wait==True:
```

```

150     time.sleep(1)
151     self.tb1.angle=self.tb1.yaw
152     self.tb2.angle=self.tb2.yaw
153     self.tb3.angle=self.tb3.yaw
154     self.transf_matrix_H()
155     self.wait=False
156
157     self.update_pose()
158
159     while self.sys_start==True:
160
161         self.turtle_state()
162
163         if self.state == 'Fin':
164             self.show_info()
165
166         sys_start=False
167
168         self.rate.sleep()
169 #=====
170 #En esta sección entramos a los diferentes casos para el control del robot líder,
171 #el desarrollo para los robots seguidores tendría que ser diferente
171 def turtle_state(self):
172     global tx ,ty ,Kv,Ka,tol
173     if self.state == 'Inicio':
174
175         self.tb1.gx=tx
176         self.tb1 gy=ty
177         self.tb2.gx=tx
178         self.tb2 gy=ty
179         self.tb3.gx=tx
180         self.tb3 gy=ty
181
182     #Asignamos quien es el líder dependiendo de quien este mas cerca del inicio
183     #de la trayectoria
183     list_tb =[move_p2p.euclidean_distance(tx ,ty ,self.tb1.x ,self.tb1.y) ,move_p2p.
184     euclidean_distance(tx ,ty ,self.tb2.x ,self.tb2.y) ,move_p2p.euclidean_distance(tx ,
185     ty ,self.tb3.x ,self.tb3.y)]
184     role_l=min(list_tb )
185
186     for count,line in enumerate(list_tb ):

```

```

187     if line == role_l:
188         if count == 0:
189             self.tb1.role='Lider'
190             self.tb2.role='Seguidor'
191             self.tb3.role='Seguidor'
192         elif count == 1:
193             self.tb1.role='Seguidor'
194             self.tb2.role='Lider'
195             self.tb3.role='Seguidor'
196         else:
197             self.tb1.role='Seguidor'
198             self.tb2.role='Seguidor'
199             self.tb3.role='Lider'
200     print 'Seleccion del robot Lider'
201     print 'Trayectoria: x = {}, y = {}'.format(tx ,ty)
202     print 'Distancia Robot 1: {}, {}'.format(move_p2p.euclidean_distance(tx ,ty ,
203                                         self.tb1.x, self.tb1.y),self.tb1.role)
204     print 'Distancia Robot 2: {}, {}'.format(move_p2p.euclidean_distance(tx ,ty ,
205                                         self.tb2.x, self.tb2.y),self.tb2.role)
206     print 'Distancia Robot 3: {}, {}'.format(move_p2p.euclidean_distance(tx ,ty ,
207                                         self.tb3.x, self.tb3.y),self.tb3.role)

208
209     self.tb1.gx, self.tb1.gy, self.tb2.gx, self.tb2.gy, self.tb3.gx, self.tb3.gy=form.
210     first_form(self.tb1.role, self.tb2.role, self.tb3.role, self.tb1.x, self.tb1.y, self.
211     tb1.yaw, self.tb2.x, self.tb2.y, self.tb2.yaw, self.tb3.x, self.tb3.y, self.tb3.yaw)
212     self.cur_goal()

213     rospy.loginfo('Trayectoria robot 1: [x = %, y = %]',self.tb1.gx, self.tb1.gy)
214     rospy.loginfo('Trayectoria robot 2: [x = %, y = %]',self.tb2.gx, self.tb2.gy)
215     rospy.loginfo('Trayectoria robot 3: [x = %, y = %]',self.tb3.gx, self.tb3.gy)

216     self.state='Mantener'
217     self.rate.sleep()
218     elif self.state == 'Mantener':
219         self.maintain_form()

```

```

220     self.state='Mantener'
221     self.rate.sleep()
222 else:
223     rospy.loginfo("Error, intentando de nuevo... ")
224     self.state='Fin'
225     self.rate.sleep()
226 #=====
227 def move_1by1(self):
228     global Ka,Kv
229     vel_msg=Twist()
230
231 #Rotamos el Robot 1
232 while move_p2p.angle_error(self.tb1.curgx,self.tb1.curgy,self.tb1.curx,self.tb1.
233 .cury,self.tb1.curyaw)>= tol:
234     if move_p2p.angle_error(self.tb1.curgx,self.tb1.curgy,self.tb1.curx,self.tb1.
235 .cury,self.tb1.curyaw)> 2:
236         if self.tb1.angle + pi > pi:
237             self.tb1.angle=self.tb1.angle - pi
238         else:
239             self.tb1.angle=self.tb1.angle + pi
240         self.transf_matrix_H()
241         self.cur_goal()
242         self.update_pose()
243         vel_msg.linear.x=0
244
245         vel_msg.angular.z=move_p2p.angular_vel(self.tb1.curgx,self.tb1.curgy,self.tb1.
246 .curx,self.tb1.cury,Ka,self.tb1.curyaw)
247         self.velocity_publisher_tb1.publish(vel_msg)
248         self.velocity_publisher_tb1.publish(Twist())
249 #Movemos el Robot 1
250 while move_p2p.euclidean_distance(self.tb1.curgx,self.tb1.curgy,self.tb1.curx,
251 self.tb1.cury)>= tol:
252     if move_p2p.angle_error(self.tb1.curgx,self.tb1.curgy,self.tb1.curx,self.tb1.
253 .cury,self.tb1.curyaw)> 2:
254         if self.tb1.angle + pi > pi:
255             self.tb1.angle=self.tb1.angle - pi
256         else:
257             self.tb1.angle=self.tb1.angle + pi
258         self.transf_matrix_H()
259         self.cur_goal()
260         self.update_pose()

```

```

256     vel_msg.linear.x=move_p2p.linear_vel(self.tb1.curgx ,self.tb1.curgy ,self.tb1.
curx ,self.tb1.cury ,Kv)
257
258     vel_msg.angular.z=move_p2p.angular_vel(self.tb1.curgx ,self.tb1.curgy ,self.tb1.
.curx ,self.tb1.cury ,Ka ,self.tb1.curyaw)
259     self.velocity_publisher_tb1.publish(vel_msg)
260     self.velocity_publisher_tb1.publish(Twist())
261
262 #Rotamos el Robot 2
263 while move_p2p.angle_error(self.tb2.curgx ,self.tb2.curgy ,self.tb2.curx ,self.tb2.
cury ,self.tb2.curyaw) >= tol:
264     if move_p2p.angle_error(self.tb2.curgx ,self.tb2.curgy ,self.tb2.curx ,self.tb2.
cury ,self.tb2.curyaw) > 2:
265         if self.tb2.angle + pi > pi:
266             self.tb2.angle=self.tb2.angle - pi
267         else:
268             self.tb2.angle=self.tb2.angle + pi
269         self.transf_matrix_H()
270         self.cur_goal()
271         self.update_pose()
272         vel_msg.linear.x=0
273
274     vel_msg.angular.z=move_p2p.angular_vel(self.tb2.curgx ,self.tb2.curgy ,self.tb2.
.curx ,self.tb2.cury ,Ka ,self.tb2.curyaw)
275     self.velocity_publisher_tb2.publish(vel_msg)
276     self.velocity_publisher_tb2.publish(Twist())
277 #Movemos el Robot 2
278 while move_p2p.euclidean_distance(self.tb2.curgx ,self.tb2.curgy ,self.tb2.curx ,
self.tb2.cury ) >= tol:
279     if move_p2p.angle_error(self.tb2.curgx ,self.tb2.curgy ,self.tb2.curx ,self.tb2.
cury ,self.tb2.curyaw) > 2:
280         if self.tb2.angle + pi > pi:
281             self.tb2.angle=self.tb2.angle - pi
282         else:
283             self.tb2.angle=self.tb2.angle + pi
284         self.transf_matrix_H()
285         self.cur_goal()
286         self.update_pose()
287         vel_msg.linear.x=move_p2p.linear_vel(self.tb2.curgx ,self.tb2.curgy ,self.tb2.
curx ,self.tb2.cury ,Kv)
288

```

```

289     vel_msg.angular.z=move_p2p.angular_vel(self.tb2.curgx ,self.tb2.curgy ,self.tb2
 .curx ,self.tb2.cury ,Ka ,self.tb2.curyaw )
    self.velocity_publisher_tb2.publish(vel_msg)
291     self.velocity_publisher_tb2.publish(Twist())
292
293 #Rotamos el Robot 3
294 while move_p2p.angle_error(self.tb3.curgx ,self.tb3.curgy ,self.tb3.curx ,self.tb3
 .cury ,self.tb3.curyaw ) >= tol :
295     if move_p2p.angle_error(self.tb3.curgx ,self.tb3.curgy ,self.tb3.curx ,self.tb3.
 cury ,self.tb3.curyaw ) > 2:
296         if self.tb3.angle + pi > pi:
297             self.tb3.angle=self.tb3.angle - pi
298         else:
299             self.tb3.angle=self.tb3.angle + pi
300         self.transf_matrix_H()
301         self.cur_goal()
302         self.update_pose()
303         vel_msg.linear.x=0
304
305     vel_msg.angular.z=move_p2p.angular_vel(self.tb3.curgx ,self.tb3.curgy ,self.tb3
 .curx ,self.tb3.cury ,Ka ,self.tb3.curyaw )
306     self.velocity_publisher_tb3.publish(vel_msg)
307     self.velocity_publisher_tb3.publish(Twist())
308 #Movemos el Robot 3
309 while move_p2p.euclidean_distance(self.tb3.curgx ,self.tb3.curgy ,self.tb3.curx ,
self.tb3.cury ) >= tol :
310     if move_p2p.angle_error(self.tb3.curgx ,self.tb3.curgy ,self.tb3.curx ,self.tb3.
cury ,self.tb3.curyaw ) > 2:
311         if self.tb3.angle + pi > pi:
312             self.tb3.angle=self.tb3.angle - pi
313         else:
314             self.tb3.angle=self.tb3.angle + pi
315         self.transf_matrix_H()
316         self.cur_goal()
317         self.update_pose()
318         vel_msg.linear.x=move_p2p.linear_vel(self.tb3.curgx ,self.tb3.curgy ,self.tb3.
curx ,self.tb3.cury ,Kv)
319
320     vel_msg.angular.z=move_p2p.angular_vel(self.tb3.curgx ,self.tb3.curgy ,self.tb3
 .curx ,self.tb3.cury ,Ka ,self.tb3.curyaw )
321     self.velocity_publisher_tb3.publish(vel_msg)

```

## Apéndices

```
322     self.velocity_publisher_tb3.publish(Twist())  
323  
324 def lider_angle(self):  
325     global Ka  
326     ang_form=0  
327     if self.tb1.role == 'Lider':  
328         self.tb2.angle=self.tb1.angle  
329         self.tb3.angle=self.tb1.angle  
330         self.tb2.H=self.tb1.H  
331         self.tb3.H=self.tb1.H  
332         self.update_pose()  
333  
334     ang_form=self.tb1.cur yaw  
335     elif self.tb2.role == 'Lider':  
336         self.tb1.angle=self.tb2.angle  
337         self.tb3.angle=self.tb2.angle  
338         self.tb1.H=self.tb2.H  
339         self.tb3.H=self.tb2.H  
340         self.update_pose()  
341  
342     ang_form=self.tb2.cur yaw  
343     elif self.tb3.role == 'Lider':  
344         self.tb1.angle=self.tb3.angle  
345         self.tb2.angle=self.tb3.angle  
346         self.tb1.H=self.tb3.H  
347         self.tb2.H=self.tb3.H  
348         self.update_pose()  
349  
350     ang_form=self.tb3.cur yaw  
351  
352 vel_msg=Twist()  
353  
354 #Rotamos el Robot 1  
355 while abs(ang_form - self.tb1.cur yaw) >= tol:  
356     self.update_pose()  
357     vel_msg.angular.z=move_p2p.angular_vel2(ang_form ,self.tb1.cur yaw,Ka)  
358     self.velocity_publisher_tb1.publish(vel_msg)  
359     self.velocity_publisher_tb1.publish(Twist())  
360 #Rotamos el Robot 2  
361 while abs(ang_form - self.tb2.cur yaw) >= tol:  
362     self.update_pose()
```

```

363     vel_msg.angular.z=move_p2p.angular_vel2(ang_form ,self.tb2.cur yaw,Ka)
364     self.velocity_publisher_tb2.publish(vel_msg)
365     self.velocity_publisher_tb2.publish(Twist())
366 #Rotamos el Robot 3
367 while abs(ang_form - self.tb3.cur yaw) >= tol:
368     self.update_pose()
369     vel_msg.angular.z=move_p2p.angular_vel2(ang_form ,self.tb3.cur yaw,Ka)
370     self.velocity_publisher_tb3.publish(vel_msg)
371     self.velocity_publisher_tb3.publish(Twist())
372
373     self.tb1.angle=self.tb1.yaw
374     self.tb2.angle=self.tb2.yaw
375     self.tb3.angle=self.tb3.yaw
376     self.transf_matrix_H()
377
378 def maintain_form(self):
379     global tol
380
381     if self.tb1.role =='Lider':
382         if self.tb1.x > 0.5:
383             self.form = 'Delta'
384         elif self.tb1.x < -0.5:
385             self.form = 'Delta'
386         else:
387             self.form = 'Serie'
388     if self.tb2.role =='Lider':
389         if self.tb2.x > 0.5:
390             self.form = 'Delta'
391         elif self.tb2.x < -0.5:
392             self.form = 'Delta'
393         else:
394             self.form = 'Serie'
395     if self.tb3.role =='Lider':
396         if self.tb3.x > 0.5:
397             self.form = 'Delta'
398         elif self.tb3.x < -0.5:
399             self.form = 'Delta'
400         else:
401             self.form = 'Serie'
402
403     if self.form == 'Delta':

```

## Apéndices

```
404     self.update_pose()
405     self.tb1.gx, self.tb1 gy, self.tb2.gx, self.tb2.gy, self.tb3.gx, self.tb3.gy= form.
406     first_form( self.tb1.role, self.tb2.role, self.tb3.role, self.tb1.x, self.tb1.y, self.
407     tb1.yaw, self.tb2.x, self.tb2.y, self.tb2.yaw, self.tb3.x, self.tb3.y, self.tb3.yaw)
408     self.cur_goal()
409
410
411     vel_msg1=Twist()
412     vel_msg2=Twist()
413
414     if self.tb1.role == 'Lider':
415         d_des_lf1=0.23*cos(pi/6)
416         d_act_lf1=move_p2p.euclidean_distance(self.tb1.x, self.tb1.y, self.tb2.x, self.
417 .tb2.y)*cos(move_p2p.cos_law_angle(self.tb1.x, self.tb1.y, self.tb2.x, self.tb2.y,
418 self.tb1.yaw))
419         ed_lf1=d_act_lf1-d_des_lf1
420
421     d_des_lf2=0.23*cos(pi/6)
422     d_act_lf2=move_p2p.euclidean_distance(self.tb1.x, self.tb1.y, self.tb3.x, self.
423 .tb3.y)*cos(move_p2p.cos_law_angle(self.tb1.x, self.tb1.y, self.tb3.x, self.tb3.y,
424 self.tb1.yaw))
425     ed_lf2=d_act_lf2-d_des_lf2
426
427
428     af1=move_p2p.steering_angle(self.tb2.gx, self.tb2.gy, self.tb2.x, self.tb2.y)
429     af2=move_p2p.steering_angle(self.tb3.gx, self.tb3.gy, self.tb3.x, self.tb3.y)
430
431     if not ed_lf1 < 0:
432         ea_lf1=self.yaw_f(af1, self.tb2.angle)-self.tb2.curyaw
433     else:
434         ea_lf1=0
435
436     if not ed_lf2 < 0:
437         ea_lf2=self.yaw_f(af2, self.tb3.angle)-self.tb3.curyaw
438     else:
439         ea_lf2=0
440
441     if ed_lf1 > tol:
442         if move_p2p.angle_error(self.tb2.curgx, self.tb2.curgy, self.tb2.curx, self.
443 tb2.cury, self.tb2.curyaw) > 2:
444             if self.tb2.angle + pi > pi:
445                 self.tb2.angle=self.tb2.angle - pi
446             else:
```

```
438         self.tb2.angle=self.tb2.angle + pi
439         self.transf_matrix_H()
440         self.cur_goal()
441         ea_lf1=self.yaw_f(af1,self.tb2.angle)-self.tb2.curyaw
442         self.update_pose()
443
444         vel_msg1.linear.x=move_p2p.linear_vel_if(ed_lf1,5)
445         vel_msg1.angular.z=move_p2p.angular_vel_if(ea_lf1,1.5)
446
447         self.velocity_publisher_tb2.publish(vel_msg1)
448     else:
449         self.velocity_publisher_tb2.publish(Twist())
450
451     if ed_lf2 > tol:
452         if move_p2p.angle_error(self.tb3.curgx,self.tb3.curgy,self.tb3.curx,self.
453 tb3.cury,self.tb3.curyaw) > 2:
454             if self.tb3.angle + pi > pi:
455                 self.tb3.angle=self.tb3.angle - pi
456             else:
457                 self.tb3.angle=self.tb3.angle + pi
458             self.transf_matrix_H()
459             self.cur_goal()
460             ea_lf2=self.yaw_f(af2,self.tb3.angle)-self.tb3.curyaw
461             self.update_pose()
462
463             vel_msg2.linear.x=move_p2p.linear_vel_if(ed_lf2,5)
464             vel_msg2.angular.z=move_p2p.angular_vel_if(ea_lf2,1.5)
465
466             self.velocity_publisher_tb3.publish(vel_msg2)
467     else:
468         self.velocity_publisher_tb3.publish(Twist())
469
470     elif self.form == 'Serie':
471         self.update_pose()
472         self.tb1.gx,self.tb1 gy,self.tb2.gx,self.tb2.gy,self.tb3.gx,self.tb3.gy=form.
473 second_form(self.tb1.role,self.tb2.role,self.tb3.role,self.tb1.x,self.tb1.y,self.
474 .tb1.yaw,self.tb2.x,self.tb2.y,self.tb2.yaw,self.tb3.x,self.tb3.y,self.tb3.yaw)
475         self.cur_goal()
476
477         vel_msg1=Twist()
478         vel_msg2=Twist()
```

```

476
477     if self.tb1.role == 'Lider':
478         d_des_lf1=0.23
479         d_act_lf1=move_p2p.euclidean_distance(self.tb1.x, self.tb1.y, self.tb2.x, self
480 .tb2.y)*cos(move_p2p.cos_law_angle(self.tb1.x, self.tb1.y, self.tb2.x, self.tb2.y,
481 self.tb1.yaw))
482         ed_lf1=d_act_lf1-d_des_lf1
483
484     d_des_lf2=0.23
485     d_act_lf2=move_p2p.euclidean_distance(self.tb2.x, self.tb2.y, self.tb3.x, self
486 .tb3.y)*cos(move_p2p.cos_law_angle(self.tb2.x, self.tb2.y, self.tb3.x, self.tb3.y,
487 self.tb2.yaw))
488     ed_lf2=d_act_lf2-d_des_lf2
489
490     af1=move_p2p.steering_angle(self.tb2.gx, self.tb2.gy, self.tb2.x, self.tb2.y)
491     af2=move_p2p.steering_angle(self.tb3.gx, self.tb3.gy, self.tb3.x, self.tb3.y)
492
493     if not ed_lf1 < 0:
494         ea_lf1=self.yaw_f(af1, self.tb2.angle)-self.tb2.curyaw
495     else:
496         ea_lf1=0
497
498     if not ed_lf2 < 0:
499         ea_lf2=self.yaw_f(af2, self.tb3.angle)-self.tb3.curyaw
500     else:
501         ea_lf2=0
502
503     if ed_lf1 > tol:
504         if move_p2p.angle_error(self.tb2.curgx, self.tb2.curgy, self.tb2.curx, self.
505 tb2.curz, self.tb2.curyaw) > 2:
506             if self.tb2.angle + pi > pi:
507                 self.tb2.angle=self.tb2.angle - pi
508             else:
509                 self.tb2.angle=self.tb2.angle + pi
510             self.transf_matrix_H()
511             self.cur_goal()
512             ea_lf1=self.yaw_f(af1, self.tb2.angle)-self.tb2.curyaw
513             self.update_pose()
514
515             vel_msg1.linear.x=move_p2p.linear_vel_lf(ed_lf1,5)
516             vel_msg1.angular.z=move_p2p.angular_vel_lf(ea_lf1,1.5)

```

```

512
513         self.velocity_publisher_tb2.publish(vel_msg1)
514     else:
515         self.velocity_publisher_tb2.publish(Twist())
516
517     if ed_lf2 > tol:
518         if move_p2p.angle_error(self.tb3.curgx, self.tb3.curgy, self.tb3.curx, self.
519             tb3.cury, self.tb3.curyaw) > 2:
520             if self.tb3.angle + pi > pi:
521                 self.tb3.angle = self.tb3.angle - pi
522             else:
523                 self.tb3.angle = self.tb3.angle + pi
524             self.transf_matrix_H()
525             self.cur_goal()
526             ea_lf2 = self.yaw_f(af2, self.tb3.angle) - self.tb3.curyaw
527             self.update_pose()
528
529             vel_msg2.linear.x = move_p2p.linear_vel_lf(ed_lf2, 5)
530             vel_msg2.angular.z = move_p2p.angular_vel_lf(ea_lf2, 1.5)
531
532             self.velocity_publisher_tb3.publish(vel_msg2)
533         else:
534             self.velocity_publisher_tb3.publish(Twist())
535
536     def yaw_f(self, a, angle):
537         dif = a - angle
538         if dif > pi:
539             return a - angle - 2 * pi
540         elif dif < -pi:
541             return a - angle + 2 * pi
542         else:
543             return a - angle
544 #=====
545 # Sección de transformación de coordenadas
546 def transf_matrix_H(self):
547     #Se obtiene la matriz de transformación H actual
548     self.tb1.H = lineal_transformations.cur_H(self.tb1.angle, self.tb1.x, self.tb1.y)
549     self.tb2.H = lineal_transformations.cur_H(self.tb2.angle, self.tb2.x, self.tb2.y)
550     self.tb3.H = lineal_transformations.cur_H(self.tb3.angle, self.tb3.x, self.tb3.y)
551
552     def cur_position(self):

```

```
552     #Transformamos las coordenadas actuales del robot del marco de referencia fijo  
553     #al del robot  
554     (self.tb1.curx ,self.tb1.cury)=lineal_transformations.cur_point(self.tb1.H, self.  
555     tb1.x, self.tb1.y)  
556     (self.tb2.curx ,self.tb2.cury)=lineal_transformations.cur_point(self.tb2.H, self.  
557     tb2.x, self.tb2.y)  
558     (self.tb3.curx ,self.tb3.cury)=lineal_transformations.cur_point(self.tb3.H, self.  
559     tb3.x, self.tb3.y)  
560  
561  
562     def cur_yaw(self):  
563         #Transformamos la rotacion del robot del marco de referencia fijo al actual  
564         dif_tb1=self.tb1.yaw-self.tb1.angle  
565         if dif_tb1 > pi:  
566             self.tb1.curyaw=self.tb1.yaw-self.tb1.angle-2*pi  
567         elif dif_tb1 < -pi:  
568             self.tb1.curyaw=self.tb1.yaw-self.tb1.angle+2*pi  
569         else:  
570             self.tb1.curyaw=self.tb1.yaw-self.tb1.angle  
571  
572         dif_tb2=self.tb2.yaw-self.tb2.angle  
573         if dif_tb2 > pi:  
574             self.tb2.curyaw=self.tb2.yaw-self.tb2.angle-2*pi  
575         elif dif_tb2 < -pi:  
576             self.tb2.curyaw=self.tb2.yaw-self.tb2.angle+2*pi  
577         else:  
578             self.tb2.curyaw=self.tb2.yaw-self.tb2.angle  
579  
580         dif_tb3=self.tb3.yaw-self.tb3.angle  
581         if dif_tb3 > pi:  
582             self.tb3.curyaw=self.tb3.yaw-self.tb3.angle-2*pi  
583         elif dif_tb3 < -pi:  
584             self.tb3.curyaw=self.tb3.yaw-self.tb3.angle+2*pi  
585         else:  
586             self.tb3.curyaw=self.tb3.yaw-self.tb3.angle  
587  
588     def cur_goal(self):  
589         #Transformamos las coordenadas objetivo del marco de referencia fijo al actual  
590         (self.tb1.curgx ,self.tb1.curgy)=lineal_transformations.cur_point(self.tb1.H,  
591         self.tb1.gx, self.tb1.gy)  
592         (self.tb2.curgx ,self.tb2.curgy)=lineal_transformations.cur_point(self.tb2.H,  
593         self.tb2.gx, self.tb2.gy)
```

```

587     (self.tb3.curgx, self.tb3.curgy)=lineal_transformations.cur_point(self.tb3.H,
588     self.tb3.gx, self.tb3 gy)
589
590     def update_pose(self):
591         self.cur_position()
592         self.cur_yaw()
593     #=====
594     def show_info(self):
595         print("===== ")
596         rospy.loginfo("Posicion del robot 1:\n[x = %f y = %f]",self.tb1.x,self.tb1.y)
597         rospy.loginfo("Orientacion del robot 1 (quaternion):\n[%f, %f, %f, %f]",self.
598         tb1.qx, self.tb1.qy, self.tb1.qz, self.tb1.qw)
599         rospy.loginfo("Orientacion del robot 1:\n[yaw= %f]",self.tb1.yaw)
600
601         print("===== ")
602         rospy.loginfo("Posicion del robot 2:\n[x = %f y = %f]",self.tb2.x,self.tb2.y)
603         rospy.loginfo("Orientacion del robot 2 (quaternion):\n[%f, %f, %f, %f]",self.
604         tb2.qx, self.tb2.qy, self.tb2.qz, self.tb2.qw)
605         rospy.loginfo("Orientacion del robot 2:\n[yaw= %f]",self.tb2.yaw)
606
607         print("===== ")
608         rospy.loginfo("Posicion del robot 3:\n[x = %f y = %f]",self.tb3.x,self.tb3.y)
609         rospy.loginfo("Orientacion del robot 3 (quaternion):\n[%f, %f, %f, %f]",self.
610         tb3.qx, self.tb3.qy, self.tb3.qz, self.tb3.qw)
611         rospy.loginfo("Orientacion del robot 3:\n[yaw= %f]",self.tb3.yaw)
612
613     def shutdown(self):
614         rospy.loginfo("Stopping TurtleBot... ")
615         self.velocity_publisher_tb1.publish(Twist())
616         self.velocity_publisher_tb2.publish(Twist())
617         self.velocity_publisher_tb3.publish(Twist())
618         rospy.sleep(1)
619
620     #=====
621 if __name__=='__main__':
622     try:
623         t1=TurtleBot()
624         t1.start_robot()
625     except rospy.ROSInterruptException:
626         pass

```

## A.4. Funciones Auxiliares

### Move\_p2p

```
1 #!/usr/bin/env python
2
3 from math import pow, atan2, sqrt, pi, acos, cos, sin
4
5 def euclidean_distance(gx,gy,x,y):
6     return sqrt(pow((gx-x),2)+pow((gy-y),2))
7
8 def linear_vl(gx,gy,x,y,Kv):
9     aux_vl=round(Kv*euclidean_distance(gx,gy,x,y),2)
10    if aux_vl > 0.22:
11        return 0.22
12    elif aux_vl < 0:
13        return 0
14    else:
15        return aux_vl
16
17 def steering_angle(gx,gy,x,y):
18     return atan2(gy-y,gx-x)
19
20 def angle_error(gx,gy,x,y,angle):
21     return abs(steering_angle(gx,gy,x,y)-angle)
22
23 def angular_vl(gx,gy,x,y,Ka,angle):
24     aux_va=round(Ka*(steering_angle(gx,gy,x,y)-angle),2)
25     if aux_va > 2.84:
26         return 2.84
27     elif aux_va < -2.84:
28         return -2.84
29     else:
30         return aux_va
31
32 def angular_vl2(a1,a2,Ka):
33     aux_va=round(Ka*(a1-a2),2)
34     if aux_va > 2.84:
35         return 2.84
36     elif aux_va < -2.84:
37         return -2.84
38     else:
```

```
39     return aux_va
40
41 #=====
42
43 def cos_law_angle(x1,y1,x2,y2,a_tb):
44     dlf=euclidean_distance(x1,y1,x2,y2)
45
46     #Hayamos el tercer punto del triangulo formado
47     x3=x1+dlf*cos(pi+a_tb)
48     y3=y1+dlf*sin(pi+a_tb)
49
50     A=dlf
51     B=dlf
52     C=euclidean_distance(x2,y2,x3,y3)
53
54     #Ley de cosenos CC=AA+BB-2ABcos(aC)
55     aC=acos((C*C-A*A-B*B)/(-2*A*B))
56     return aC
57
58 def linear_vel_if(e,Kv):
59     aux_vl=Kv*e
60     if aux_vl > 0.22:
61         return 0.22
62     else:
63         return aux_vl
64
65 def angular_vel_if(e,Ka):
66     aux_va=Ka*e
67     if aux_va > 2.84:
68         return 2.84
69     elif aux_va < -2.84:
70         return -2.84
71     else:
72         return aux_va
```

**Lineal\_transformations**

```
1 #!/usr/bin/env python
2
3 from math import cos, sin, pi
4 from numpy import array, transpose, matmul, linalg
5
6 def cur_H(angulo, dx, dy):
7     return array([[cos(angulo), -sin(angulo), 0, dx], [sin(angulo), cos(angulo), 0, dy], [0, 0, 1, 0], [0, 0, 0, 1]])
8
9 #Del marco de referencia fijo , al actual
10 def cur_point(H,x0,y0):
11     Hinv=linalg.inv(H) #Inversa de la matriz de transformacion
12     P0=array([x0,y0,0,1]) #Posicion destino con respecto al marco de referencia
13     P1=matmul(Hinv,P0) #Punto destino con respecto al marco actual
14
15     return P1[0], P1[1]
16
17 #Del marco de referencia actual , al fijo
18 def ref_point(H,x,y):
19     P1=array([x,y,0,1])
20     P0=matmul(H,P1)
21
22     return P0[0], P0[1]
```

## Formation

```
1 #!/usr/bin/env python
2
3 from math import cos, pi, sin
4
5 #Formacion Delta
6 def first_form(role1, role2, role3, x1, y1, a1, x2, y2, a2, x3, y3, a3):
7     roles=[role1, role2, role3]
8
9     d=0.23
10    a=5*pi/6
11
12    for i in range(len(roles)):
13        if roles[i] == 'Lider':
14            mainrole=i
15
16    if mainrole == 0:
17        g1x=x1
18        g1y=y1
19        g2x=x1+d*cos(-a+a1)
20        g2y=y1+d*sin(-a+a1)
21        g3x=x1+d*cos(a+a1)
22        g3y=y1+d*sin(a+a1)
23        return g1x, g1y, g2x, g2y, g3x, g3y
24    elif mainrole == 1:
25        g2x=x2
26        g2y=y2
27        g1x=x2+d*cos(-a+a2)
28        g1y=y2+d*sin(-a+a2)
29        g3x=x2+d*cos(a+a2)
30        g3y=y2+d*sin(a+a2)
31        return g1x, g1y, g2x, g2y, g3x, g3y
32    elif mainrole == 2:
33        g3x=x3
34        g3y=y3
35        g1x=x3+d*cos(-a+a3)
36        g1y=y3+d*sin(-a+a3)
37        g2x=x3+d*cos(a+a3)
38        g2y=y3+d*sin(a+a3)
39        return g1x, g1y, g2x, g2y, g3x, g3y
40
```

## Apéndices

```
41 #Formacion Serie
42 def second_form(role1,role2,role3,x1,y1,a1,x2,y2,a2,x3,y3,a3):
43     roles=[role1,role2,role3]
44
45     d=0.23
46     a=pi
47
48     for i in range(len(roles)):
49         if roles[i] == 'Lider':
50             mainrole=i
51
52     if mainrole == 0:
53         g1x=x1
54         g1y=y1
55         g2x=x1+d*cos(a+a1)
56         g2y=y1+d*sin(a+a1)
57         g3x=x1+2*d*cos(a+a1)
58         g3y=y1+2*d*sin(a+a1)
59         return g1x,g1y,g2x,g2y,g3x,g3y
60     elif mainrole == 1:
61         g2x=x2
62         g2y=y2
63         g1x=x2+d*cos(a+a2)
64         g1y=y2+d*sin(a+a2)
65         g3x=x2+2*d*cos(a+a2)
66         g3y=y2+2*d*sin(a+a2)
67         return g1x,g1y,g2x,g2y,g3x,g3y
68     elif mainrole == 2:
69         g3x=x3
70         g3y=y3
71         g1x=x3+d*cos(a+a3)
72         g1y=y3+d*sin(a+a3)
73         g2x=x3+2*d*cos(a+a3)
74         g2y=y3+2*d*sin(a+a3)
75         return g1x,g1y,g2x,g2y,g3x,g3y
```

## B. Planos

### B.1. Escenario de pruebas

