

UNIVERSIDAD DE COSTA RICA
Facultad de Ingeniería
Escuela de Ingeniería Eléctrica

IE0499 – Proyecto Eléctrico

**Implementación de algoritmos para navegación autónoma
en una plataforma robótica móvil F1 a escala, en entornos
de simulación virtual utilizando ROS.**

por

Esteban Rodríguez Quintana

Ciudad Universitaria Rodrigo Facio

Diciembre de 2021

Implementación de algoritmos para navegación autónoma en una plataforma robótica móvil F1 a escala, en entornos de simulación virtual utilizando ROS.

por

Esteban Rodríguez Quintana

B66076

IE0499 – Proyecto Eléctrico

Aprobado por

Ing. Leonardo José Marín Paniagua, PhD.

Profesor guía

Ing. Marvin Coto Jiménez, PhD.

Profesor lector

Ing. José David Rojas Fernández, Dr.

Profesor lector

Diciembre de 2021

Resumen

Implementación de algoritmos para navegación autónoma en una plataforma robótica móvil F1 a escala, en entornos de simulación virtual utilizando ROS.

por

Esteban Rodríguez Quintana

Universidad de Costa Rica

Escuela de Ingeniería Eléctrica

Profesor guía: Ing. Leonardo José Marín Paniagua, PhD.

Diciembre de 2021

En este proyecto se implementan algoritmos de navegación autónoma en un entorno de simulación virtual desarrollado por el proyecto de código abierto [F1TENTH](#), utilizando un conjunto de bibliotecas de *software* y herramientas especializadas en robótica conocido como [Robot Operating System \(ROS\)](#). Las implementaciones de los algoritmos seleccionados han sido realizadas en el lenguaje de alto nivel [Python](#).

La finalidad principal de este informe es recopilar información valiosa concerniente a algoritmos de evasión de obstáculos y seguimiento de trayectorias para una plataforma robótica móvil en configuración Ackermann, similar a la que será adquirida por el Laboratorio de Investigación en Ingeniería de Control ([CERLab](#)), en el corto plazo. Además, se centra en los retos técnicos planteados por la competición [F1TENTH](#). Este proyecto incluye la implementación del algoritmo de Persecución Pura, Braitenberg y además la mezcla entre ambos para combinar sus comportamientos.

En términos generales, para la selección adecuada de parámetros, los 2 primeros algoritmos implementados de manera independiente tuvieron éxito en realizar recorridos completos sin colisiones en los circuitos de prueba sin obstáculos, mientras que para el recorrido con obstáculos estáticos, la mezcla de comportamientos por ponderación también cumplió con el objetivo de seguimiento de trayectoria deseada mientras se realiza evasión de obstáculos de manera reactiva.

El código completo de los algoritmos se encuentra documentado en los Anexos y además se habilitará un repositorio para que futuros proyectos del CERLab tengan acceso a su contenido de los mismos.

Palabras claves: Robótica móvil, algoritmos, navegación, Ackermann, programación. .

Acerca de IE0499 – Proyecto Eléctrico

El Proyecto Eléctrico es un curso semestral bajo la modalidad de trabajo individual supervisado, con el propósito de aplicar estrategias de diseño y análisis a un problema de temática abierta de la ingeniería eléctrica. Es un requisito de graduación para el grado de Bachiller en Ingeniería Eléctrica de la Universidad de Costa Rica.

Abstract

Implementación de algoritmos para navegación autónoma en una plataforma robótica móvil F1 a escala, en entornos de simulación virtual utilizando ROS.

Original in Spanish. Translated as: “Implementation of algorithms for autonomous navigation in a scaled F1 mobile robotic platform, in virtual simulation environments using ROS.”

by

Esteban Rodríguez Quintana

University of Costa Rica
Department of Electrical Engineering
Tutor: Ing. Leonardo José Marín Paniagua, PhD.
December of 2021

In this project, autonomous navigation algorithms are implemented in a virtual simulation environment developed by the open source project [F1TENTH](#), using middleware with a set of software libraries and tools for robotics known as [Robot Operating System](#) (ROS).

The main purpose of this report is to collect information regarding obstacle avoidance and trajectory tracking algorithms for a mobile robotic platform in Ackermann configuration, for future investigations by the [Control Engineering Research Laboratory](#). Furthermore, it focuses on the technical challenges posed by the F1TENTH comp. This project implements the Pure Pursuit and Braitenberg algorithms and also the combination between them to mix their behaviours.

In general terms, for proper parameter selection, the first 2 implemented algorithms were successful in performing full collision-free runs on the unobstructed test circuits. For the static obstacle track, the combination of behaviours by weighting also met the desired path tracking goal while reactively performing obstacle avoidance. The complete code of the algorithms is documented in the Annexes, and a repository will also be set up so that future CERLab projects have access to its content.

Keywords: Mobile robotics, algorithms, navigation, Ackermann, programming.

About IE0499 – Proyecto Eléctrico (“Electrical Project”)

The “Electrical Project” (or “capstone project”) is a course of supervised individual work of one semester, with the purpose of applying design and analysis strategies to a problem in an open topic in electrical engineering. It is a requisite of graduation for the Bachelor of Science in Electrical Engineering, granted by the University of Costa Rica.

Dedicado a mi familia y amigos.

Agradecimientos

Agradecimientos especiales a mi familia y amigos.

Índice general

Índice general	xii
Índice de figuras	xiv
Índice de tablas	xvi
Nomenclatura	xvii
1 Introducción	1
1.1. Alcances del proyecto	3
1.2. Justificación	3
1.3. Objetivo general	4
1.4. Objetivos específicos	4
1.5. Metodología	5
1.6. Diagrama Gantt	6
2 Marco Teórico	7
2.1. Breve Introducción a la Robótica Móvil	7
2.2. Navegación Autónoma de Robots Móviles	9
2.3. <i>F1tenth</i> , Proyecto Open Source y Competición	10
2.3.1. Plataforma <i>F1TENTH</i>	10
2.3.2. Descripción y requerimientos para la competencia	12
2.4. Robot Móvil en Configuración Ackermann	14
2.4.1. Modelado del Robot Móvil en configuración Ackermann	15
2.5. Algoritmos de Navegación Autónoma	19
2.5.1. Planificación Local y Global	19
2.5.2. Algoritmos de seguimiento de trayectorias	19
2.5.3. Algoritmos de evasión de obstáculos	21
3 Herramientas utilizadas	25
3.1. Robot Operating System (ROS)	25
3.2. RViz	27

3.3. Integración ROS, RViz y simulador <i>F1TENTH</i>	27
3.4. Preparación y funcionamiento del entorno de simulación	28
4 Algoritmos implementados	31
4.1. Algoritmo de seguimiento de trayectoria	31
4.1.1. Persecución Pura	31
4.2. Algoritmo de evasión de obstáculos	35
4.2.1. Vehículo de Braitenberg: Miedo	35
4.3. Mezcla de comportamientos: Seguimiento de trayectorias + Evasión de obstáculos en alta velocidad	40
5 Simulación de algoritmos implementados	43
5.1. Pruebas iniciales	43
5.1.1. Seguimiento de trayectorias por Persecución Pura.	43
5.1.2. Evasión de obstáculos sencilla - Braitenberg modificado: Miedo	45
5.2. Seguimiento de trayectorias en alta velocidad	47
5.2.1. Implementación de Persecución Pura	47
5.2.2. Resultados preliminares de seguimiento de trayectorias para alta velocidad	55
5.3. Evasión de obstáculos estáticos	55
5.3.1. Implementación de Braitenberg modificado: Miedo	55
5.3.2. Resultados preliminares para evasión de obstáculos. Braitenberg Modificado: Miedo	66
5.4. Mezcla de comportamientos: Seguimiento de trayectorias + evasión de obstáculos en alta velocidad	68
5.4.1. Persecución Pura + Braitenberg modificado: Miedo	68
5.4.2. Resultados preliminares de mezcla de comportamientos	73
5.5. Resumen de resultados y análisis final	73
5.5.1. Tabla comparativa de resultados	73
6 Conclusiones y recomendaciones	75
6.1. Conclusiones	75
6.2. Recomendaciones	77
6.3. Trabajos futuros	78
Bibliografía	79
A Códigos de implementación	83
A.1. Persecución Pura	83
A.1.1. Nodo de algoritmo: Persecución Pura	83
A.2. Braitenberg Modificado: Miedo	89
A.2.1. Nodo de algoritmo: Braitenberg Modificado: Miedo	89
A.3. Mezcla de comportamientos	91

A.3.1.	Planificador Local	91
A.3.2.	Planificador Global	94
A.3.3.	Nodo de algoritmo: mezcla de comportamientos	96
A.3.4.	Archivos de configuración y launch file	98
A.4.	Misceláneos generales, launch files y archivos de configuración	101
A.4.1.	path_plotter.py	101
A.4.2.	raceline_viz.py	102
A.4.3.	raceline_viz.py	103
A.4.4.	autonomous_navigation_nodes.launch	105
A.4.5.	raceline_draw.launch	106
A.4.6.	Archivos de parámetros para algoritmos de Persecución Pura y Braitenberg Modificado: Miedo	107

Índice de figuras

2.1.	Robots de tipo biológico. Extraída de wwwwhatsnew.com	7
2.2.	Robot tipo aéreo. Extraída de wikimedia.org	8
2.3.	Robots con ruedas en configuración Mecanum. Extraída de cerlab.ucr.ac.cr	8
2.4.	Niveles de autonomía de conducción según SAE (J 3016).	9
2.5.	Capacidades necesarias para la navegación autónoma.	10
2.6.	Vehículo <i>F1TENTH</i> . Tomado de racecarj.com	11
2.7.	Distribución de niveles vehículo <i>F1TENTH</i> . Tomado de F1TENTH, build page	11
2.8.	Distribución de niveles vehículo <i>F1TENTH</i> . Tomado de F1TENTH, build page	12
2.9.	Abstracción de componentes esenciales para la navegación autónoma.	13
2.10.	Estructura básica de la configuración Ackermann.	14
2.11.	Diagrama para modelado de la configuración Ackermann sin simplificar.	17
2.12.	Diagrama para modelado de la configuración Ackermann simplificado al modelo de bicicleta.	17
2.13.	Algoritmo de Persecución Pura 1. Adaptado de [11].	20
2.14.	Algoritmo de Persecución Pura 2. Adaptado de [11].	20
2.15.	Vehículo de Braitenberg 1.	22
2.16.	Vehículo de Braitenberg 2a (Miedo) y 2b (Agresividad). Adaptada de [9]	22
3.1.	Composición gráfica de ROS. Tomada de ROS, ecosystem	26
3.2.	Distribuciones de ROS recientes	26
3.3.	Interfaz gráfica de RViz	27
3.4.	Visualización del entorno de simulación	28
3.5.	Árbol de nodos del entorno de simulación	29
4.1.	Diagrama de algoritmo de Persecución Pura.	32
4.2.	Diagrama de algoritmo de Persecución Pura. Adaptado de [11].	32
4.3.	Gráfico obtenido con <code>rqt_graph</code> para el algoritmo de Persecución Pura.	34
4.4.	Vehículo de Braitenberg 2a (derecha) vs. Implementación (izquierda). [9]	36
4.5.	Diagrama de Braitenberg Modificado: Miedo implementado.	37
4.6.	Gráfico obtenido con <code>rqt_graph</code> para el algoritmo de Braitenberg Modificado: Miedo.	38
4.7.	Gráfico obtenido con <code>rqt_graph</code> para la mezcla Persecución Pura-Braitenberg Modificado .	40

5.1.	Prueba inicial para Persecución Pura, <i>lookahead</i> variable y velocidad constante $v = 4 \text{ m/s}$	44
5.2.	Prueba inicial para Persecución Pura, $lookahead = 2 \text{ m}$ y velocidad = $4 \text{ m/s} * P_{v_{gain}}$	45
5.3.	Prueba inicial de algoritmo de Braitenberg Modificado: Miedo	46
5.4.	Prueba inicial de algoritmo de Braitenberg Modificado: Miedo. Ampliación 1	46
5.5.	Prueba inicial de algoritmo de Braitenberg Modificado: Miedo. Ampliación 2	46
5.6.	Simulación de Persecución Pura para circuito 1	48
5.7.	Simulación de Persecución Pura para circuito 1. Ampliación 1	49
5.8.	Simulación de Persecución Pura para circuito 1. Ampliación 2	49
5.9.	Error en distancia para x, Persecución Pura para circuito 1	50
5.10.	Error en distancia para y, Persecución Pura para circuito 1	50
5.11.	Simulación de Persecución Pura para circuito 2: Catalunya	51
5.12.	Simulación de Persecución Pura para circuito 2: Catalunya. Ampliación 1	52
5.13.	Simulación de Persecución Pura para circuito 2: Catalunya. Ampliación 2	53
5.14.	Error en distancia para x, Persecución Pura para circuito 2: Catalunya	54
5.15.	Error en distancia para y, Persecución Pura para circuito 2: Catalunya	54
5.16.	Simulación de Braitenberg Modificado: Miedo para circuito 1. $n = 4$	56
5.17.	Simulación de Braitenberg Modificado: Miedo para circuito 1. $n = 4$. Ampliación 1	57
5.18.	Simulación de Braitenberg Modificado: Miedo para circuito 1. $n = 4$. Ampliación 2	57
5.19.	Simulación de Braitenberg Modificado: Miedo para circuito 1. $n = 12$	58
5.20.	Simulación de Braitenberg Modificado: Miedo para circuito 1. $n = 12$	59
5.21.	Simulación de Braitenberg Modificado: Miedo para circuito 2: Catalunya. $n = 4$	60
5.22.	Simulación de Braitenberg Modificado: Miedo, circuito 2: Catalunya. $n = 4$. Ampliación 1	61
5.23.	Simulación de Braitenberg Modificado: Miedo, circuito 2: Catalunya. $n = 4$. Ampliación 2	61
5.24.	Simulación de Braitenberg Modificado: Miedo para circuito 2: Catalunya. $n = 6$	62
5.25.	Simulación de Braitenberg Modificado: Miedo, circuito 2: Catalunya. $n = 6$. Ampliación 1	63
5.26.	Simulación de Braitenberg Modificado: Miedo, circuito 2: Catalunya. $n = 6$. Ampliación 2	63
5.27.	Simulación de Braitenberg Modificado: Miedo para circuito 2: Catalunya. $n = 6$. Ampliación 3	64
5.28.	Simulación de Braitenberg Modificado: Miedo para circuito 2: Catalunya. $n = 12$	65
5.29.	Simulación de Braitenberg Modificado: Miedo, circuito 2: Catalunya. $n = 12$. Ampliación 1	65
5.30.	Simulación de Braitenberg Modificado: Miedo, circuito 2: Catalunya. $n = 12$. Ampliación 1	66
5.31.	Simulación de mezcla de comportamientos, circuito 2: Catalunya	69
5.32.	Simulación de mezcla de comportamientos, circuito 2: Catalunya. Ampliación 1	70
5.33.	Simulación de mezcla de comportamientos, circuito 2: Catalunya. Ampliación 2	71
5.34.	Simulación de mezcla de comportamientos, circuito 2: Catalunya. Ampliación 3	71
5.35.	Error en distancia para x. Mezcla de comportamientos	72
5.36.	Error en distancia para y. Mezcla de comportamientos	72

Índice de tablas

2.1. Pseudo código algoritmo de Persecución Pura	21
2.2. Pseudo código algoritmo de Braitenberg Modificado: Miedo	23
4.1. Pseudo código algoritmo de Persecución Pura	34
4.2. Implementación del algoritmo de Persecución Pura, pseudo descripción del nodo auxiliar . .	35
4.3. Implementación del algoritmo de Persecución Pura, pseudo descripción del nodo principal.	35
4.4. Pseudo código algoritmo de Braitenberg Modificado: Miedo	38
4.5. Implementación del algoritmo de Braitenberg Modificado: Miedo, pseudo descripción del nodo.	39
4.6. Implementación de mezcla de comportamientos PP + Braitenberg modificado, pseudo descripción del nodo.	41
4.7. Pseudo código algoritmo de PP + Braitenberg modificado	41
5.1. Tabla resumen de parámetros utilizados en pruebas iniciales.	43
5.2. Tabla resumen de resultados para Braitenberg Modificado: Miedo.	67
5.3. Tabla comparativa de índices de desempeño	74

Nomenclatura

β	Ángulo de derrape
$\ddot{\square}$	Segunda derivada de una determinada variable
δ	Ángulo de dirección
δ_l	Ángulo de dirección de rueda izquierda
δ_r	Ángulo de dirección de rueda derecha
$\dot{\square}$	Primera derivada de una determinada variable
γ	Radio de curvatura
μ	Coeficiente de fricción
$\overline{\square}$	Máximo valor posible
Ψ	Ángulo de rumbo
x	Ubicación del centro de gravedad en x
y	Ubicación del centro de gravedad en y
\square_{lat}	Variable en dirección lateral
\square_{long}	Variable en dirección longitudinal
θ_{actual}	Orientación en y actual
$\underline{\square}$	Mínimo valor posible
a	Aceleración
$C_{f,r}$	Coeficiente de rigidez en curvas frontal/trasera
$d\theta$	Derivada de orientación
d_{prom_n}	Distancia promedio n

d_{umbral}	Distancia de umbral
dx	Derivada de posición en x
dy	Derivada de posición en y
$F_{z,f}$	Fuerza vertical sobre en el eje delantero
$F_{z,r}$	Fuerza vertical sobre en el eje trasero
g	Gravedad
h_{cg}	Altura del centro de gravedad
I_z	Inercia del vehículo alrededor de z
l	Distancia lookahead
l_{wr}	Distancia de separación entre las ruedas
m	Masa del vehículo
n	Cantidad de segmentos de medición
P_{vgain}	Ganancia de velocidad
r	Radio de giro
v	Velocidad
v_s	Velocidad de ángulo de dirección
v_s	Velocidad máxima sin derrape
x_{actual}	Posición en x actual
y_{actual}	Posición en y actual
\square	Espacio para colocar una determinada variable
$GSTRW$	Global Streering Weight
GSW	Global Speed Weight
l_f	Distancia del centro de gravedad al eje delantero
l_r	Distancia del centro de gravedad al eje trasero
LSW	Local Speed Weight

LSW Local Streering Weight

v_{weighted} velocidad resultante de ponderación

EIE Escuela de Ingeniería Eléctrica de la Universidad de Costa Rica

IEEE Instituto de Ingenieros Eléctricos y Electrónicos (del inglés *Institute of Electrical and Electronics Engineers*)

SI Sistema internacional de unidades

Capítulo 1

INTRODUCCIÓN

El término *robot* viene del idioma checo, en el cual “*robota*” significa fuerza de trabajo. Este término fue utilizado por primera vez en 1921 por el dramaturgo checo Karel Capek, en su obra teatral “*Rossum’s Universal Robots*” [7]. A lo largo de las últimas décadas, desde los años cercanos a 1960: con la explosión de la revolución industrial; hasta la actualidad: con la aparición de tecnologías de la industria 3.0 y temas como la automatización de procesos y el creciente interés en la exploración interplanetaria, la robótica ha adquirido un papel fundamental en el desarrollo de gran cantidad de tecnologías y aplicaciones de todo tipo. Entre ellas podemos citar la automatización de procesos industriales y de producción, industria biomédica, soluciones de transporte, aplicaciones espaciales y de comunicaciones, entre otras. La robótica es un campo de estudio multidisciplinario que requiere del conocimiento y aporte de diversas ramas de las ciencias y la tecnología, como la ingeniería eléctrica, ingeniería mecánica e ingeniería de control, así como, las ciencias de la computación, física, matemáticas y estadística [34], por mencionar solo algunas.

A lo largo de las últimas décadas, una enorme cantidad de investigaciones y progresos se han alcanzado desarrollando tecnologías inteligentes como el control crucero adaptivo, asistencia de mantenimiento de línea, algoritmos de seguimiento de línea o trayectorias y algoritmos de toma de decisiones [10]. Los robots móviles autónomos, tema central de este proyecto, son robots que poseen la capacidad de navegar de forma autónoma a través del entorno que les rodea, mientras desarrollan tareas orientadas a cumplir un objetivo relacionado con su posición y orientación [8] o postura. Dicha capacidad de navegación autónoma dependerá tanto de la configuración física y mecánica del robot, como de su habilidad de percepción del entorno, capacidad de cómputo y de los algoritmos que ríjan su comportamiento.

La naturaleza de este proyecto está ligada a la investigación e implementación de algoritmos de navegación en lenguajes de alto nivel en ambientes de simulación virtual, y estará centrado en dar soluciones de navegación autónoma enfocadas en la competición de vehículos de Fórmula 1 a escala 1/10: *F1TENTH*. Este trabajo se desarrolla en aporte a las labores de investigación del CERLab (*Control Engineering Research Laboratory*) de la Escuela de Ingeniería Eléctrica de la Universidad de Costa Rica.

Finalmente, antes de continuar, se le recuerda al lector que este trabajo sigue las normas del SI¹, por consiguiente, las distancias se miden en metros (*m*), los ángulos en grados (*º*) y las velocidades en

¹Sistema Internacional de Unidades.

metros sobre segundo ($m/s.$)

1.1. Alcances del proyecto

Con este proyecto se desea centralizar información relacionada con la competencia *F1TENTH* para entender las necesidades técnicas que esta plantea desde el punto de vista del modelado e implementación de algoritmos de navegación autónoma con seguimiento de trayectorias y evasión de obstáculos para un robot móvil en configuración Ackermann de Fórmula 1 a escala, y poder encontrar así, solución a algunos de estos requerimientos, además de, esclarecer el camino para futuras implementaciones utilizando la plataforma *F1TENTH* en el laboratorio del CERLab. Lo anterior utilizando lenguajes de programación de alto nivel y herramientas de simulación virtual. No se abarca el desarrollo de herramientas de simulación, ni su modificación o implementación de modelado 3D, ya que estas herramientas serán utilizadas como cajas grises, en otras palabras, se asume que el funcionamiento interno del simulador es adecuado y se procurará no realizar modificaciones mayores al mismo.

Asimismo, se abarca la descripción de las características físicas del robot, así como su modelado mecánico y parametrización. Sin embargo, no se desarrolla ni deduce el modelo del sistema mecánico, ya que se utiliza el modelado dado por fuentes bibliográficas [5] y [1] y por el ambiente de simulación. No se incluye la deformación de las ruedas en el modelo del sistema. Además, no se contempla la descripción teórica de la sintonización de los controladores, ni el modelado de motores DC o componentes electrónicos. El trabajo se desarrolla bajo condiciones de superficie horizontal en entornos cerrados y de alta velocidad. Además, se contempla el deslizamiento lateral de las ruedas y del vehículo las cuales son características dinámicas modeladas por el simulador que brinda la organización *F1TENTH*.

1.2. Justificación

El fin principal de este proyecto eléctrico es realizar aportes significativos a las labores que se desempeñan en el Laboratorio de Investigación en Ingeniería de Control o CERLab (*Control Engineering Research Laboratory*) de la Universidad de Costa Rica. Parte de la investigación que se desarrolla en el CERLab está relacionada con la robótica móvil colaborativa, y dentro de este tema, surge la necesidad de desarrollar plataformas robóticas móviles funcionales para la implementación de algoritmos y soluciones, además de, la necesidad de dotar de conocimiento técnico al estudiantado que colabora en dichas investigaciones. El proceso de adquisición de conocimiento y herramientas técnicas para las labores del laboratorio suele representar una curva de aprendizaje compleja. Por otra parte, las plataformas son sistemas que deben ser utilizados con precaución para evitar daños y accidentes. Por las razones anteriores, se ha hecho una serie de esfuerzos por desarrollar proyectos eléctricos que permitan realizar simulación, implementación, pruebas y validación de algoritmos para las plataformas móviles de interés, y que dan solución a distintas necesidades del laboratorio, además de disminuir y administrar de mejor manera el tiempo de desarrollo e implementación de las plataformas. Este proyecto es parte de dichos esfuerzos.

Adicionalmente, el aporte más significativo de este trabajo es el de ser una recopilación de información, consideraciones, algoritmos e implementaciones a nivel de simulación que permitirán el desarrollo de una plataforma robótica móvil en configuración Ackermann de la competencia y proyecto *Open Source: F1TENTH*. Dicha plataforma móvil Ackermann será adquirida por el CERLab en el corto plazo.

1.3. Objetivo general

Implementar una plataforma robótica móvil en configuración Ackermann para la validación de algoritmos de seguimiento de trayectoria y evasión de obstáculos en condición de plano horizontal, utilizando entornos de simulación virtual basados ROS y en el modelo de Fórmula 1 a escala del proyecto *Open Source* y competencia *F1TENTH*.

1.4. Objetivos específicos

1. Describir las características físicas, así como el modelado matemático y la parametrización de la plataforma Ackermann *F1TENTH*.
2. Centralizar la información de todos los algoritmos para navegación autónoma requeridos para la plataforma Ackermann, enfocados en las necesidades técnicas para la competición *F1TENTH*.
3. Investigar el funcionamiento de ROS y su interacción con el software RViz, para el modelado y simulación de la evolución de la postura de una plataforma robótica Ackermann *F1TENTH*.
4. Implementar al menos un algoritmo de navegación autónoma de seguimiento de trayectorias en lenguaje de alto nivel para una plataforma móvil *F1TENTH*, en un plano horizontal.
5. Implementar al menos un algoritmo de evasión de obstáculos en lenguaje de alto nivel para una plataforma móvil *F1TENTH*, en un plano horizontal.
6. Implementar al menos un algoritmo de evasión de obstáculos con seguimiento de trayectoria en lenguaje de alto nivel para una plataforma móvil *F1TENTH*, en un plano horizontal.
7. Validar el funcionamiento de los algoritmos de seguimiento de trayectoria, evasión de obstáculos y comportamientos combinados implementados, mediante el software de simulación RViz y el cálculo de índices de desempeño para la plataforma propuesta en al menos dos circuitos con obstáculos diferentes.

1.5. Metodología

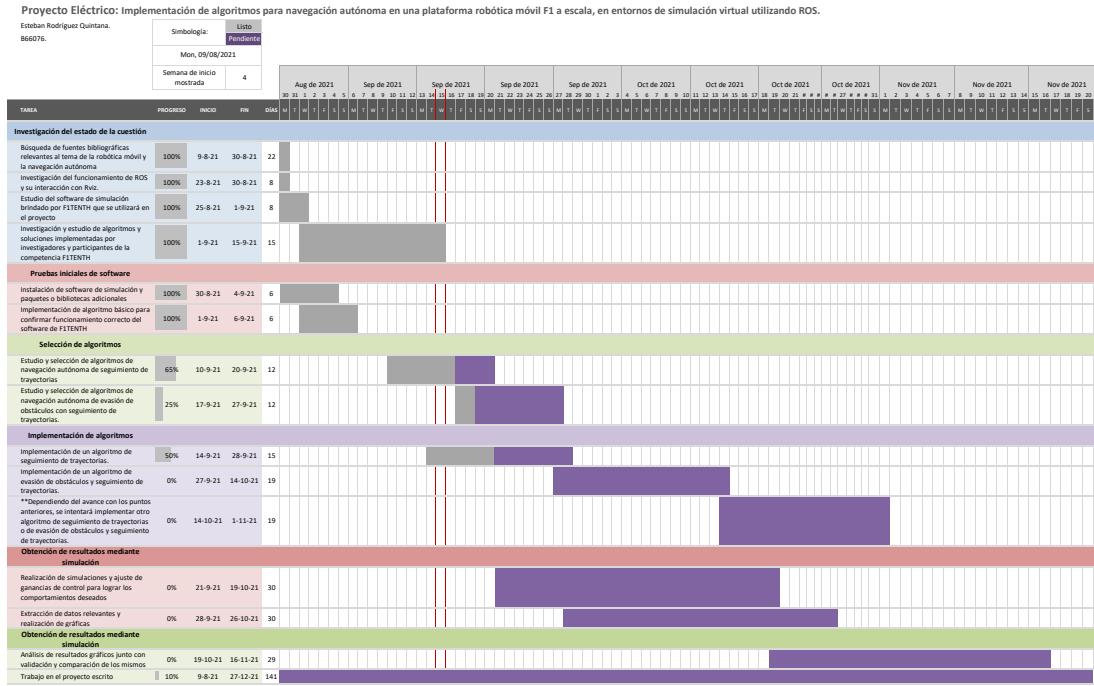
A continuación, se describen las actividades que se realizarán durante el desarrollo de este proyecto, todas alineadas al cumplimiento de cada uno de los objetivos:

1. Investigación del estado de la cuestión: Con el objetivo de describir la plataforma Ackermann y de centralizar la información requerida para la competencia *F1TENTH*.
 - a) Investigación bibliográfica del tema de la robótica móvil y la navegación autónoma.
 - b) Investigación del funcionamiento de ROS y su interacción con RViz.
 - c) Estudio del software de simulación brindado por *F1TENTH* que se utilizará en el proyecto.
 - d) Investigación y estudio de algoritmos y soluciones implementadas por investigadores y participantes de la competencia *F1TENTH*.
2. Pruebas iniciales de software: Para asegurar funcionamiento de las herramientas.
 - a) Instalación de software de simulación y paquetes o bibliotecas adicionales.
 - b) Implementación de algoritmo básico para confirmar funcionamiento correcto del software de *F1TENTH*.
3. Selección de algoritmos:
 - a) Estudio y selección de algoritmos de navegación autónoma de seguimiento de trayectorias.
 - b) Estudio y selección de algoritmos de navegación autónoma de evasión de obstáculos.
 - c) Estudio y selección de algoritmos de navegación autónoma de evasión de obstáculos con seguimiento de trayectorias.
4. Implementación de algoritmos:
 - a) Implementación de al menos un algoritmo de seguimiento de trayectorias.
 - b) Implementación de al menos un algoritmo de evasión de obstáculos.
 - c) Implementación un algoritmo de evasión de obstáculos con seguimiento de trayectorias.
5. Obtención de resultados mediante simulación: simulaciones en al menos dos circuitos distintos con y sin obstáculos.
 - a) Realización de simulaciones y ajuste de ganancias de control para lograr los comportamientos deseados.
 - b) Extracción de datos relevantes y realización de gráficas.
6. Análisis de resultados:
 - a) Análisis de resultados gráficos junto con validación y comparación de los mismos.

Paralelamente a cada una de estas etapas se irá documentando la información y procedimientos pertinentes para su posterior inclusión en el trabajo escrito.

1.6. Diagrama Gantt

En este diagrama se contemplan las etapas de desarrollo del proyecto a manera de guía. No se incluye la realización de presentaciones ni tareas secundarias, ya que estas se encuentran expresas en el cronograma del curso. El diagrama posee un formato que permite ampliarlo sin distorsión.



Capítulo 2

MARCO TEÓRICO

2.1. Breve Introducción a la Robótica Móvil

Con el fin de dar contexto a la temática de este proyecto planteamos la siguiente pregunta: ¿Qué es un robot móvil? Un robot móvil es un conjunto de subsistemas robóticos (articulaciones, patas, ruedas, sensores, procesamiento, etc.) que requieren de mecanismos de locomoción que les permitan moverse sin limitaciones a través de su entorno [41] y realizar distintas tareas objetivo [8]. Cuando se habla de movimiento sin limitaciones, entiéndase que su estructura principal no está anclada a un punto fijo en el espacio. Hay una larga variedad de formas en las que un robot de este tipo puede movilizarse en su entorno, y se clasifican de acuerdo al espacio físico en que se movilizan, los mecanismos que usan para dicho fin y la disposición física y espacial que estos mecanismos tienen en el cuerpo del robot, algunos ejemplos de estos son:

- Los robots con extremidades que simulan mecanismos de movimiento biológico de seres humanos o animales, como los robots bípedos y cuadrúpedos.



Figura 2.1: Robots de tipo biológico. Extraída de wwwwhatsnew.com

- Robots que se mueven a través de los medios como el agua o el aire.



Figura 2.2: Robot tipo aéreo. Extraída de [wikimedia.org](#)

- Los robots que se movilizan empleando ruedas impulsadas activamente, como los de configuración diferencial, Mecanum y Ackermann.



Figura 2.3: Robots con ruedas en configuración Mecanum. Extraída de [cerlab.ucr.ac.cr](#)

Este proyecto se centra en un robot móvil de tipo Ackermann, que se moviliza en planos horizontales y que considera el derrape dentro de su modelado.

Como la robótica es un campo de estudio tan amplio y contemporáneo, hay aplicaciones que aún se encuentran en etapas de investigación y desarrollo, mientras otras ya son implementadas de manera más extensiva, por ejemplo, en el ámbito de industria para realizar tareas en líneas de producción o en el campo de la agricultura [28], también se emplea la robótica móvil en áreas como: la inspección, vigilancia, mantenimiento [8], movilidad, búsqueda y rescate, educación, entre otras.

2.2. Navegación Autónoma de Robots Móviles

La navegación autónoma de robots móviles se refiere a la capacidad que estos poseen de tomar decisiones por sí mismo, basadas en el entorno y su percepción del mismo, razón por la cual los vehículos autónomos requieren de métodos de localización robustos y precisos, especialmente cuando se mueven a alta velocidad [42]. Existen distintos niveles de autonomía en la conducción de vehículos y robots móviles. Según el organismo de estándares: SAE (*Society of Automation Engineers*), hay 6 niveles de autonomía de conducción¹. En la figura 2.4, se muestra un resumen de estos niveles de autonomía de conducción para agregar más contexto a la investigación desarrollada en este proyecto eléctrico.



Figura 2.4: Niveles de autonomía de conducción según SAE (J 3016).

Para que un robot se pueda movilizar en su entorno de manera semi autónoma o autónoma requiere de las siguientes capacidades elementales:

- Percepción.
- Localización.
- Estrategias o algoritmos de planificación y navegación.
- Sistema de control de movimiento.

En la figura 2.5 se resumen estas capacidades necesarias para la navegación autónoma. Además, en cuanto a estas mismas capacidades, la figura 2.9 muestra una abstracción de los componentes esenciales en el campo de la navegación autónoma de robots móviles.

¹Más información en: [SAE, levels of driving automation](#).



Figura 2.5: Capacidades necesarias para la navegación autónoma.

2.3. **F1tenth, Proyecto Open Source y Competición**

F1TENTH es una comunidad internacional de investigadores, ingenieros y entusiastas de los sistemas autónomos. Se fundó originalmente en la Universidad de Pensilvania en 2016, pero desde entonces se ha extendido a muchas otras instituciones en todo el mundo. Su misión es fomentar el interés, la emoción y el pensamiento crítico sobre el campo, cada vez más omnipresente, de los sistemas autónomos, y especialmente en la robótica móvil. El proyecto *F1TENTH* consta de 4 pilares fundamentales que son: Construir, aprender, competir e investigar.² Comúnmente, una vez al año, se organiza una competencia en conferencias internacionales, en ella participan vehículos que siguen los parámetros del proyecto y que además son desarrollados tanto por aficionados como estudiantes de universidades de todo el mundo. En las últimas versiones de la competición se han incorporado herramientas de simulación virtual, para realizar las competiciones en ella y no necesariamente tener que trasladarse a otras locaciones e incluso optimizar el proceso de desarrollo de los vehículos.

2.3.1. Plataforma *F1TENTH*

La plataforma que utiliza *F1TENTH* es un robot móvil de configuración Ackermann, simulando un vehículo de Fórmula 1. Este consta de un chasis, ruedas, motores, unidades de procesamiento y tarjeta gráfica, así como un sensor LiDAR, encoders, acelerómetros, giróscopos y magnetómetros y otras partes que se pueden encontrar en el *Bill Of Materials* disponible en [Master BOM](#).

En la figura 2.6, se muestra la estructura del robot *F1TENTH*. En la figura 2.7, se puede observar la división en niveles que posee la plataforma móvil.

²Puede consultar [F1TENTH, home page](#).

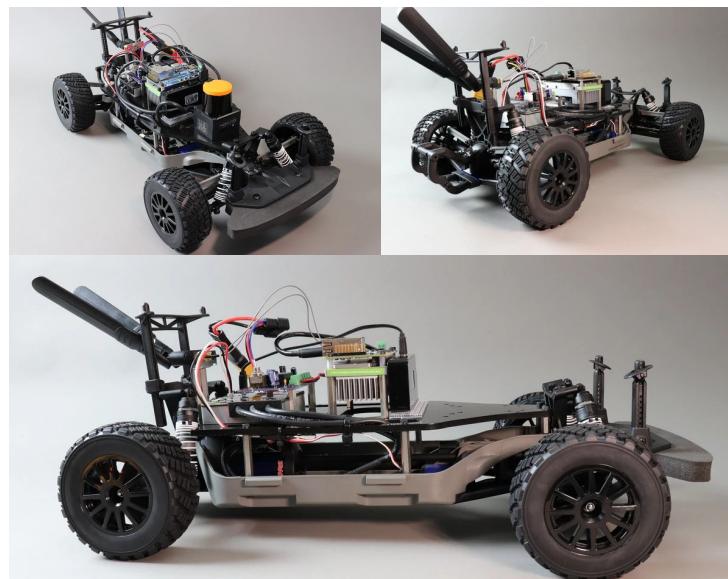


Figura 2.6: Vehículo F1TENTH. Tomado de racecarj.com

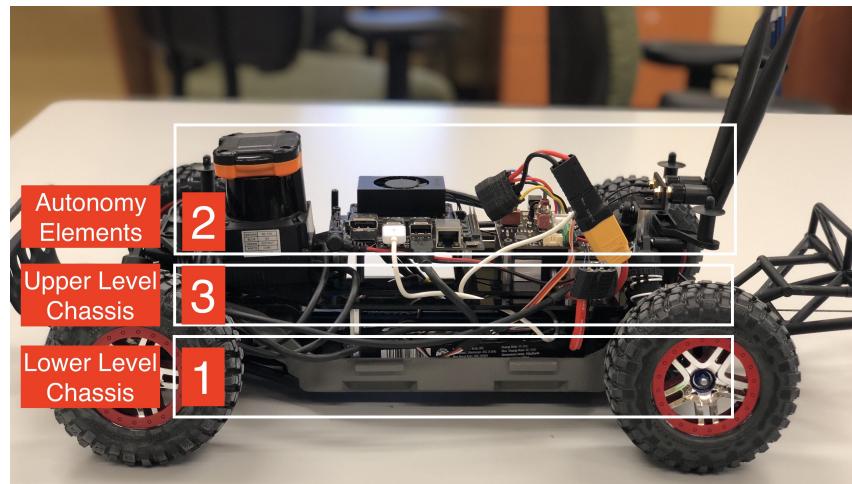


Figura 2.7: Distribución de niveles vehículo F1TENTH. Tomado de F1TENTH, build page

Adicionalmente, en la figura 2.8 se muestra un diagrama de muy alto nivel de los elementos principales que componen al robot móvil F1TENTH.

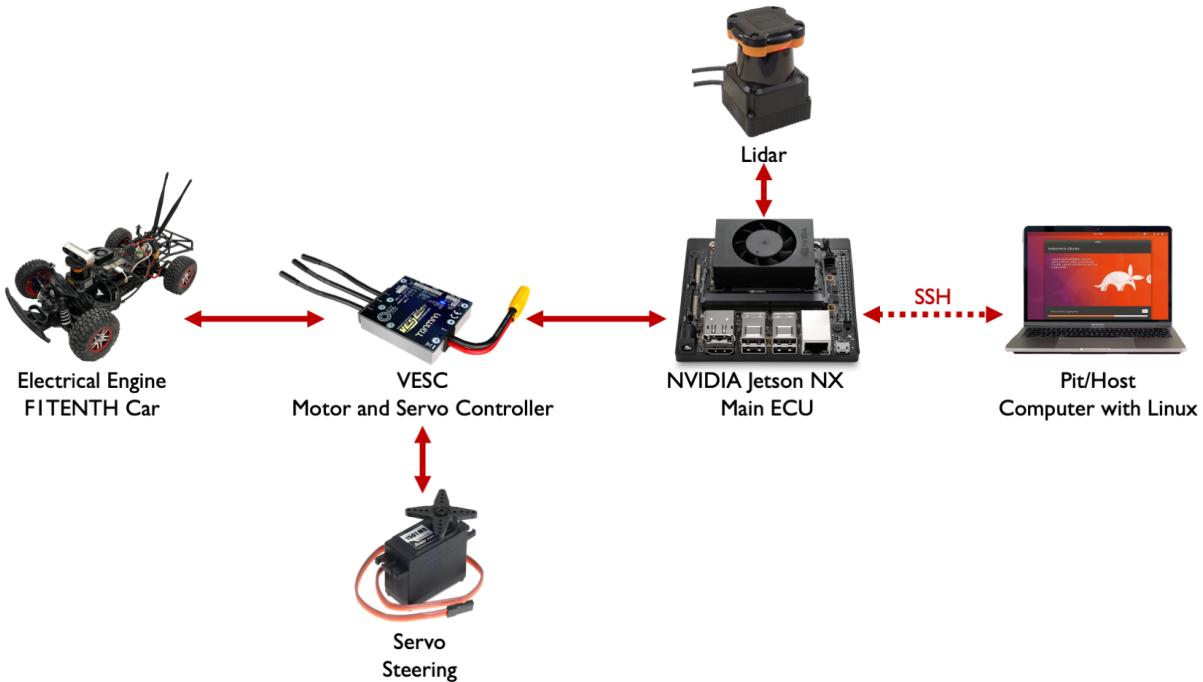


Figura 2.8: Distribución de niveles vehículo *F1TENTH*. Tomado de [F1TENTH, build page](#)

2.3.2. Descripción y requerimientos para la competencia

Se recomienda consultar [F1TENTH Build](#), en donde está disponible una wiki completa para iniciar con la construcción, configuración, instalación y simulación del proyecto de la comunidad *F1TENTH*. Para obtener información de las competencias realizadas dentro de la comunidad, sus requisitos, requerimientos, fechas y más eventos puede consultar la sección de [RACE](#) en [f1tenth.org/](#). Adicionalmente, la comunidad cuenta con un [Slack](#) en donde se pueden hacer consultas respecto a la competencia y las plataformas desarrolladas por la comunidad.

La competencia se ha realizado de manera virtual y presencial. Generalmente, se organizan varias fechas al año para eventos específicos de robótica y congresos internacionales como los organizados por [IFAC \(International Federation of Automatic Control\)](#). El objetivo de la competencia es que los vehículos, reales o simulados, completen recorridos de un circuito de competencias sin colisionar con obstáculos ni otros vehículos. Ha sido habitual que la competencia funcione por rondas en donde las plataformas compiten 1 contra 1 en el circuito y la que realiza el recorrido en menor tiempo es la que avanza a la siguiente etapa, cabe mencionar que cuando un vehículo colisiona contra un límite físico de la pista o el oponente se le da una penalización de tiempo, por lo cual, la evasión de obstáculos es un requisito indispensable para poder competir. En la figura 2.9 se muestra una abstracción de los componentes necesarios para la navegación autónoma.

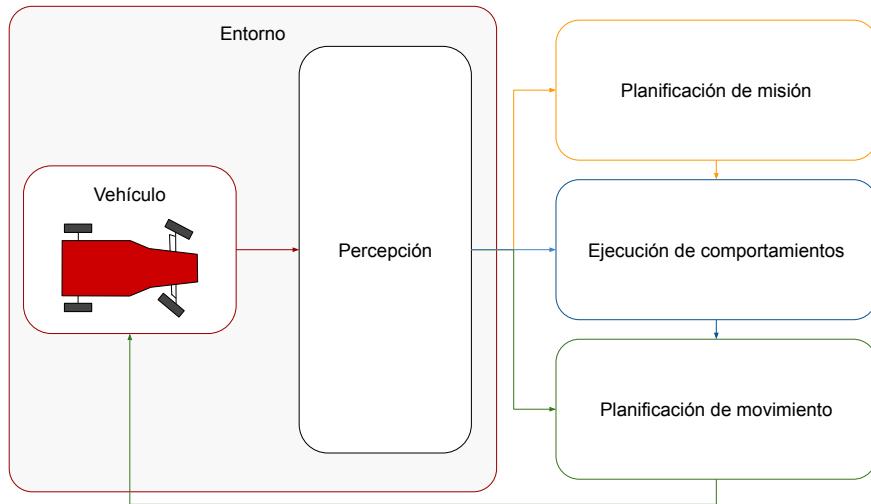


Figura 2.9: Abstracción de componentes esenciales para la navegación autónoma.

Parte de los requerimientos necesarios para que el vehículo logre cumplir los objetivos de la competición se mencionan a continuación, y también, se pueden estudiar en [F1TENTH - Course Documentation](#), donde encontrará capacitación en temas importantes para el desarrollo de la plataforma y su puesta en marcha, además de, información relacionada con técnicas y estrategias de carrera como:

- Métodos reactivos: *Follow the gap* [40] y variantes.
- Mapeo y localización: *Scan Matching* [16], filtro de partículas [24] y SLAM [46].
- Planificación: Persecución Pura [11,39,44], RRT (*Rapidly-exploring Random Tree*) [21], generación y seguimiento de trayectorias por MPC [10, 23] y optimización de velocidad [19] y la línea de carrera [36].
- Visión, percepción y detección: detección y estimación de pose [3], expansión de simple vista por geometría multi vista [22] y detección de objetos con YOLO (*You Only Look Once*) [12].
- Métodos de control [18, 43, 45].

Dentro de [F1TENTH - Course Documentation](#) podrá encontrar recursos valiosos en las lecturas 20 hasta 25, donde se abordan temas especiales para la navegación autónoma y la competencia.

En [14] se abordan la planificación de movimiento, generación de trayectorias, restricciones de estado, modelado de vehículos, parametrización del control, generación, evaluación y optimización de trayectorias, perfiles de velocidad, cambio de línea, giros en U, conducción defensiva, detección de error y recuperación, planificación de maniobras complejas, incorporación de obstáculos dinámicos, seguimiento de trayectorias complejas y estacionamiento en parqueos. Por lo anterior, es una gran fuente de información y que además posee gran cantidad de referencias a otros trabajos y publicaciones científicas.

2.4. Robot Móvil en Configuración Ackermann

Este trabajo está centrado en la configuración de robot móvil en configuración Ackermann, y en esta sección extenderemos su estructura, modelado y peculiaridades. La configuración Ackermann, comúnmente conocida como: *car-like* [26], por su uso extensivo en los automóviles, consta de: 4 ruedas que se encuentran colocadas en dos ejes, uno en la parte delantera y otro en la parte trasera del robot o vehículo. En el eje delantero, se ubican las dos ruedas destinadas a dar dirección y en el eje trasero las ruedas encargadas de dar propulsión al vehículo, tal y como se muestra en la figura 2.10.

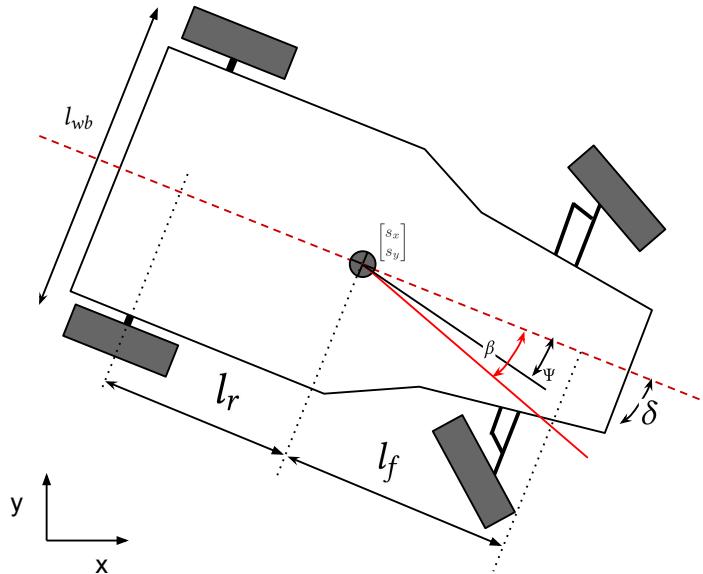


Figura 2.10: Estructura básica de la configuración Ackermann.

Como se observa en la figura 2.10, en el eje de dirección, cuyas ruedas pueden girar alrededor de su eje z, el ángulo de orientación de las ruedas es distinto entre ellas cuando se realiza una maniobra de giro. Esta peculiaridad de diseño mecánico es debida a que la rueda interior a la curva se encuentra a una distancia menor del eje de rotación del vehículo que la rueda exterior a la curva; lo anterior para evitar que las ruedas delanteras derrapen, ya que geométricamente su recorrido es distinto alrededor del eje de rotación del cuerpo del vehículo. Respecto al punto anterior, cabe mencionar que se puede dar derrape en condiciones de alta velocidad, más no debido a la geometría intrínseca del vehículo gracias al diseño mecánico Ackermann.

En resumen, la rueda exterior a la curva recorre una mayor distancia que la rueda interior a la curva durante un giro, esto debido a la restricción cinemática que ejerce el eje trasero sobre el cuerpo del vehículo.

La descripción matemática de la configuración a trabajar permite definir el estado del robot (postura) y la detección de objetos en el entorno, lo cual es esencial para las tareas de localización y navegación [8].

En este sentido, se procede a definir el modelado del robot móvil en configuración Ackermann en la siguiente sub sección.

2.4.1. Modelado del Robot Móvil en configuración Ackermann

Esta sección se basa en los aportes de [1, 5], donde se desarrolla el modelado del vehículo en configuración Ackermann, desde un modelo sencillo: Punto-Masa³, hasta un modelo mucho más complejo: Multicuerpo⁴. Para efectos de este proyecto y de la herramienta de simulación brindada por el proyecto *F1TENTH*, se empleará un modelo de conocido como "*Single-Track Model*", el cual contempla características tanto cinemáticas como dinámicas del robot. Entre estas características, la más importante para nuestro enfoque es el derrape del vehículo a alta velocidad, ya que la competición *F1TENTH* implica recorrido de circuitos a alta velocidad.

Se introduce la siguiente simbología que será utilizada más adelante:

Variable	Símbolo	Unidad
Ángulo de dirección	δ	rad
Ángulo de derrape	β	rad
Ángulo de rumbo	Ψ	rad
Velocidad del ángulo de dirección	v_δ	$\frac{rad}{s}$
Velocidad	v	$\frac{m}{s}$
Velocidad máxima sin derrape	v_S	$\frac{m}{s}$
Separación entre ruedas	l_{wb}	m
Altura del centro de gravedad	h_{cg}	m
Distancia de centro de gravedad a eje trasero	l_r	m
Distancia de centro de gravedad a eje delantero	l_f	m
Masa del vehículo	m	kg
Inercia del vehículo alrededor de z	I_z	$kg\cdot m^2$
Coeficiente de fricción	μ	-
Coeficiente de rigidez en curvas frontal/trasera	$C_{f,r}$	-
Mínimo valor posible	$\underline{\square}$	-
Máximo valor posible	$\overline{\square}$	-
Variable en dirección lateral	\square_{lat}	-
Variable en dirección longitudinal	\square_{long}	-

Inicialmente, se desarrollan algunas restricciones de movimiento del eje de dirección y de aceleración. Posteriormente, se cubre el modelo *Single-Track*.

³El modelo Punto-Masa abstrae el vehículo como un punto de masa que puede ser acelerado en un rango restringido, y que ignora el radio de giro mínimo que posee la plataforma. [1, 5].

⁴Este modelo considera las cargas verticales sobre las 4 ruedas debidas al *pitch*, *roll* y *yaw*, así como el giro, derrape y no linealidades en la dinámica de las ruedas [1, 5].

Restricciones de dirección y aceleración

Las restricciones a la velocidad del ángulo de dirección, ángulo de dirección y velocidad son sencillas y vienen dadas por 2.1:

$$\delta_S \in [\underline{\delta}_S, \bar{\delta}_S], \quad v_S \in [\underline{v}_S, \bar{v}_S], \quad v \in [\underline{v}, \bar{v}] \quad (2.1)$$

Considerando que la potencia y el frenado son limitados, tenemos 2.2:

$$a_{long} \in [\underline{a}, \bar{a}(v)], \quad \bar{a}(v) = \begin{cases} a_{max}^{\frac{v_S}{v}}, & \text{para } v > v_S \\ a_{max}, & \text{de otra manera.} \end{cases} \quad (2.2)$$

Finalmente, considere el círculo de fricción, conocido también como *Kamm's Circle*, que limita la aceleración absoluta en 2.3:

$$\sqrt{a_{long}^2 + a_{lat}^2} \leq a_{max}, \quad (a_{max} = v\dot{\Psi}) \quad (2.3)$$

Modelo Single-Track

Este modelo es también conocido como Modelo de la bicicleta. Posee una simplificación de 4 ruedas a 2 ruedas, ver Figura 2.12, colocando las 2 ruedas equivalentes a lo largo del eje central del vehículo y en las posiciones correspondientes de al eje trasero y delantero⁵. Se toma en cuenta la transferencia de carga del vehículo debida a la aceleración longitudinal a_{long} , sin contemplar la dinámica de la suspensión. Estas fuerzas verticales ejercidas en el eje delantero y trasero son denotadas como $F_{z,f}$ (Ec. 2.4) y $F_{z,r}$ (Ec. 2.5), respectivamente.

$$F_{z,f} = \frac{mg l_f - ma_{long} h_{cg}}{l_r + l_f} \quad (2.4)$$

$$F_{z,r} = \frac{mg l_r + ma_{long} h_{cg}}{l_r + l_f} \quad (2.5)$$

⁵Este modelo posee singularidades a baja velocidad, por tanto, para velocidades menores a $0.1 \frac{m}{s}$ se utiliza el modelo *Kinematic Single-Track*. Podrá encontrar dicho modelo en [1, 5].

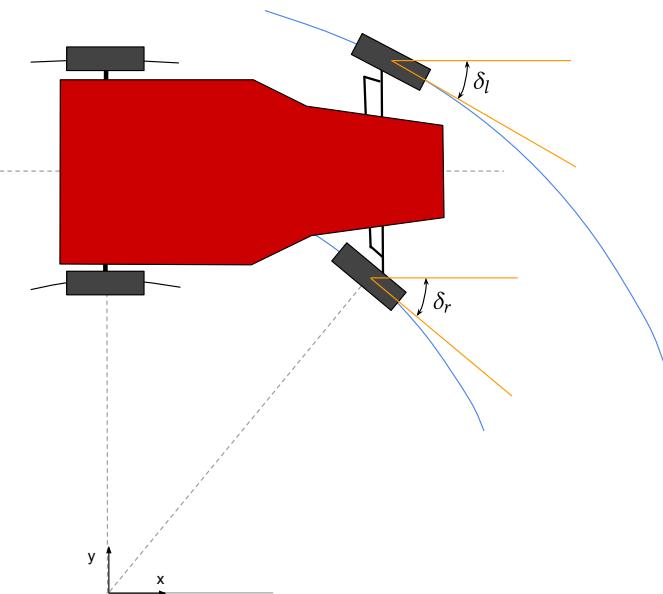


Figura 2.11: Diagrama para modelado de la configuración Ackermann sin simplificar.

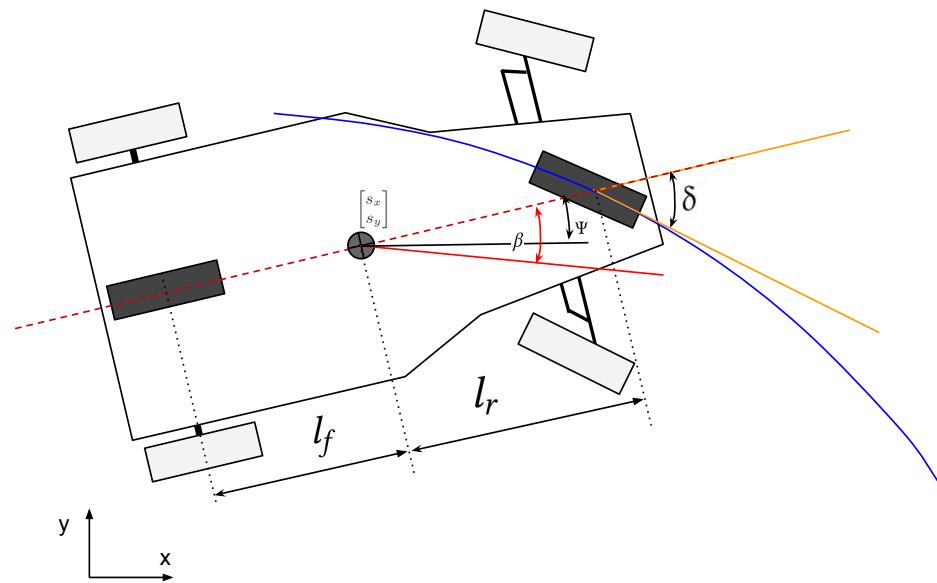


Figura 2.12: Diagrama para modelado de la configuración Ackermann simplificado al modelo de bicicleta.

Del modelo definido en [1,5] y utilizando el diagrama de la figura 2.12:

$$\dot{\delta} = v_\delta \quad (2.6)$$

$$\begin{aligned} \dot{\beta} = & \frac{\mu}{v(l_r + l_f)} (C_{S,f}(gl_r - a_{long}h_{cg})\delta - \\ & (C_{S,r}(glf + a_{long}h_{cg}) + C_{S,f}(gl_r - a_{long}h_{cg}))\beta + \\ & (C_{S,r}(glf + a_{long}h_{cg})l_r - C_{S,f}(gl_r - a_{log}h_{cg})lf) \frac{\dot{\Psi}}{v} - \Psi \quad (2.7) \end{aligned}$$

$$\begin{aligned} \ddot{\Psi} = & \frac{m\mu}{I_z(l_r + l_f)} [l_f C_{S,f}(gl_r - a_{long}h_{cg})\delta + \\ & (l_r C_{S,r}(glf + a_{long}h_{cg}) - l_f C_{S,f}(gl_r - a_{long}h_{cg}))\beta] - \\ & (l_f 2C_{S,r}(glf + a_{long}h_{cg}) + l_r 2C_{S,f}(gl_r - a_{log}h_{cg}) \frac{\dot{\Psi}}{v}) \quad (2.8) \end{aligned}$$

$$\dot{v} = a_{long} \quad (2.9)$$

$$\dot{S}_x = v \cos(\beta + \psi) \quad (2.10)$$

$$\dot{S}_y = v \sin(\beta + \psi) \quad (2.11)$$

Adicionalmente la definición del ángulo de derrape:

$$\beta = \arctan \left(\tan(\delta) \frac{l_r}{l_{wb}} \right) \quad (2.12)$$

2.5. Algoritmos de Navegación Autónoma

Los vehículos autónomos basan sus decisiones en los módulos de planificación para crear o encontrar un camino, compuesto por una serie de puntos [27], que lleven al vehículo desde la posición actual hasta una posición final deseada. Lo anterior, siguiendo una trayectoria definida y evadiendo obstáculos cuando sea necesario. La toma de decisiones en los sistemas de conducción autónoma está estructurada jerárquicamente en: planificación de ruta, decisión de comportamiento, planificación de movimiento local y control de retroalimentación [30].

2.5.1. Planificación Local y Global

En la navegación autónoma, los planificadores son los encargados de definir las acciones o decisiones que toma un robot dadas las circunstancias o situaciones que percibe de su entorno, por ejemplo objetos, localización por sensores y datos de trayectoria deseada. Generalmente, estos planificadores requieren información del entorno a priori para funcionar, sin embargo, hay algoritmos que dotan al robot de la capacidad de recorrer y recordar un mapa, mismo que luego puede ser utilizado para una planificación más efectiva y puramente autónoma.

Es normal que en la navegación de robots móviles se utilicen principalmente dos planificadores, bien conocidos como planificador global y planificador local. El planificador global es el encargado de definir un recorrido óptimo en el entorno que lo lleve de su punto inicial al destino [4], este planificador ya conoce el mapa a priori y además posee información sobre los obstáculos estáticos presentes en el mismo. El planificador local es el encargado de recalcular la ruta deseada cuando en el camino aparecen obstáculos dinámicos, o sea, obstáculos que no eran conocidos a priori por el planificador global, y tiene la tarea esencial de evadirlos para posteriormente volver a recuperar la trayectoria definida por el planificador global.

Ejemplos de planificadores globales comúnmente utilizados son: Voronoi [6], Dijkstra [13] o descomposición celular [2]; mientras los planificadores locales pueden ser: Braitenberg [9], Slines [27] o líneas de Clothoids [25].

En resumen, el planificador global se encarga de los algoritmos de seguimiento de trayectorias y el planificador local se encarga de la evasión de obstáculos. En el caso de este proyecto, el planificador global está dado por un conjunto de puntos pre calculados en el marco de coordenadas global, los cuales son la base del movimiento para el funcionamiento del algoritmo de seguimiento de trayectorias implementado.

2.5.2. Algoritmos de seguimiento de trayectorias

Algoritmo de Persecución Pura

El algoritmo de Persecución Pura fue empleado en sus inicios para calcular el arco requerido para llevar a un robot desde un punto dado hasta una trayectoria definida [11]. Este algoritmo es uno de las formas más efectivas de realizar seguimiento de trayectorias, sin embargo, su desempeño y precisión se ven limitadas por el parámetro de distancia *lookahead*. [44].

Como ya fue mencionado, el algoritmo de Persecución Pura trabaja calculando la curvatura que lleva a un vehículo desde un punto A (posición actual) hasta una posición B (posición de destino). Ahora, la tarea principal de este algoritmo es seleccionar una posición de destino que se encuentre localizada a una distancia *lookahead* de la posición actual del vehículo y encontrar un arco que une dichos puntos, este arco debe de ser determinado de manera única, ya que entre dos puntos dados existirán infinitos arcos secantes a ambos puntos. El punto localizado a la distancia *lookahead* puede ser considerado como el análogo del punto frente al vehículo al cual los seres humanos vemos al conducir a través de una carretera. [11, 17, 44]. En la figura 2.13 y 2.14 se ejemplifica el algoritmo de Persecución Pura.

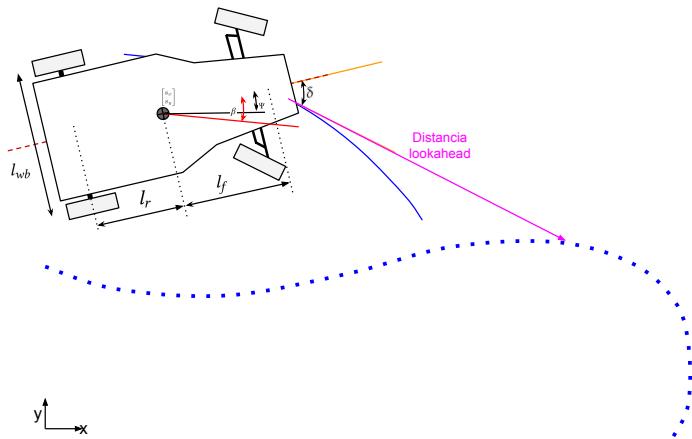


Figura 2.13: Algoritmo de Persecución Pura 1. Adaptado de [11].

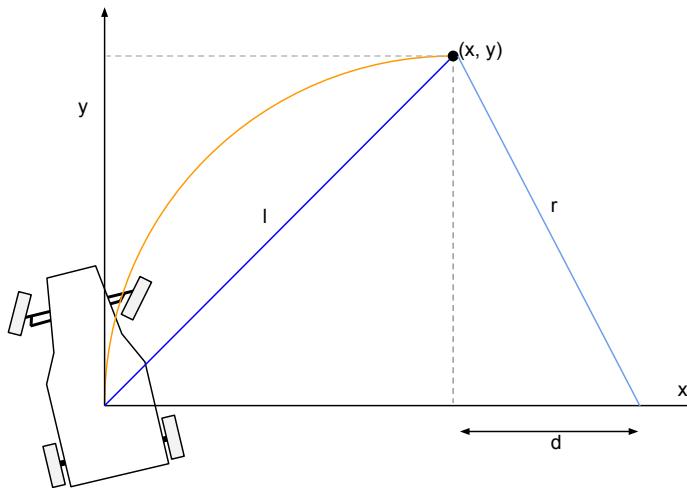


Figura 2.14: Algoritmo de Persecución Pura 2. Adaptado de [11].

Dadas las figuras 2.13 y 2.14, vemos que:

$$x^2 + y^2 = l^2 \quad (2.13)$$

$$x + d = r \quad (2.14)$$

De combinar las ecuaciones anteriores, como se detalla en [11], se obtiene el radio de giro y su inverso, la curvatura:

$$r = \frac{l^2}{2x} \implies \gamma = \frac{2x}{l^2} \quad (2.15)$$

Algoritmo: Persecución pura

Entradas: Trayectoria (pre calculada) $[X_{ref}, Y_{ref}, V_{ref}]$, postura actual (Odometría)

Salidas: δ, v

Parámetros: l (figura 2.14), $P_{v_{gain}}$

Mientras No se envíe una señal de parada

 Obtener postura $(x_{actual}, y_{actual}, \theta_{actual})$.

 Seleccionar un punto de la trayectoria pre calculada:

 Leer puntos de trayectoria.

 Calcular al punto más cercano y su índice correspondiente en el arreglo de puntos.

 Calcular radio de curvatura de arco de intersección según 2.15.

 Calcular actuación para cumplir con el arco que pasa por el punto deseado:

 Calcular δ según 2.12.

 Calcular v con $P_{v_{gain}}$.

 Enviar comando a los actuadores.

fin

Tabla 2.1: Pseudo código algoritmo de Persecución Pura

2.5.3. Algoritmos de evasión de obstáculos

Algoritmo de Braitenberg

Este es un algoritmo de evasión de obstáculos que permite al robot ejecutar comandos que se ven determinados de manera proporcional a la lectura de sensores. Esto hace que el comportamiento del robot sea considerado como “reactivo” [38], ya que reacciona ante el entorno que le rodea, por lo que, si un estímulo en el entorno captado por los sensores desaparece, la actuación vinculada a dicho estímulo también desaparecerá. Braitenberg plantea dos tipos de vehículos en su libro. El primero consta de una única rueda y un único sensor, motivo por el cual solo se moverá en una dirección, este vehículo se ejemplifica en la figura 2.15.

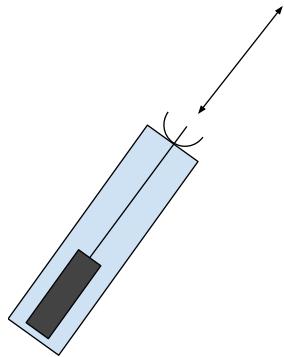


Figura 2.15: Vehículo de Braitenberg 1.

El segundo tipo de vehículos de Braitenberg es más funcional y consta de 2 sensores, derecho e izquierdo, e igualmente, de dos ruedas en ambas posiciones, respectivamente. Lo anterior permite al robot realizar giros y evasión de obstáculos de manera efectiva dependiendo de las limitaciones físicas y cinemáticas que posee. Este segundo tipo de vehículo permite representar con mayor claridad la proporcionalidad mencionada anteriormente entre mediciones y actuación, ya que las mediciones del sensor derecho e izquierdo afectarán los comandos de velocidad aplicados a las ruedas, dependiendo del comportamiento que se deseé replicar: Miedo o agresividad [9]. Este comportamiento de Braitenberg 2a (Miedo) y 2b (Agresividad) se ejemplifica en la figura 2.16.

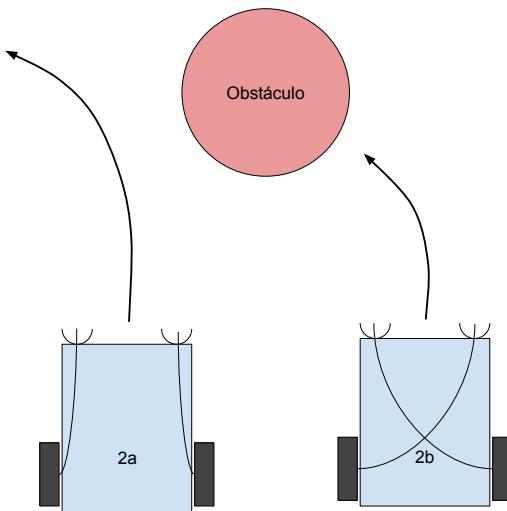


Figura 2.16: Vehículo de Braitenberg 2a (Miedo) y 2b (Agresividad). Adaptada de [9]

Como se observa en la figura 2.16, el vehículo 2a (Miedo) se aleja, mientras que, el vehículo 2b

(Agresividad) se dirige hacia el obstáculo. Cabe mencionar que dichos comportamientos parten de la conexión directa o cruzada entre sensores y ruedas mostrada en la imagen.

Algoritmo: Braitenberg Modificado: Miedo

Entradas: Lecturas de sensor LiDAR 360°

Salidas: $\Delta\delta$

Parámetros: d_{umbral} (figura 4.5)

Mientras No se envíe una señal de parada **o** se colisione

 Obtener lecturas LiDAR.

 Computar los datos leídos:

 Cantidad de segmentos y división, según 4.1.

 Distancias promedios d_{prom0} y d_{prom1} , según 4.2. Tal que: $\frac{n}{2} - 1$ es el segmento derecho y $\frac{n}{2}$ el izquierdo.

Si $d_{prom1} < d_{umbral}$, **Entonces:**

$$\delta = \delta - \Delta\delta$$

O si $d_{prom0} \leq d_{umbral}$, **Entonces:**

$$\delta = \delta + \Delta\delta$$

Si no, entonces:

$$\delta = 0$$

 Aplicar comando obtenido en el paso anterior a las ruedas de dirección

fin

Tabla 2.2: Pseudo código algoritmo de Braitenberg Modificado: Miedo

Algoritmo Time Elastic Band (TEB)

El algoritmo clásico de la banda elástica se basa en encontrar una ruta global para la distancia más corta entre el robot y el objetivo. No obstante, el TEB optimiza la ruta mediante una función objetivo de tiempo óptimo y admite robots no holonómicos⁶ similares a los automóviles [32]. Además, mientras que la banda elástica tiene en cuenta las restricciones dinámicas, el TEB considera las restricciones kino-dinámicas⁷ en la planificación de la trayectoria.

Es decir, el planificador de TEB es principalmente una solución a un problema de optimización multiobjetivo escalarizado y escaso. Esto incluye limitaciones como la velocidad máxima, la aceleración máxima, el radio de giro mínimo, entre otros [20]. Usualmente, como no tiene una única solución, se puede quedar atascado en algunos puntos óptimos en donde, el robot a veces no puede pasar un obstáculo incluso aunque ya exista una posible trayectoria que el robot pueda seguir, esto se corrigió con una versión mejorada del TEB, donde se optimiza una trayectoria global de forma paralela al problema

⁶Son vehículos que tienen restricción de movimiento en alguno de sus ejes. Los vehículos Ackermann son no holonómicos, mientras los Mecanum son holonómicos.

⁷Se refiere a restricciones de movimiento cinemático y dinámico.

de optimización multi objetivo que ya resuelve, siempre y cuando sea necesario cambiar a esta versión mejorada [29, 37]. Existe una biblioteca de ROS, que es parte del [navigation stack](#), en donde se implementa el algoritmo de [TEB](#). Si bien las herramientas de navigation stack están enfocadas a robots diferenciales y holonómicos, se encuentran disponibles adaptaciones que consideran las restricciones cinemáticas de una plataforma Ackermann. Dicha adaptación a plataformas *car-like* se aborda en [35].

Dynamic Window Approach (DWA)

Es un método que se utiliza generalmente en robots móviles. Se trata de un algoritmo que sirve para la planificación de trayectorias, el cual provee un controlador entre el robot y la trayectoria global. Este enfoque pondera y calcula las diferentes entradas de control y de percepción, como, por ejemplo, si el robot evade los obstáculos, la distancia que existe del objetivo al robot, entre otras, y con ese resultado traza la ruta más adecuada [20].

El principio de funcionamiento de este enfoque se resume de la siguiente manera:

- Se toman muestras de dx , dy y $d\theta$ del espacio de control.
- Se predicen los siguientes estados a partir de los estados actuales basándose en las muestras de dx , dy y $d\theta$.
- Se califica cada trayectoria de las predicciones con la distancia a los obstáculos, la distancia al objetivo, distancia a la ruta global y velocidad y se eliminan las trayectorias que chocan con los obstáculos.
- Se elige la trayectoria con la puntuación más alta y se envía la trayectoria al robot.
- Se repite hasta que alcance la meta.

Para utilizar este enfoque de una manera adecuada, se deben configurar correctamente los parámetros de la configuración del robot, la tolerancia de objetivos, el avance de simulación, la puntuación de la trayectoria, plan global y el tiempo de la simulación. Este último, es el más importante de todos, ya que va a ir afectando en gran medida el tiempo de cálculo de la trayectoria [29, 47].

Existe una biblioteca de ROS, que es parte del [navigation stack](#), en donde se implementa el algoritmo de [DWA](#). Si bien las herramientas de navigation stack están enfocadas a robot diferenciales y holonómicos, se encuentran disponibles adaptaciones que consideran las restricciones cinemáticas de una plataforma Ackermann, en [33] se aborda una adaptación de DWA para configuraciones Ackermann.

Capítulo 3

HERRAMIENTAS UTILIZADAS

Este corto capítulo estará dedicado a las herramientas de software utilizadas en la implementación de este proyecto y es de carácter meramente informativo. Se aborda a grandes rasgos el funcionamiento y utilización de ROS y del simulador RViz en la sección 3.3, además de una descripción del entorno desarrollado por la organización y proyecto de código abierto *F1TENTH*. En la sección 3.4 se muestra el funcionamiento del entorno y como iniciar la ejecución del simulador para posteriormente poder correr los programas que implementan las funcionalidades de navegación autónoma que se describen en el capítulo 2, se implementan en el capítulo 4 y se simulan en el capítulo 5.

3.1. Robot Operating System (ROS)

ROS, en inglés “[Robotic Operating System](#)”, es un meta sistema operativo de código abierto utilizando como plataforma de desarrollo para la escritura de software especializado para robots. ROS ofrece gran flexibilidad a la hora de desarrollar, ya que permite la integración entre varios lenguajes de programación gracias a la estructura de funcionamiento que posee. Se le llama meta sistema operativo, o *middleware*, ya que no es un sistema operativo en el sentido tradicional de gestión y programación de procesos, sino que más bien proporciona una capa de comunicaciones estructurada por encima de los sistemas operativos host de un clúster de cómputo heterogéneo [15, 31].

Este útil meta sistema operativo proporciona los servicios que normalmente un sistema operativo proporciona al usuario, tales como la abstracción de hardware, implementación de funciones de uso común, transmisión de mensajes entre procesos, control de dispositivos a bajo nivel y administración de paquetes; también proporciona bibliotecas para obtener, construir, escribir y ejecutar código en múltiples computadoras [15]. Por lo anterior, ROS es categorizado por sus desarrolladores como un SDK (*Software Development Kit*) que provee bloques que permiten construir aplicaciones de robótica, desde un proyecto sencillo de clase hasta realizar investigación científica. Se observa en la figura 3.1 su nivel de composición a nivel gráfico.

ROS funciona al nivel de un sistema de archivos compuesto por paquetes, meta-paquetes, manifiesto de paquetes, repositorios, tipos de mensajes y tipos de servicios. A nivel gráfico, su funcionamiento se resumen en la comunicación mediante nodos: descritos mediante la biblioteca de cliente de ROS,



Figura 3.1: Composición gráfica de ROS. Tomada de [ROS, ecosystem](#)

maestro: proporciona el registro de nodos y control de interacción, servidor de parámetros: almacenamiento de datos central que forma parte del maestro, mensajes: manera en que los nodos se comunican entre sí, tópicos: espacio en el cual se publican mensajes y a los cuales los nodos pueden suscribirse para acceder a la información ahí disponible, servicios: servicios que usan un modelo solicitud/respuesta, y finalmente, bolsa: formato que permite guardar y reproducir datos de mensajes de ROS.

Existe toda una [comunidad](#) alrededor de ROS a través de la cual se comparten recursos como [software](#) y conocimiento y en donde también se almacenan recursos en distribuciones de ROS, repositorios y un [ROS Wiki](#). En la figura 3.2 podrá encontrar las [distribuciones](#) de ROS más recientes desde el 2016, de las cuales, la que se emplea en este proyecto es [Noetic](#).

Distro	Release date	Poster	Tuturtle, turtle in tutorial	EOL date
ROS Noetic Ninjemy (Recommended)	May 23rd, 2020			May, 2025 (Focal EOL)
ROS Melodic Morenia	May 23rd, 2018			May, 2023 (Bionic EOL)
ROS Lunar Loggerhead	May 23rd, 2017			May, 2019
ROS Kinetic Kame	May 23rd, 2016			April, 2021 (Xenial EOL)

Figura 3.2: Distribuciones de ROS recientes

3.2. RViz

El nombre de esta herramienta del entorno de ROS viene de “*ROS visualization*”. Es utilizada para la visualización de robots, sensores y prueba de algoritmos. RViz utiliza la data adquirida por los sensores (reales o simulados) para crear de la manera más precisa posible una visualización del estado del robot dentro de un ambiente ya sea real o simulado. En la Figura 3.3 se muestra la pantalla principal del entorno.

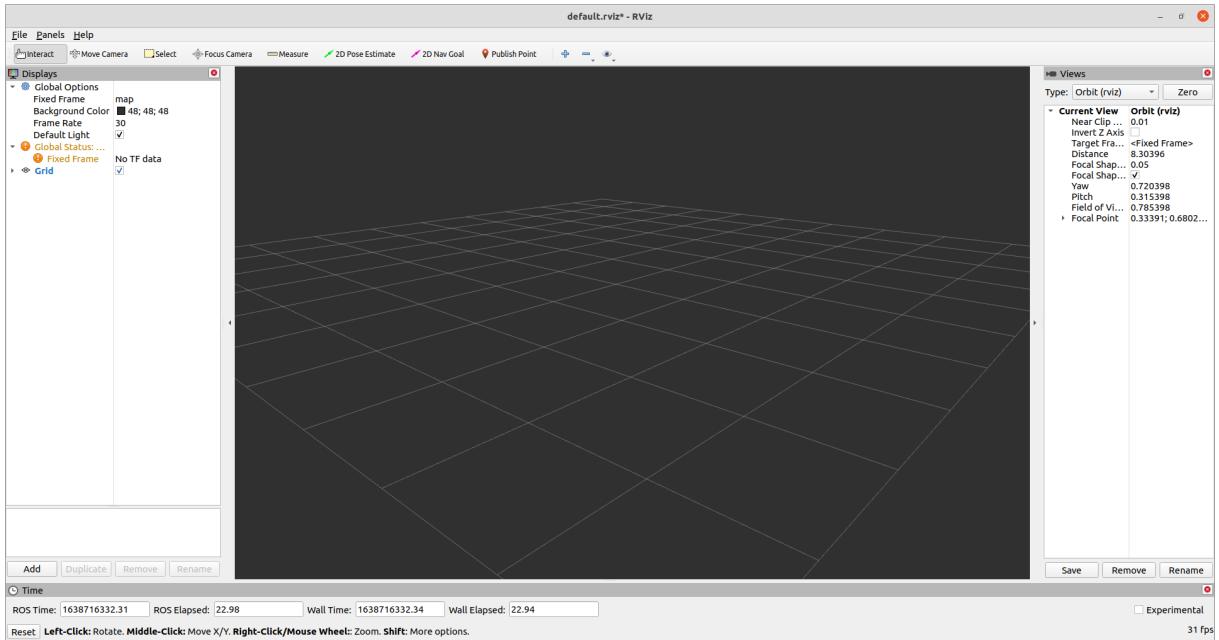


Figura 3.3: Interfaz gráfica de RViz

3.3. Integración ROS, RViz y simulador *F1TENTH*

La comunidad del *F1TENTH* ha desarrollado varios entornos de simulación basados en la utilización de RViz para facilitar la simulación del robot que se utiliza en su competencia y el proyecto de código abierto que desarrollan. En el caso de este proyecto se utiliza el [entorno de simulación](#) disponible dentro del [repositorio oficial de *F1TENTH*](#), en donde además se pueden encontrar herramientas útiles para el desarrollo de la plataforma en configuración Ackermann, como por ejemplo, una serie de [circuitos de competencia](#) que funcionan como mapas dentro de los entornos que desarrollan, lo cual permite probar una amplia gama de recorridos, este repositorio además tiene la ventaja de que para cada circuito de competencia brinda un archivo en donde se pueden encontrar líneas de carrera con un trazado que puede ser utilizado como insumo para alimentar a un planeador global que realice seguimiento de trayectorias. Dentro de las mismas herramientas desarrolladas por la comunidad *F1TENTH* se encuentra el repositorio [F1TENTH SVLSimulator](#), el cual se recomienda revisar para futuros proyectos e investiga-

ciones, ya que es un simulador altamente depurado y que se integra con otras herramientas sofisticadas de visualización y manejo físico/gráfico.

Como recurso adicional se recomienda consultar [F1TENTHBuild](#), en donde está disponible una wiki completa para iniciar con la construcción, configuración, instalación y simulación del proyecto de la comunidad *F1TENTH*. Se debe mencionar que la edición de los mapas presentados en este proyecto eléctrico se realizó bajo la premisa de agregar obstáculos estáticos que no varíen entre una iteración y otra, para así poder garantizar una mayor uniformidad en el análisis de los resultados obtenidos.

3.4. Preparación y funcionamiento del entorno de simulación

Primeramente, se debe mencionar que la computadora donde se ejecutaron todas las simulaciones presentadas en este proyecto cuenta con 16 Gb de RAM, procesador Intel® Core™ i7-7700HQ CPU @ 2.80GHz × 8, tarjeta gráfica NVIDIA de 4 Gb y Mesa Intel® HD Graphics 630 (KBL GT2). El sistema operativo empleado es Ubuntu 20.04.3 LTS. En la figura 3.4 se muestra el [entorno de simulación](#) utilizado para el desarrollo de este proyecto, junto con el circuito de competencia y además un modelo sencillo del robot Ackermann. Se puede observar una serie de puntos que varían entre rojo y amarillo en los bordes de la pista y de los obstáculos, estos puntos son la representación visual de las mediciones tomadas por el sensor LiDAR que se simula como parte de los componentes del robot dentro de software. En la figura 3.5 se muestran los nodos utilizados por el entorno de la *F1TETNH* para lograr el funcionamiento de las simulaciones que se muestran en este proyecto.

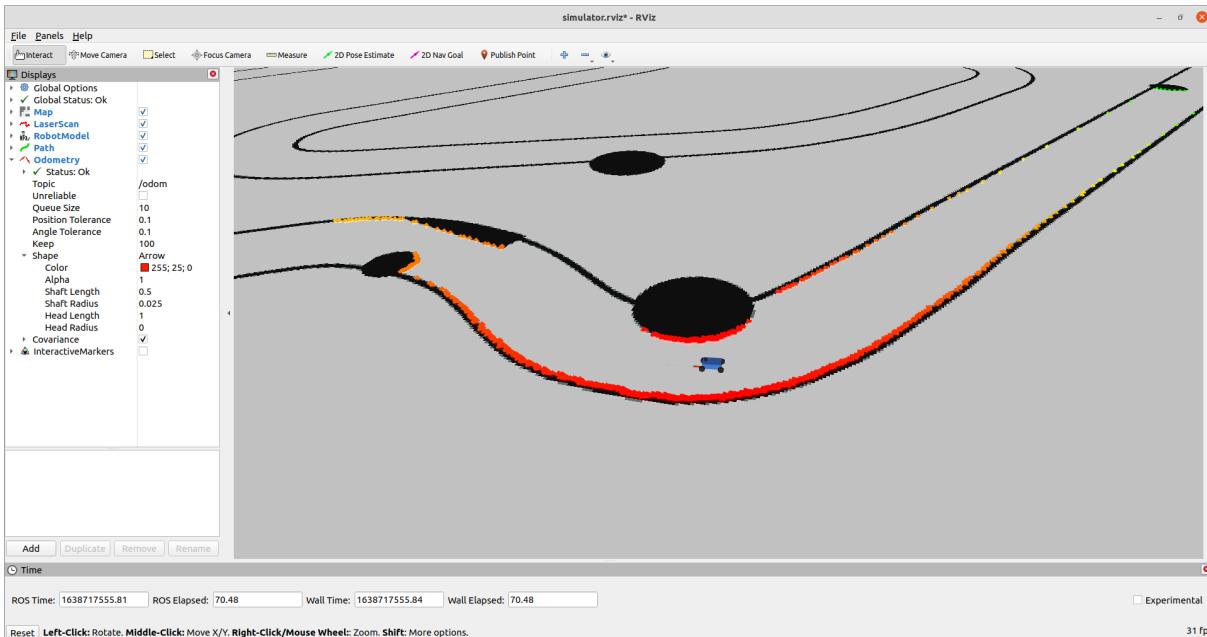


Figura 3.4: Visualización del entorno de simulación

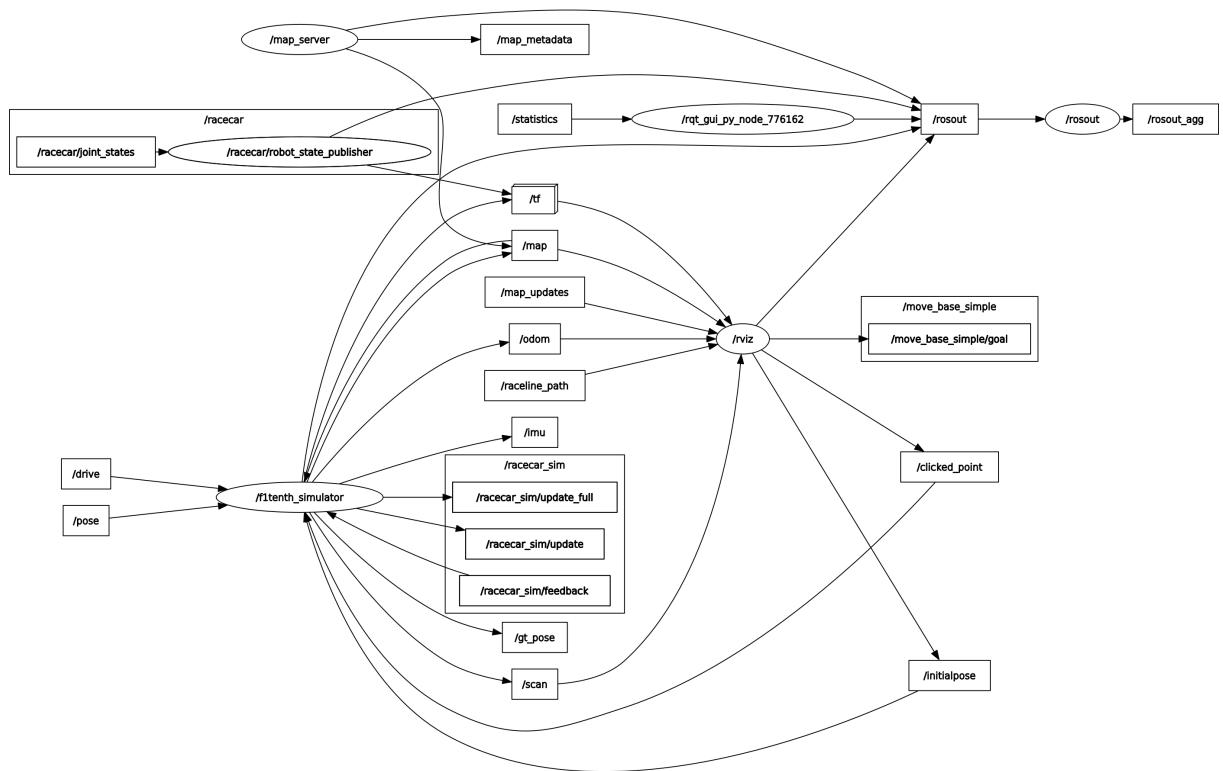


Figura 3.5: Árbol de nodos del entorno de simulación

En el repositorio correspondiente a este entorno podrá encontrar [documentación](#) detallada de cómo se constituye y configura el simulador. Está disponible una descripción de sus dependencias, inicio rápido, visualización con RViz, ROS API, instrucciones para agregar nuevos nodos de planeación, tópicos, marcos de referencia, parámetros de simulación, parámetros del vehículo y más información respecto a su funcionamiento y configuración.

Capítulo 4

ALGORITMOS IMPLEMENTADOS

En esta sección se abarcan a detalle los algoritmos implementados para este proyecto eléctrico: Seguimiento de trayectorias por Persecución Pura y evasión de obstáculos por algoritmo de Braitenberg, Adicionalmente se describe la implementación de mezcla de comportamientos, la cual combina mediante ponderación los comandos de salida de los dos algoritmos mencionados y utiliza los mismos como planificador global y local, correspondientemente.

4.1. Algoritmo de seguimiento de trayectoria

4.1.1. Persecución Pura

Como ya se abordó en la sección 2.5.2, este algoritmo de seguimiento de trayectoria calcula el ángulo requerido por el sistema de dirección para que el robot apunte desde su posición actual hasta algún punto de vista hacia el frente del robot a una distancia l (*lookahead*), mostrada en la figura 4.2; Dicha distancia es el parámetro más influyente sobre la respuesta del robot al seguimiento de una trayectoria y a la separación que exista entre los puntos que la constituyen, como veremos en los siguientes capítulos. Lo anterior se puede encontrar descrito gráficamente en la figura 4.2 En el caso de esta implementación se cuenta con una línea de carrera previamente computada y optimizada para el trazado de competencia de varias pistas de carrera, reducidas en la escala 1/10, estas líneas de carrera se encuentran disponibles en el repositorio: [fitenth_racetracks](#). Además, para referencia de futuros proyectos eléctricos, se recomienda visitar los repositorios: [TUM - Institute of Automotive Technology](#), [MotionPlanning](#) y [fitenth_motion_planning](#), ahí se pueden encontrar recursos e implementaciones útiles para futuras investigaciones.

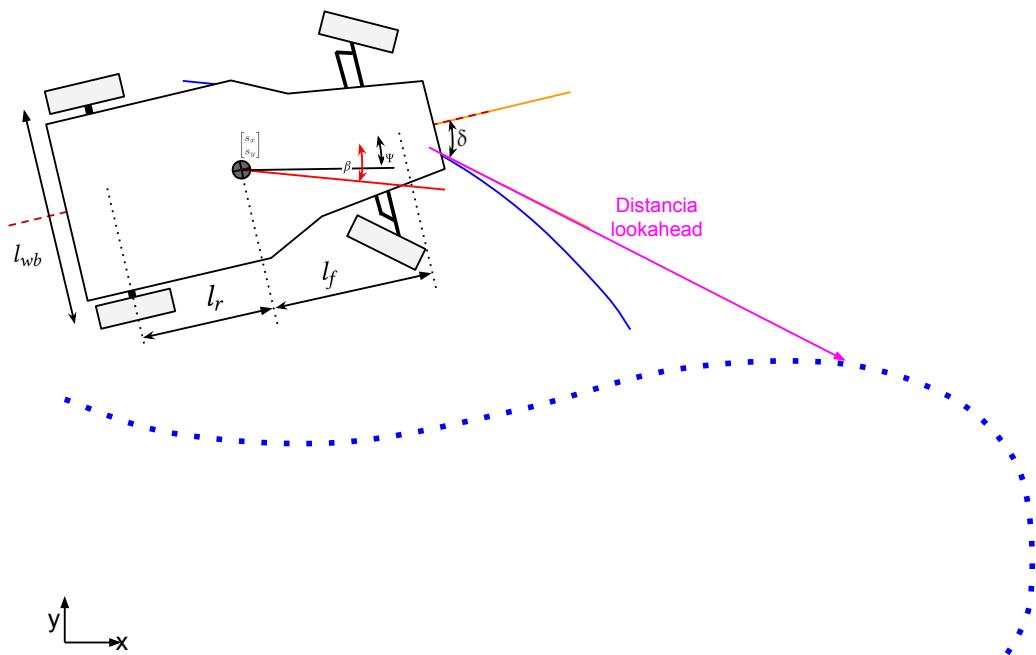


Figura 4.1: Diagrama de algoritmo de Persecución Pura.

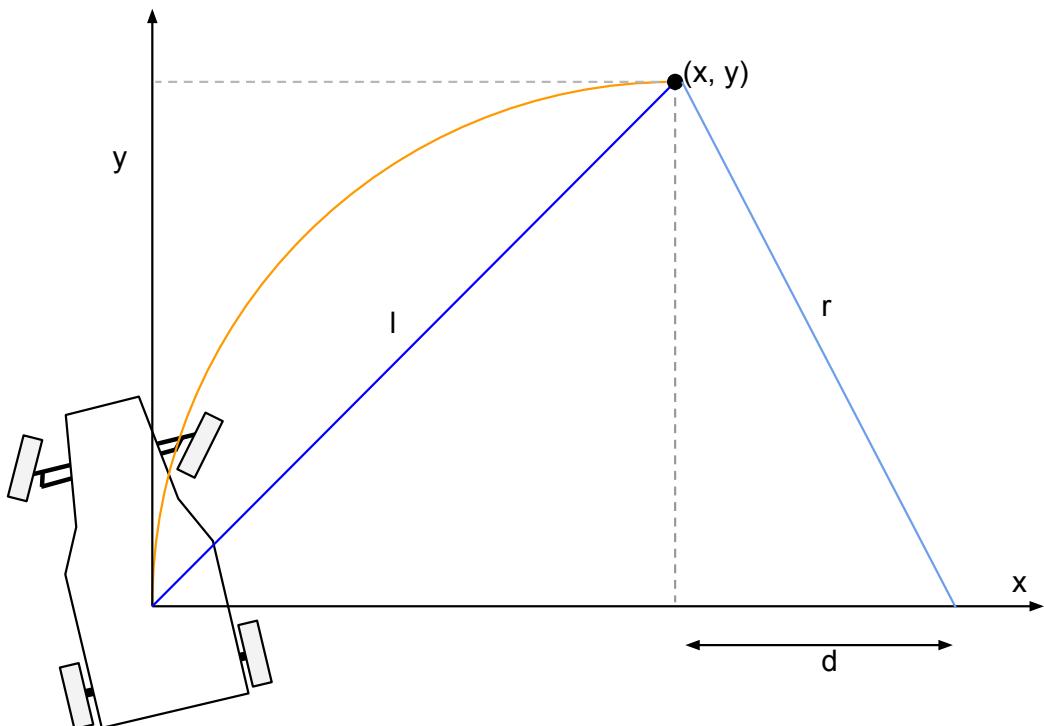


Figura 4.2: Diagrama de algoritmo de Persecución Pura. Adaptado de [11].

El pseudocódigo simplificado de la implementación que se realizó se encuentra en la tabla 4.1. El algoritmo puede entenderse en mayor detalle en los pasos descritos a continuación:

1. Determinar la posición actual del vehículo. Obtención de la postura: (x, y, θ) mediante la odometría con respecto al marco de referencia global, o mediante algún método avanzado de localización.
2. Encontrar el punto de la trayectoria que sea más próximo al vehículo y que esté al menos a una distancia *lookahead* del mismo.
3. Encontrar la posición del punto deseado respecto a la posición actual dentro del marco de referencia global.
4. Transformar el punto deseado al marco de referencia del vehículo.
5. Calcular la curvatura y asignar el ángulo de dirección requerido a las ruedas de dirección del vehículo para cumplir con la curvatura requerida.

En cuanto a la Persecución Pura implementada y descrita, es importante mencionar que es sencilla, sin embargo, en ella radica una complejidad dada en el cálculo de la curvatura que conecta al punto actual con el punto objetivo y que se ve determinado por la distancia *lookahead*, ya que es posible que esta solución no sea única. Por otra parte, como lo describen [11] y [1], este algoritmo no toma en consideración los efectos dinámicos sobre el vehículo, ya que no modela los actuadores del mismo y, por tanto, asume que se da una respuesta perfecta ante la curvatura requerida. En adición a lo anterior, tampoco modela el derrape ni en baja ni alta velocidad variables que si son consideradas en el modelo interno del simulador: [f1tenths_simulator](#), en el cual se implementa el modelo dinámico *Single-Track Model* (ST) descrito en [5].

Algoritmo: Persecución pura

Entradas: Trayectoria (pre calculada) $[X_{ref}, Y_{ref}, V_{ref}]$, postura actual (Odometría)

Salidas: δ, v

Parámetros: l (figura 2.14), $P_{v_{gain}}$

Mientras No se envíe una señal de parada

 Obtener postura ($x_{actual}, y_{actual}, \theta_{actual}$).

 Seleccionar un punto de la trayectoria pre calculada:

 Leer puntos de trayectoria.

 Calcular al punto más cercano y su índice correspondiente en el arreglo de puntos.

 Calcular radio de curvatura de arco de intersección según 2.15.

 Calcular actuación para cumplir con el arco que pasa por el punto deseado:

 Calcular δ según 2.12.

 Calcular v con $P_{v_{gain}}$.

 Enviar comando a los actuadores.

fin

Tabla 4.1: Pseudo código algoritmo de Persecución Pura

La implementación de Persecución Pura realizada se basa en las ecuaciones descritas en la sección 2.5.2. La tabla 4.2 muestra un resumen del código auxiliar que contiene funciones requeridas para el funcionamiento del nodo principal. La tabla 4.3 muestra nodo principal implementado. Como se observa en la figura 4.3 el nodo de Persecución Pura se suscribe al tópico /odom, de donde toma la pose del robot calculada por el simulador respecto al marco de referencia global, y a su vez publica los comandos requeridos en el tópico /drive para ser enviados posteriormente al nodo de simulación /f1tenthsimulator, donde se actualiza la visualización y el entorno virtual.

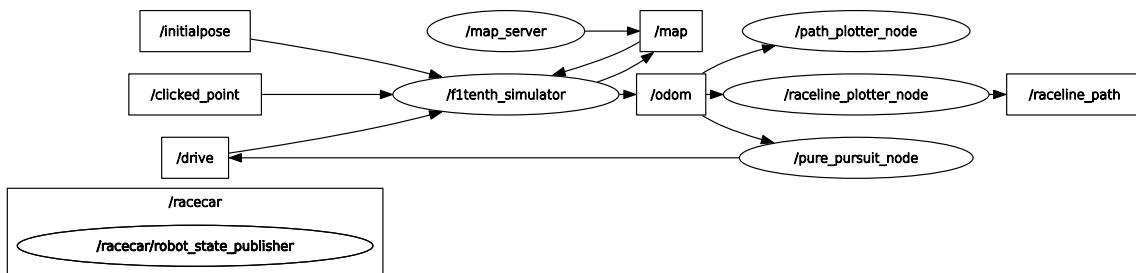


Figura 4.3: Gráfico obtenido con rqt_graph para el algoritmo de Persecución Pura.

A continuación se describen con brevedad los códigos en lenguaje de alto nivel, Python, generados y adaptados para Persecución Pura, con la finalidad de que puedan ser entendidos, modificados y utilizados en futuras implementaciones, proyectos eléctricos e investigaciones.

Nombre del archivo: pp_utils.py

Descripción: Este archivo contiene funciones útiles para la implementación del algoritmo de Persecución Pura. Se importa al código del nodo principal.

Función: `load_waypoints(waypoints_file)`: Importa los pares ordenados que componen la trayectoria a seguir por el algoritmo, dichos puntos se encuentran almacenados en un archivo waypoints_file.csv. **Retorna:** Arreglo con todos los pares ordenados de forma que puedan ser procesados posteriormente.

Función: `get_current_waypoint(waypoints, lookahead_distance, pose)`: Toma el arreglo waypoints y lo procesa utilizando el lookahead_distance y la pose para determinar el índice del arreglo en la cual se encuentra el elemento más cercano entre la posición actual y la trayectoria deseada. **Retorna:** El punto encontrado, si este existe, si no, retorna *None*.

Función: `get_actuation(pose_theta, lookahead_point, position, lookahead_distance, wheelbase)`: Se encarga de generar los comandos de velocidad y de dirección del vehículo para alcanzar el punto de destino. **Retorna:** Los comandos de velocidad v (*speed*) y dirección δ (*heading*).

Tabla 4.2: Implementación del algoritmo de Persecución Pura, pseudo descripción del nodo auxiliar.

Nombre del archivo: pure_pursuit.py

Descripción: Este archivo contiene la clase con el nodo principal encargado de manejar el algoritmo de Persecución Pura. Adquiere la odometría del simulador desde el tópico /odom y realiza planeamiento para el movimiento del robot y publica sus comandos en /drive. Acá se implementan las funciones descritas por 4.2

Función: `odom_callback(odom_msg)`: Recibe los mensajes de pose (x, y, θ) del robot publicados por el simulador en el tópico /odom y ejecuta la función `plan(pose, lookahead_distance, trajectory)` en cada lectura de dicho tópico. **Retorna:** 0.

Función: `plan(pose, lookahead_distance, trajectory)`: Esta función se encarga del planeamiento de trayectoria utilizando la pose actual, revisando la trayectoria a seguir y definiendo el punto al cual se irá, además de generar comandos de actuación. **Retorna:** la velocidad y el ángulo de dirección que posteriormente será enviado al tópico /drive para comandar las ruedas y dirección.

Tabla 4.3: Implementación del algoritmo de Persecución Pura, pseudo descripción del nodo principal.

4.2. Algoritmo de evasión de obstáculos

4.2.1. Vehículo de Braitenberg: Miedo

Al igual que para la Persecución Pura, en la sección 2.5.3 se puede encontrar detalle adicional del algoritmo de Braitenberg. En el caso de este proyecto se trabajó sobre la base de la versión 2a (Miedo) y se

le realizaron modificaciones que se abordarán más adelante en este capítulo. Además, en el capítulo 5 se pone en evidencia y analiza el efecto de dichas modificaciones.

Como se observa en la figura 4.4, el robot posee dos sensores capaces de registrar el entorno frente a ellos, sin embargo, en el caso de la implementación de este proyecto, se cuenta con un sensor LiDAR, el cual brinda información de los obstáculos de entorno que rodea al robot *FITENTH* mediante una nube de puntos. Cada dato en la nube de puntos se representa como la distancia r desde el sensor hasta un objeto o pared para en ángulo dado; se dispone de visión de 360° con 1080 rayos de medición, por tanto, cada segmento de medición corresponde con un ángulo de 0.33° .

Para efectos de la implementación realizada en este proyecto eléctrico, tanto para Braitenberg como para la mezcla de comportamientos, cuyos resultados se abordan en el capítulo 5, se ha decidido partir en n segmentos iguales el campo de visión de 360° (Ec.4.1) y calcular el promedio de las mediciones interiores de cada segmento para obtener un valor único asociado al mismo (Ec.4.2):

$$\text{grados de segmento}_n = 360/n \quad (4.1)$$

$$\text{distancia prom}_n = \frac{\sum \text{distancia de mediciones}_n}{\text{cantidad de mediciones}_n} \quad (4.2)$$

Lo anterior, para facilitar el manejo de los datos, y además, de entender con base a experimentación el efecto de esta segmentación sobre el desempeño del algoritmo y el comportamiento del vehículo. En la figura 4.4 se ejemplifica la diferencia entre el algoritmo original y la implementación que se realizó.

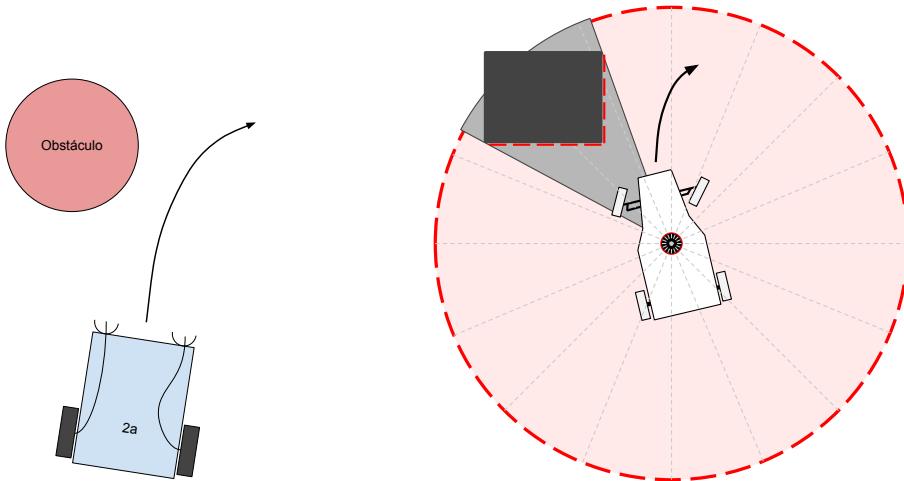


Figura 4.4: Vehículo de Braitenberg 2a (derecha) vs. Implementación (izquierda). [9]

Dada la distinción anteriormente mencionada, la implementación que se realizó se denominó como Braitenberg Modificado: Miedo. El pseudocódigo simplificado de la implementación realizada se encuentra en la tabla 4.4. El algoritmo puede describirse en mayor detalle en los pasos mostrados a continuación:

1. Leer los datos provenientes del sensor LiDAR.

2. Procesar los datos obtenidos del entorno de acuerdo a las ecuaciones 4.1 y 4.2. Esto implica segmentar en n partes el rango de visión. Es necesario que n sea un número divisible entre 2.
3. Asignar los segmentos de lectura que tendrán efecto sobre el accionar de la plataforma. En este caso se opta por seleccionar únicamente los dos segmentos en el frente del robot, de tal manera que uno de estos promedios representa la distancia a un objeto a la derecha y otro a la izquierda.
4. Determinar si el valor promedio del segmento derecho o izquierdo es menor al umbral de detección de obstáculo d_{umbral} .
5. Determinar la acción reactiva a aplicar a las ruedas de dirección. La velocidad aplicada a las ruedas traseras se mantiene constante y no cambia durante la ejecución del algoritmo.
6. Aplicar la acción de control determinada en el paso anterior.

En cuanto a la implementación de Braitenberg Modificada: Miedo, es importante recalcar que es bastante sencilla, sin embargo, como estamos hablando de un algoritmo de evasión completamente reactivo, existe algo de complejidad dada por el ajuste del parámetro de umbral de detección de obstáculo d_{umbral} , y además, la velocidad lineal de vehículo. Lo anterior es dado que las restricciones cinemáticas y dinámicas del robot móvil en configuración Ackermann suponen un radio de curvatura máximo para la evasión efectiva junto con la condición de derrape en alta velocidad. Como este es un algoritmo de evasión de obstáculos reactivo, como se observa en la figura 4.5, no toma en consideración ningún modelado cinemático ni actuación de del robot, únicamente se limita a determinar la acción necesaria para evadir de acuerdo a la lectura de sus sensores.

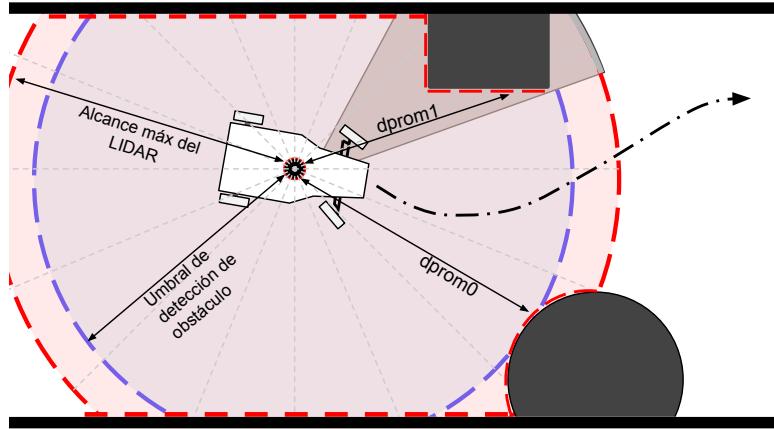


Figura 4.5: Diagrama de Braitenberg Modificado: Miedo implementado.

Algoritmo: Braitenberg Modificado: Miedo**Entradas:** Lecturas de sensor LiDAR 360°**Salidas:** $\Delta\delta$ **Parámetros:** d_{umbral} (figura 4.5)**Mientras** No se envíe una señal de parada o se colisione

Obtener lecturas LiDAR.

Computar los datos leídos:

Cantidad de segmentos y división, según 4.1.

 Distancias promedios d_{prom0} y d_{prom1} , según 4.2. Tal que: $\frac{n}{2} - 1$ es el segmento derecho y $\frac{n}{2}$ el izquierdo. **Si** $d_{prom1} < d_{umbral}$, **Entonces**:

$$\delta = \delta - \Delta\delta$$

O si $d_{prom0} \leq d_{umbral}$, **Entonces**:

$$\delta = \delta + \Delta\delta$$

Si no, entonces:

$$\delta = 0$$

Aplicar comando obtenido en el paso anterior a las ruedas de dirección

fin

Tabla 4.4: Pseudo código algoritmo de Braitenberg Modificado: Miedo

En la tabla 4.5 se describe el nodo implementado para Braitenberg Modificado: Miedo. Como se aprecia en la figura 4.6 el nodo de braitenberg_modificado se suscribe al tópico /scan, de donde toma las lecturas del LiDAR respecto al marco de referencia centrado en la posición del sensor, y a su vez publica los comandos de dirección δ requeridos en el tópico /drive para ser enviados posteriormente al nodo de simulación /f1tenths_simulator, donde se actualiza la visualización y el entorno virtual.

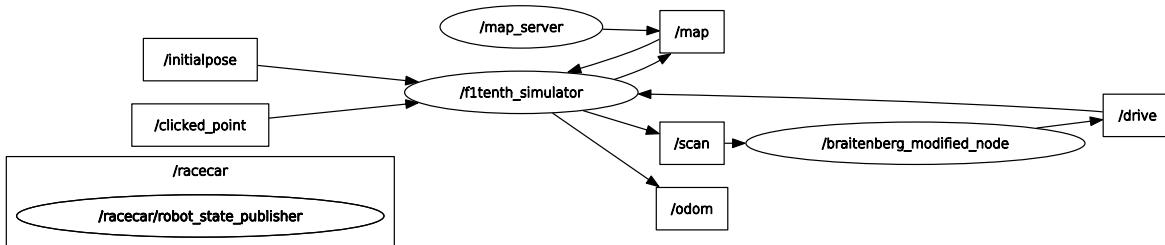


Figura 4.6: Gráfico obtenido con rqt_graph para el algoritmo de Braitenberg Modificado: Miedo.

A continuación se describe con brevedad el código en lenguaje de alto nivel, Python, generado y adaptado para Braitenberg Modificado: Miedo, con la finalidad de que pueda ser entendido, modificado y utilizado en futuras implementaciones, proyectos eléctricos y e investigaciones.

Nombre del archivo: braitenberg_modified.py

Descripción: Este archivo contiene el nodo que se utilizó para la implementación del algoritmo de Braitenberg Modificado: Miedo.

Función: `scan_callback(scan_msg)`: Obtiene los datos del sensor LiDAR proveniente del tópico de /scan, los procesa para tomar acción y además envía los comandos requeridos al tópico /drive en cada lectura realizada. **Retorna:** 0.

Tabla 4.5: Implementación del algoritmo de Braitenberg Modificado: Miedo, pseudo descripción del nodo.

4.3. Mezcla de comportamientos: Seguimiento de trayectorias + Evasión de obstáculos en alta velocidad

Esta sección aborda la implementación de la mezcla de comportamientos necesaria para cumplir el objetivo de las competencias organizadas por el proyecto *F1TENTH*: Lograr completar recorridos dentro de circuitos de carreras, con presencia de obstáculos estáticos, mediante la fusión del planeamiento global y local en condiciones de alta velocidad.

En este caso, los dos algoritmos que se fusionaron son los explicados en las secciones 4.1.1: Persecución Pura y 4.2.1: Braitenberg Modificado: Miedo. Como ya se comentó en dichas secciones, cada algoritmo publica sus mensajes de velocidad lineal y los comandos de dirección para el eje delantero en los tópicos /drive. Para realizar la fusión de estos comandos, se tomaron las salidas de ambos algoritmos y se enviaron cada una al tópico que le corresponde: /local_drive o /global_drive, posteriormente se realizó una ponderación de dichas salidas, para velocidad y dirección, en el nodo PP_Brait_modified_nav_node con los parámetros constantes: GSW^1 , LSW^2 , $GSTRW^3$ y $LSTRW^4$ y se publicó en el tópico /drive del simulador *F1TENTH*, de tal forma que la velocidad y el comando de dirección están dadas por las ecuaciones 4.3 y 4.4, respectivamente:

$$v_{weighted} = GSW * v_{global} + LSW * v_{local} \quad (4.3)$$

$$\delta_{weighted} = GSTRW * \delta_{global} + LSTRW * \delta_{local} \quad (4.4)$$

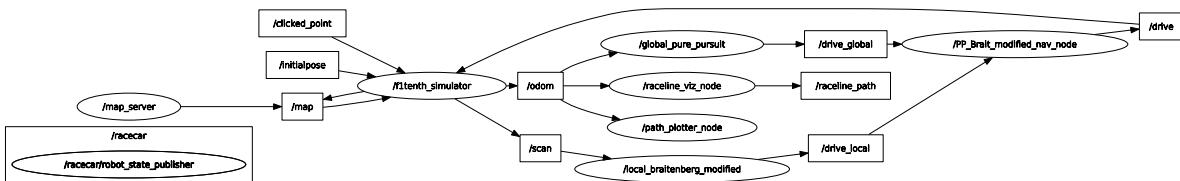


Figura 4.7: Gráfico obtenido con rqt_graph para la mezcla Persecución Pura-Braitenberg Modificado

¹ GSW : Global Speed Weight $\in [0, 1]$.

² LSW : Local Speed Weight $\in [0, 1]$.

³ $GSTRW$: Global Steering Weight $\in [0, 1]$.

⁴ $LSTRW$: Local Steering Weight $\in [0, 1]$.

Nombre del archivo: PP_Brait_modified_nav_node.py

Descripción: Este archivo contiene el nodo de la implementación de mezcla de comportamientos para PP+Brait Modificado: Miedo. La tarea principal que desempeña es la de ponderar los comandos provenientes del planificador global y local. Los valores GSW, LSW, GSTRW y LSTRW son inicializados desde un archivo de configuración y cargados al servidor de parámetros de ROS, de ahí se toman y utilizan en la ponderación.

Función: `global_drive_callback(g_msg)`: Esta función es la encargada de adquirir los mensajes del tópico /drive_global. Dentro de esta función se ejecuta el drive_handler para cada mensaje proveniente
Retorna: 0.

Función: `local_drive_callback(l_msg)`: Esta función es la encargada de adquirir los mensajes del tópico /drive_local. **Retorna:** 0.

Función: `drive_handler()`: En esta función se realiza la ponderación de los mensajes provenientes del planificador global y local utilizando las ecuaciones 4.3 y 4.4. Se encarga de publicar los comandos ponderados en el tópico /drive. **Retorna:** 0.

Tabla 4.6: Implementación de mezcla de comportamientos PP + Braitenberg modificado, pseudo descripción del nodo.

Algoritmo: mezcla de comportamientos PP + Braitenberg modificado

Entradas: Comandos provenientes del tópico /drive_local y /drive_global

Salidas: δ y v

Parámetros: GSW, LSW, GSTRW y LSTRW

Mientras No se envíe una señal de parada **o** se colisione

 Obtener lecturas datos de tópico /drive_local.

 Obtener lecturas datos de tópico /drive_glocal.

 Ponderar comandos:

 Velocidad resultante según Ec. 4.3.

 Dirección resultante según Ec. 4.4.

 Publicar los comandos ponderados en el paso anterior en el tópico /drive.

fin

Tabla 4.7: Pseudo código algoritmo de PP + Braitenberg modificado

Capítulo 5

SIMULACIÓN DE ALGORITMOS IMPLEMENTADOS

5.1. Pruebas iniciales

Esta sección muestra algunas pruebas iniciales que se realizaron en el entorno de simulación, para comprobar el funcionamiento básico de los algoritmos que se implementaron y se describen en el capítulo 4, con la intención de brindar una primera impresión del comportamiento de los mismos, representando el movimiento del robot en el plano cartesiano XY. En la tabla 5.1 se muestra un breve resumen de los parámetros utilizados para la simulación de las pruebas iniciales realizadas.

Algoritmo	Parámetros utilizados en pruebas iniciales
Persecución Pura 1	$lookahead = 1, 2, 5$ con $P_{vgain} = 0,75$
Persecución Pura 2	$lookahead = 2$ con $P_{vgain} = 0,5, 1, 1,5$ y $2,5$
Braitenberg Modificado: Miedo	$Zonas = 12$; $d_{umbral} = 2,75$

Tabla 5.1: Tabla resumen de parámetros utilizados en pruebas iniciales.

5.1.1. Seguimiento de trayectorias por Persecución Pura.

En las figuras 5.1 y 5.2 se muestra el punto de partida del robot en la coordenada $(5, 0)$ con orientación paralela al eje x , a una distancia de 5 m de una línea recta que se extiende sobre el eje $y = 0$ y que define el conjunto de puntos a seguir. Como se observa en la figura 5.1, la prueba se ejecutó para 4 valores distintos del parámetro $lookahead$ y manteniendo la velocidad constante en 4 m/s , mientras que en 5.2 la prueba se ejecutó para distintas ganancias del parámetro de velocidad P_{vgain} con una distancia $lookahead = 2 \text{ m}$.

Partiendo de un análisis gráfico de la figura 5.1, se puede interpretar que para valores pequeños, $lookahead = 1 \text{ m}$, el robot no logra tomar la trayectoria deseada, mientras que para un valor aproximadamente un orden de magnitud mayor, el seguimiento de la trayectoria tarda más tiempo para converger a un intervalo en donde el robot no esté sobre oscilando sobre la misma. Note que respecto a un $lookahead = 2 \text{ m}$, el seguimiento toma menos tiempo, sin embargo, implica mayor oscilación en un espacio

temporal menor y esta condición debe considerarse al trabajar bajo limitaciones físicas en la actuación y comportamientos dinámicos de un sistema real.

En el caso de la figura 5.2, el incremento en el valor del parámetro $P_{v_{gain}}$, ocasiona que se sobrepasen los límites físicos del vehículo generando derrapes que serán difíciles de controlar en un circuito con curvas. Dado lo anterior, este parámetro debe seleccionarse con cuidado en la implementación para evitar que por exceso de velocidad se afecte el desempeño del algoritmo de Persecución Pura, ya que, como fue mencionado en capítulos anteriores, este no toma en cuenta el modelado dinámico del sistema.

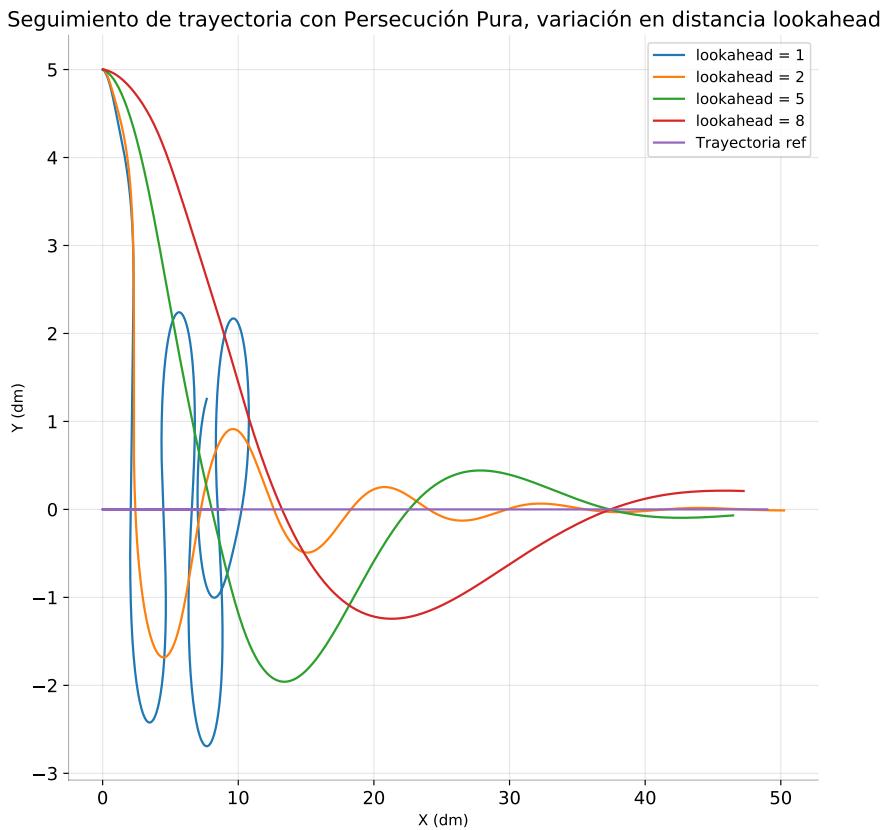


Figura 5.1: Prueba inicial para Persecución Pura, *lookahead* variable y velocidad constante $v = 4 \text{ m/s}$

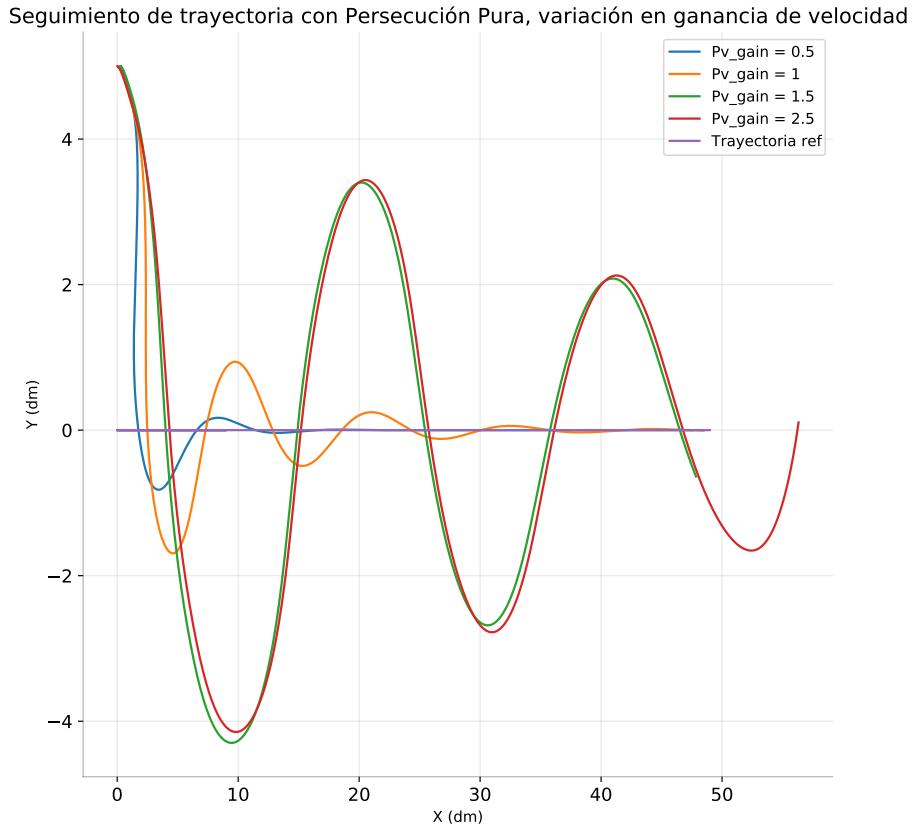


Figura 5.2: Prueba inicial para Persecución Pura, $lookahead = 2 \text{ m}$ y velocidad = $4 \text{ m/s} * P_{v_{gain}}$

5.1.2. Evasión de obstáculos sencilla - Braitenberg modificado: Miedo

La intención de esta sección es demostrar que se puede lograr la evasión efectiva de obstáculos bajo la premisa de la selección correcta de los parámetros d_{umbral} y la cantidad n de zonas o segmentos en los cuales se divide el rango de visión. Esta prueba se realizó con 12 zonas y distancia de umbral de 2,75 m.

Arena de pruebas con obstáculos estáticos

Como se puede observar en las figuras 5.3, 5.4 y 5.5, bajo las condiciones impuestas por el mapa y la selección de parámetros, se logró la evasión de los obstáculos presentes en el camino del robot para la navegación reactiva con el algoritmo de Braitenberg Modificado: Miedo.

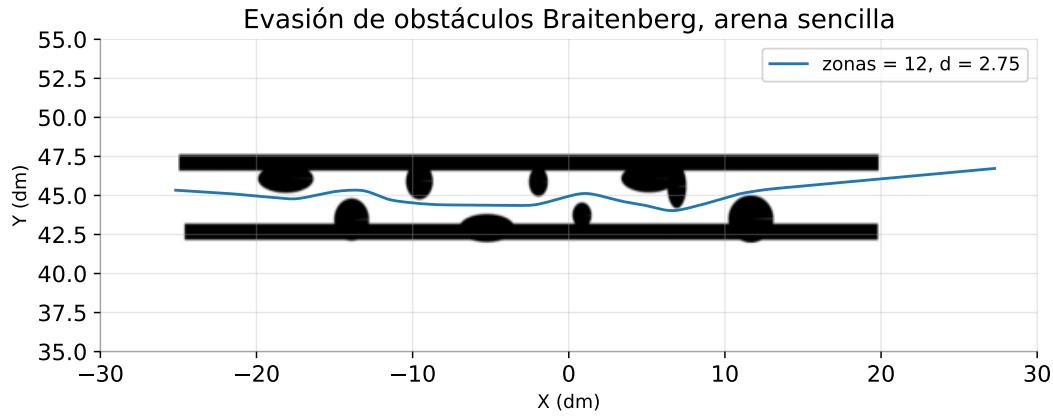


Figura 5.3: Prueba inicial de algoritmo de Braitenberg Modificado: Miedo

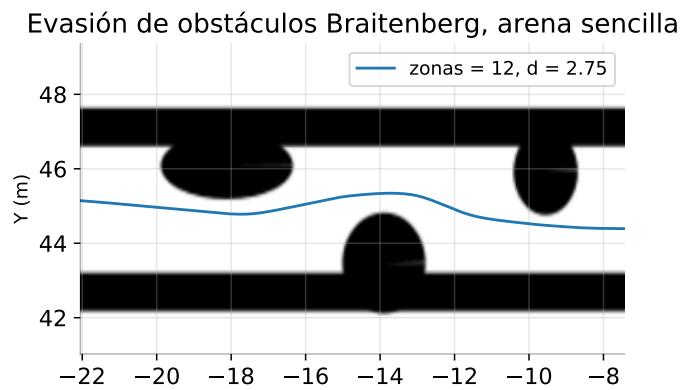


Figura 5.4: Prueba inicial de algoritmo de Braitenberg Modificado: Miedo. Ampliación 1.

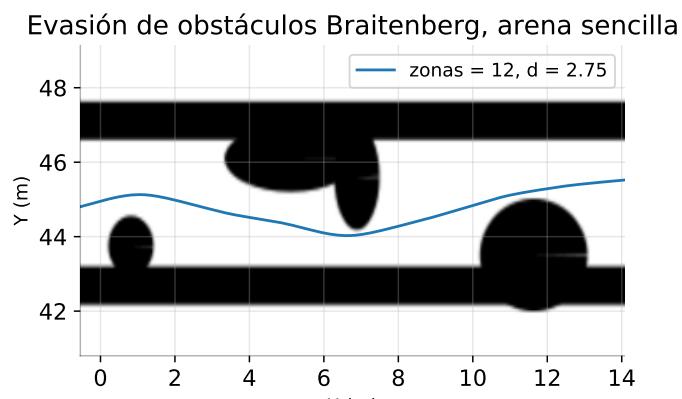


Figura 5.5: Prueba inicial de algoritmo de Braitenberg Modificado: Miedo. Ampliación 2.

5.2. Seguimiento de trayectorias en alta velocidad

Esta sección se enfoca en las pruebas que se realizaron en el entorno de simulación para demostrar el funcionamiento del algoritmo de seguimiento de trayectorias conocido como "Persecución Pura", descrito en la sección 4.1.1 y la tabla 4.1. Se realizaron simulaciones de dicha implementación para dos circuitos distintos, circuito de pruebas 1 y circuito de pruebas 2: Catalunya. El primer circuito representa un recorrido cerrado relativamente sencillo mientras que el segundo es un circuito oficial de competiciones de Fórmula 1 ubicado en España y que está conformado de un trazado más complejo; este último recorrido está escalado 1/10 para los propósitos del proyecto. Las gráficas que se mostrarán representan el movimiento del robot en el plano cartesiano horizontal XY.

5.2.1. Implementación de Persecución Pura

Basándose en la comprensión que se desarrolló sobre el efecto que la variación de los parámetros tienen sobre el comportamiento del algoritmo de Persecución Pura, gracias a las pruebas iniciales que se mostraron en la sección 5.1, se seleccionaron los parámetros $lookahead = 2$ y $P_{vgain} = 0,75$ para realizar las simulaciones y demostrar el funcionamiento de la implementación de dicho algoritmo de navegación autónoma. La selección de parámetros se realizó de esa manera dado que una combinación óptima de los mismos queda fuera del enfoque de esta proyecto, además de que supondría variación en los mismos dependiendo de la configuración del circuito donde se compite.

Circuito de pruebas 1

Esta sección muestra el resultado que se logró para la selección de parámetros en el caso del circuito de pruebas 1; $lookahead = 2$ y $P_{vgain} = 0,75$.

Como se puede observar en la figura 5.6, se graficaron dos líneas, la azul representa la trayectoria seguida por el algoritmo mientras que la naranja representa la trayectoria de referencia dada por el trazado pre computado. Se puede observar en la figura 5.7 que al iniciar la simulación el vehículo no se encontraba alineado con el primer punto de la trayectoria de referencia, sin embargo, logró aproximarse efectivamente al trazado deseado.

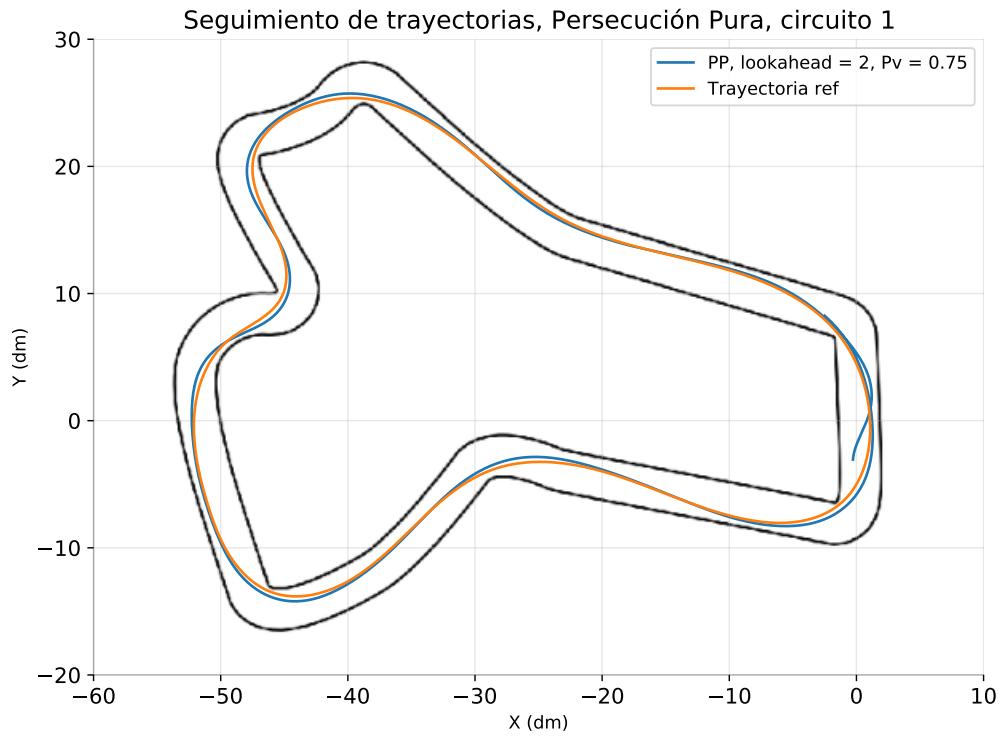


Figura 5.6: Simulación de Persecución Pura para circuito 1

Se debe de tomar en consideración que estos recorridos se realizan en alta velocidad, por lo que el efecto dinámico sobre el vehículo se ve potenciado tal y como se muestra en la figura 5.8, donde se observa que el vehículo tiende a ser empujado hacia afuera de la curva por el efecto dinámico de derrape antes mencionado.

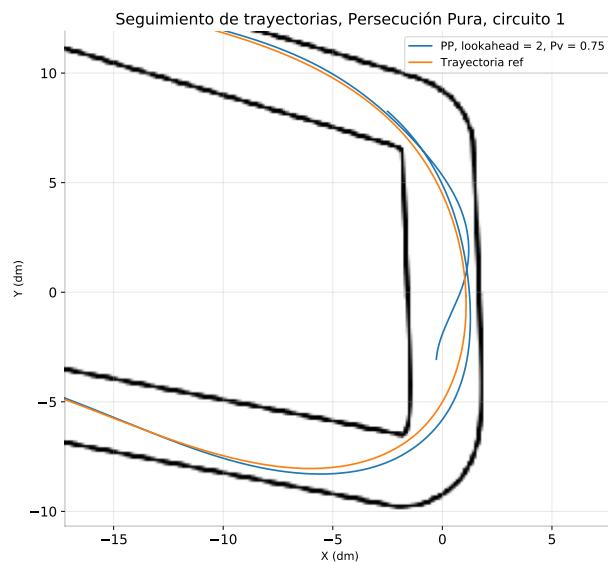


Figura 5.7: Simulación de Persecución Pura para circuito 1. Ampliación 1.

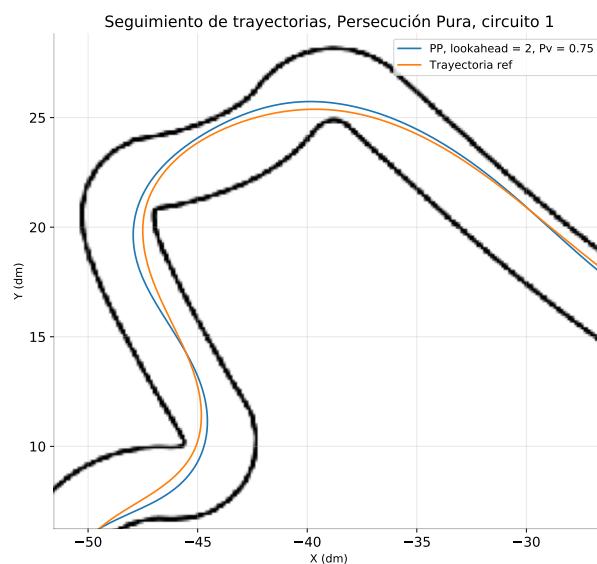


Figura 5.8: Simulación de Persecución Pura para circuito 1. Ampliación 2.

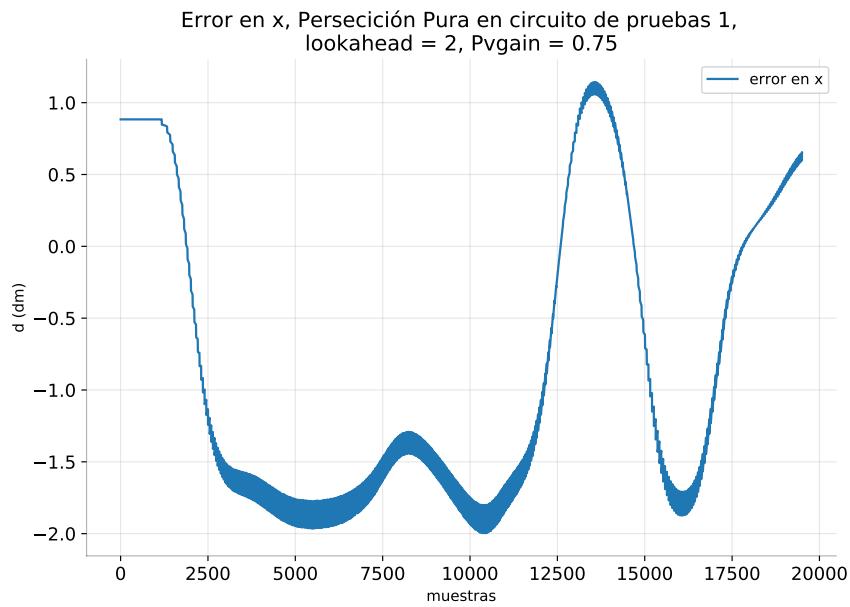


Figura 5.9: Error en distancia para x, Persecución Pura para circuito 1.

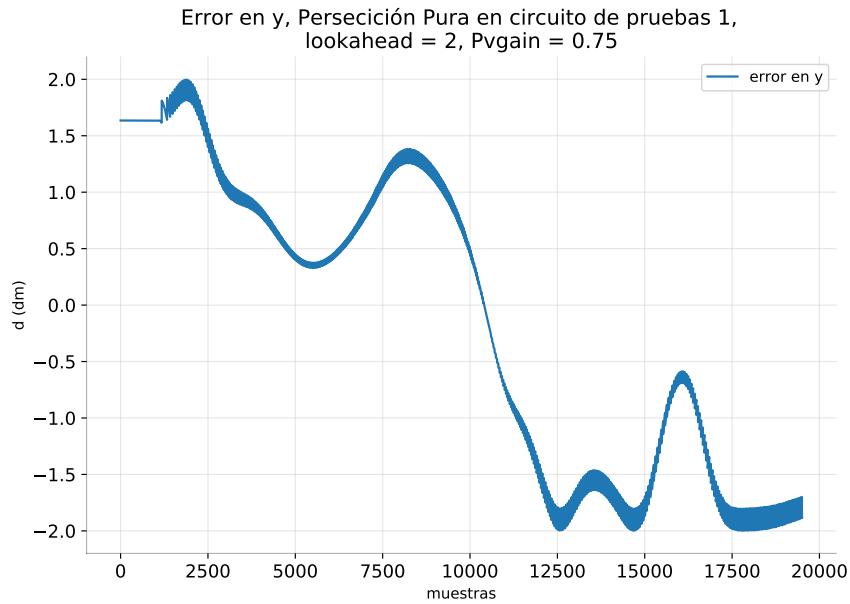


Figura 5.10: Error en distancia para y, Persecución Pura para circuito 1.

Círculo de pruebas 2: Catalunya

Estas simulaciones muestran los resultados que se lograron con la selección de parámetros en el caso del circuito de pruebas 2: Catalunya; con parámetros $lookahead = 2$ y $P_{vgain} = 0,75$.

Como se puede observar en la figura 5.11, se graficaron dos líneas, la azul representa la trayectoria seguida por el algoritmo mientras que la naranja representa la trayectoria de referencia dada por el trazado pre computado. En las figuras 5.12 y 5.13 se muestra el comportamiento del vehículo en las curvas del circuito.

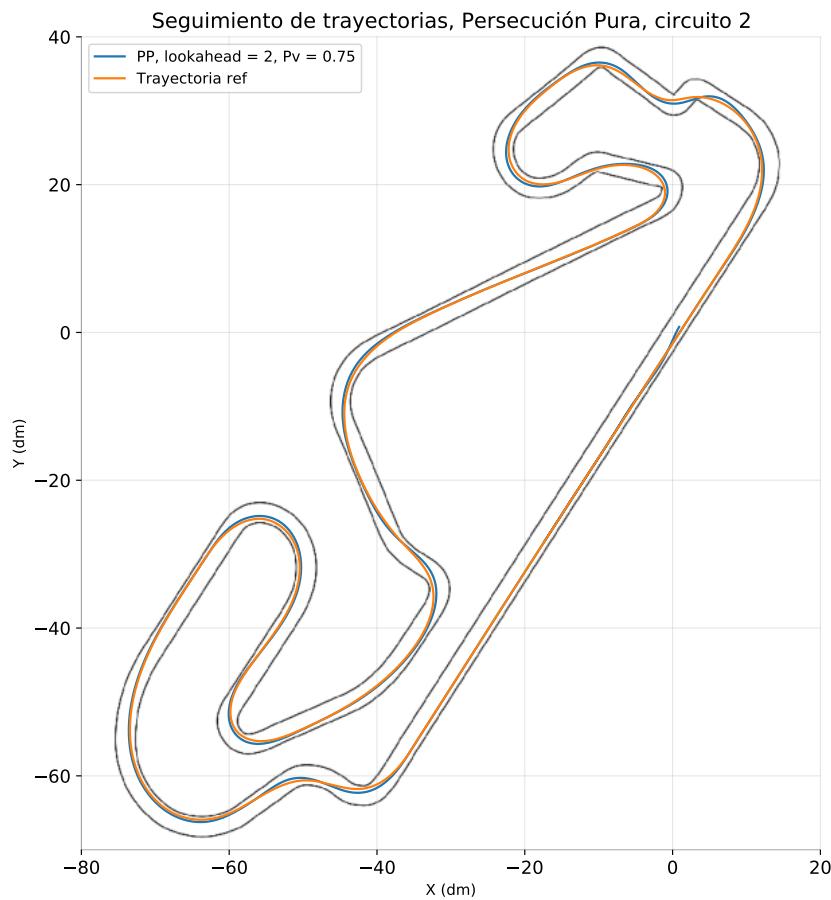


Figura 5.11: Simulación de Persecución Pura para circuito 2: Catalunya

Note que, en el caso de la imagen 5.13, hay curvas complicadas de tomar a alta velocidad, sin embargo, el vehículo logró tomarlas aún mostrando derrape inherente al comportamiento de vehículo debido

al modelo dinámico del mismo implementado en el simulador. El comportamiento dinámico inherente a las características físicas de un vehículo hace que controlar su movimiento lateral o derrape en altas velocidades sea complicado, ya que un modelo que incluya dicha dinámica debe de contemplar las fuerzas ejercidas por las ruedas sobre la carretera, coeficientes de fricción y rodadura entre ruedas y carretera, presión y deformación de las llantas, aceleraciones angulares elevadas en las curvas, centro de masa del vehículo así como su altura, inercias, peralte, entre otras. Un modelo que incluya estos detalles no solo es complicado de obtener, sino que además implica un sistema de control que sea capaz de manejar todas las entradas que esto supone para generar los comandos de velocidad y dirección para controlar el vehículo de la mejor manera posible. Como ya se ha mencionado anteriormente, el modelo implementado a nivel interno en el simulador utilizado, es el descrito en la sección 2.4.1 y las publicaciones [1], [5].

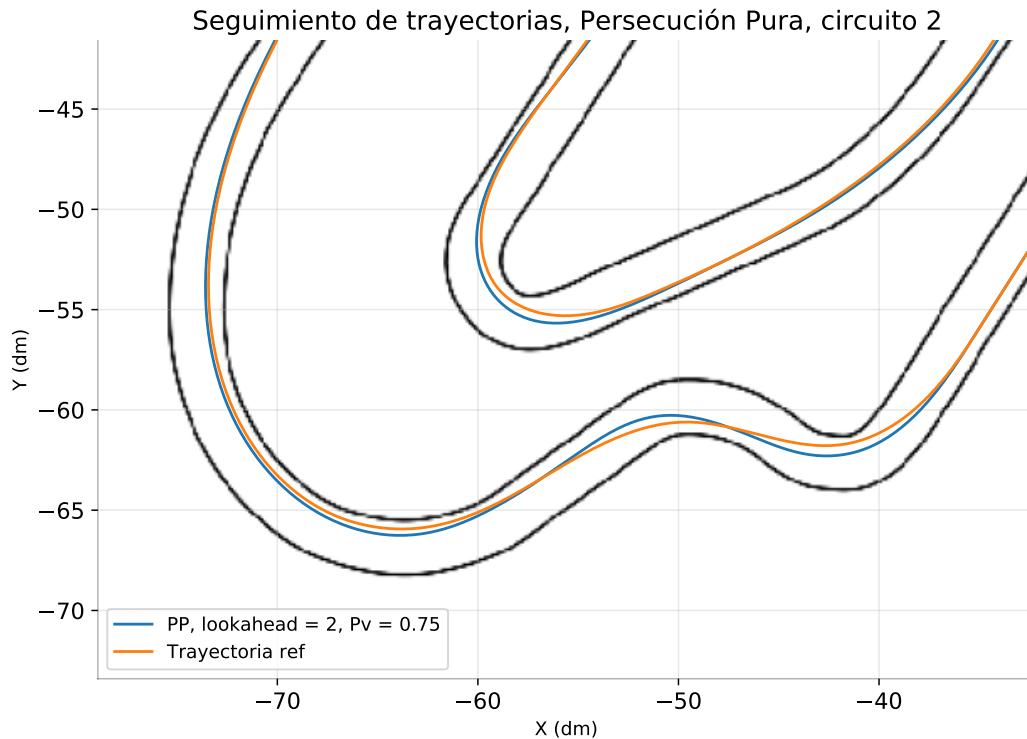


Figura 5.12: Simulación de Persecución Pura para circuito 2: Catalunya. Ampliación 1

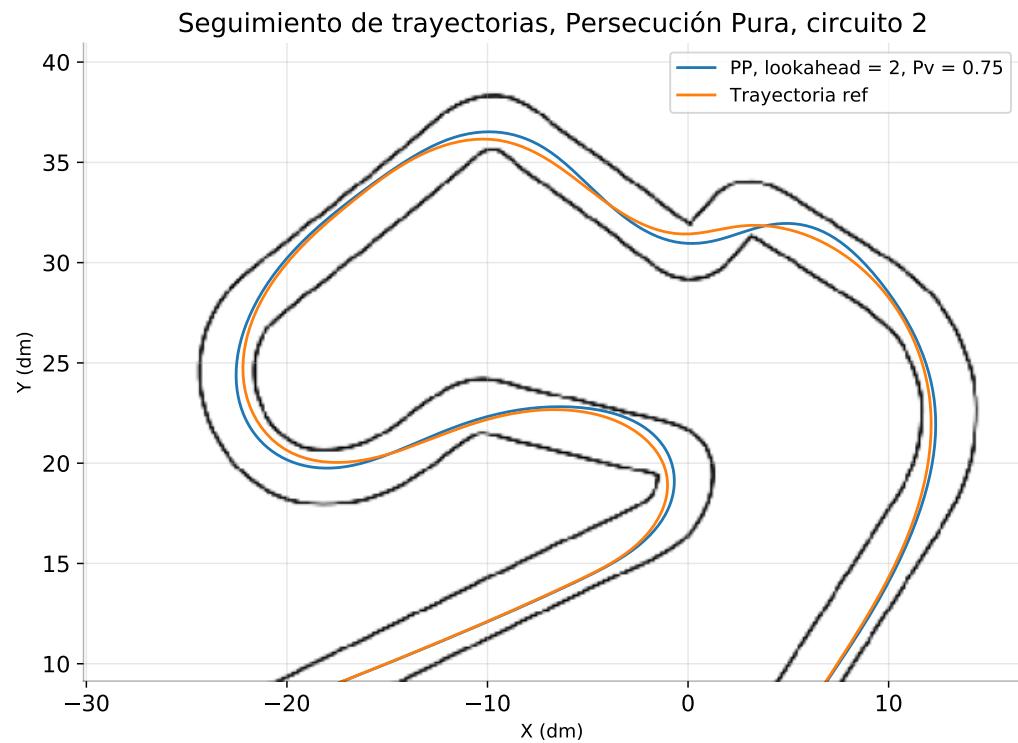


Figura 5.13: Simulación de Persecución Pura para circuito 2: Catalunya. Ampliación 2

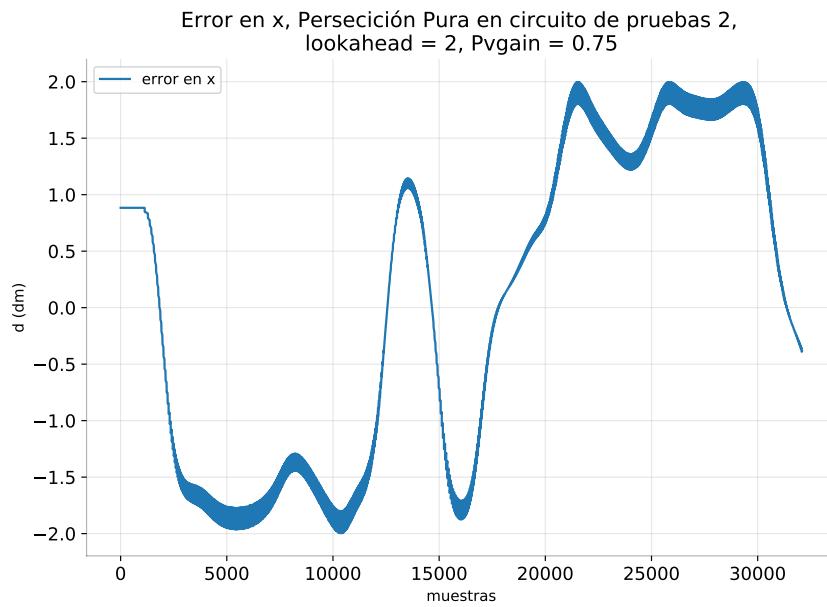


Figura 5.14: Error en distancia para x, Persecución Pura para circuito 2: Catalunya

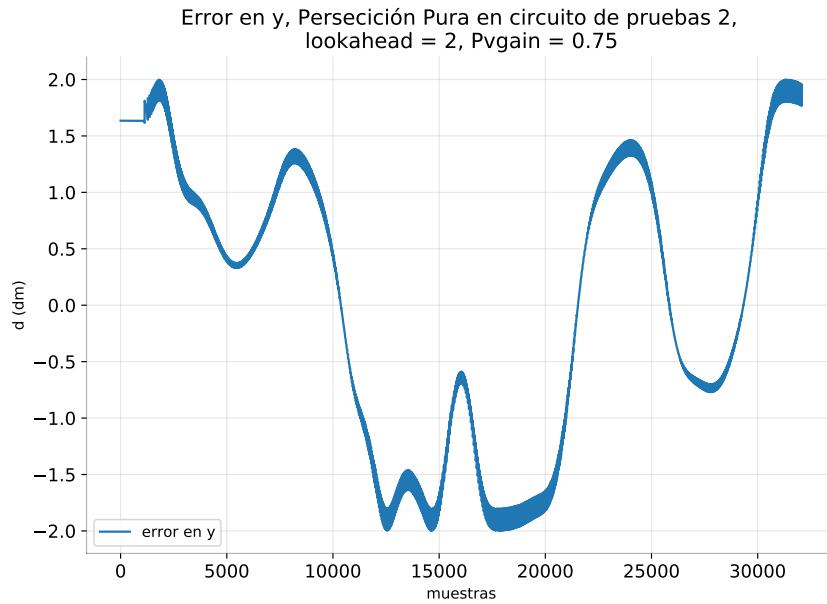


Figura 5.15: Error en distancia para y, Persecución Pura para circuito 2: Catalunya

5.2.2. Resultados preliminares de seguimiento de trayectorias para alta velocidad

En cuanto a los resultados preliminares para la Persecución Pura, bajo las condiciones de alta velocidad y la selección de parámetros realizada, se logró finalizar con éxito vueltas completas al circuito, aún presentando derrapes, en ambos circuitos, y curvas cerradas, en el caso de Catalunya. Por tanto, el algoritmo cumple con su objetivo bajo las condiciones pertinentes e inherentes a una competición *F1TENTH* y este proyecto. Es importante finalizar recordando que este algoritmo no toma en consideración el modelado dinámico del vehículo ni de las ruedas, y se recomienda que, para futuros proyectos, se implemente algún algoritmo que si las considere para lograr un mejor seguimiento de trayectoria a mayores velocidades. Como se demostró a lo largo de las simulaciones para Persecución Pura, los parámetros de $lookahead = 2$ y $P_{vgain} = 0,75$ son convenientes para las condiciones que presentan ambos circuitos de prueba.

5.3. Evasión de obstáculos estáticos

En esta sección se muestran los resultados obtenidos de las simulaciones que se realizaron para el algoritmo de evasión de obstáculos implementado y descrito en la sección 2.5.3 y 4.2.1 y la tabla 4.4.

5.3.1. Implementación de Braitenberg modificado: Miedo

En el caso de este algoritmo, a diferencia de el de Persecución Pura, fue necesario iterar múltiples veces sobre los parámetros para lograr los comportamientos deseados, por dicha razón, si fue necesario buscar una configuración de parámetros que funcionara en pro de alcanzar los objetivos de la competición *F1TENTH* y este proyecto.

Como se demuestra en las gráficas pertenecientes a esta sección, para el circuito de pruebas 1 (figuras 5.16 hasta 5.20) se iteró para 4 y 12 zonas o cantidad de segmentos de medición y distancias de umbral de 0.5 m, 1.5 m, 2 m, 2.5 m, 3 m, 3.5 m y 5 m. Mientras que para el circuito de pruebas 2: Catalunya (figuras 5.21 hasta 5.30), se iteró para 4, 6 y 12 zonas o cantidad de segmentos de medición y distancias de umbral de 0.5 m, 1.5 m, 2 m, 2.5 m, 3 m, 3.5 m y 5 m. Lo anterior con el objetivo de determinar con cuáles parámetros se logran finalizar vueltas completas a los circuitos de prueba.

Círculo de pruebas 1

Note que para la figura 5.21, se logró encontrar una combinación de parámetros que finaliza vueltas completas al circuito, esta selección fue de 4 zonas o segmentos de medición y una distancia de umbral $d_{umbral} = 3\text{ m}$. Por otra parte, se destaca que, como se muestra en detalle en la figura 5.23, a excepción de $d_{umbral} = 2,5$, la todas de las distancias de umbral por debajo y por encima de 3 m fallaron en finalizar vueltas completas.

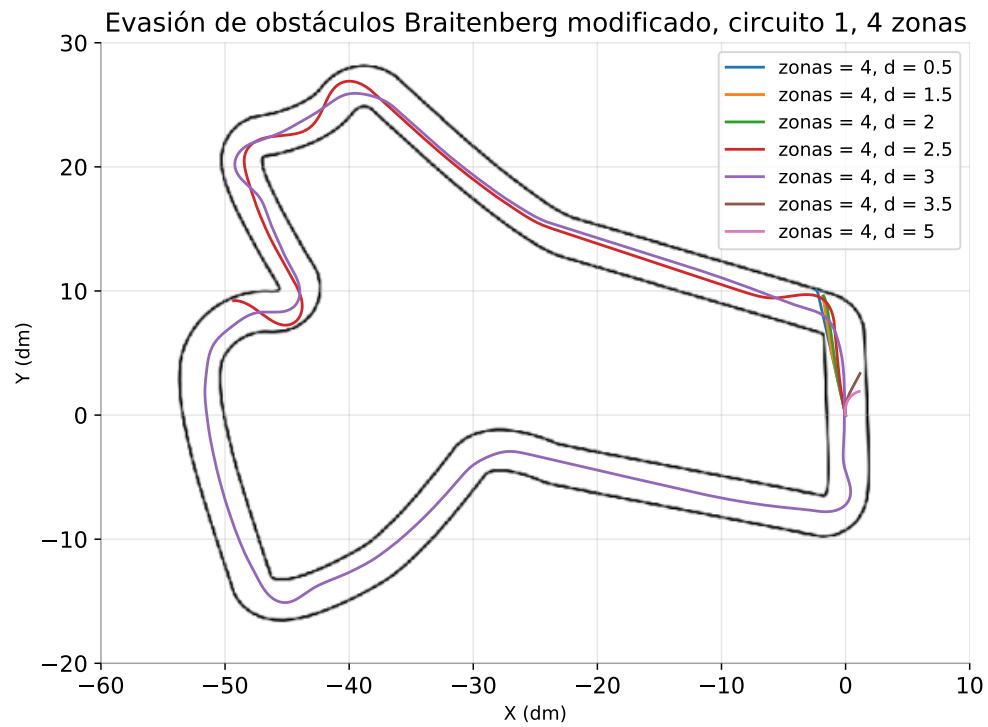


Figura 5.16: Simulación de Braatenberg Modificado: Miedo para circuito 1. $n = 4$

Evasión de obstáculos Braitenberg modificado, circuito 1, 4 zonas

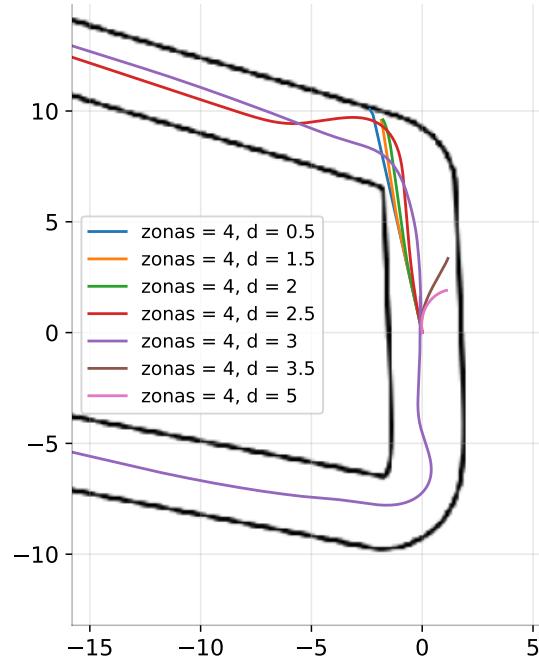


Figura 5.17: Simulación de Braatenberg Modificado: Miedo para circuito 1. n = 4. Ampliación 1.

Evasión de obstáculos Braatenberg modificado, circuito 1, 4 zonas

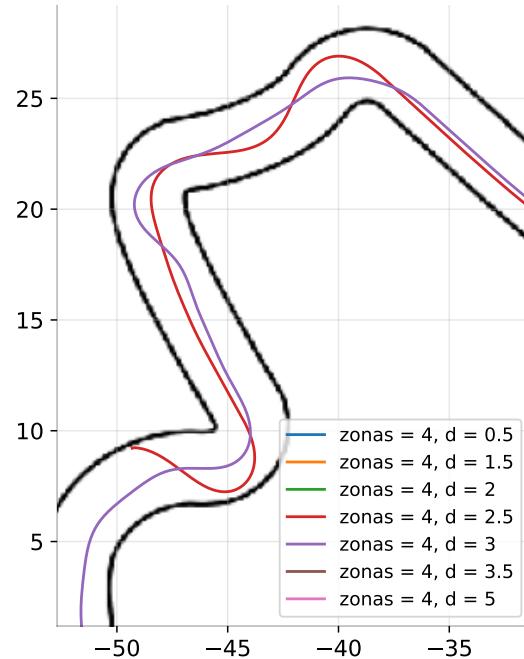


Figura 5.18: Simulación de Braatenberg Modificado: Miedo para circuito 1. n = 4. Ampliación 2.

Para las simulaciones de 12 segmentos de medición no se logró completar vueltas completas para ninguna de las selecciones del parámetro de distancia de umbral d_{umbral} . Lo más próximo a completar vueltas fue para $d_{umbral} = 3$ y $d_{umbral} = 3,5$, como se muestra en la figura 5.20 de lo cual se puede destacar, ya que en el rango de valores superiores de $d_{umbral} = 2,5$ dentro del rango de las pruebas realizadas, fue donde se alcanzaron mayores distancias sin colisionar. Esto puede ser un factor a considerar para futuras implementaciones, porque una exploración más profunda de la parametrización puede arrojar mayor variedad de resultados, ya que a mayor cantidad de zonas de medición, sus promedios varían, y, por tanto, existiría variación en el parámetro de d_{umbral} requerido para cumplir el objetivo.

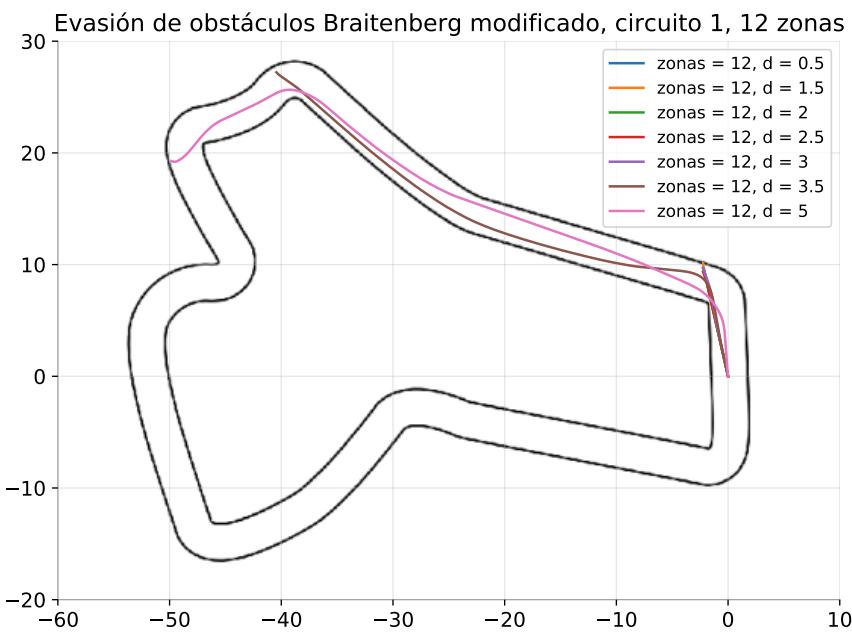


Figura 5.19: Simulación de Braitenberg Modificado: Miedo para circuito 1. n = 12

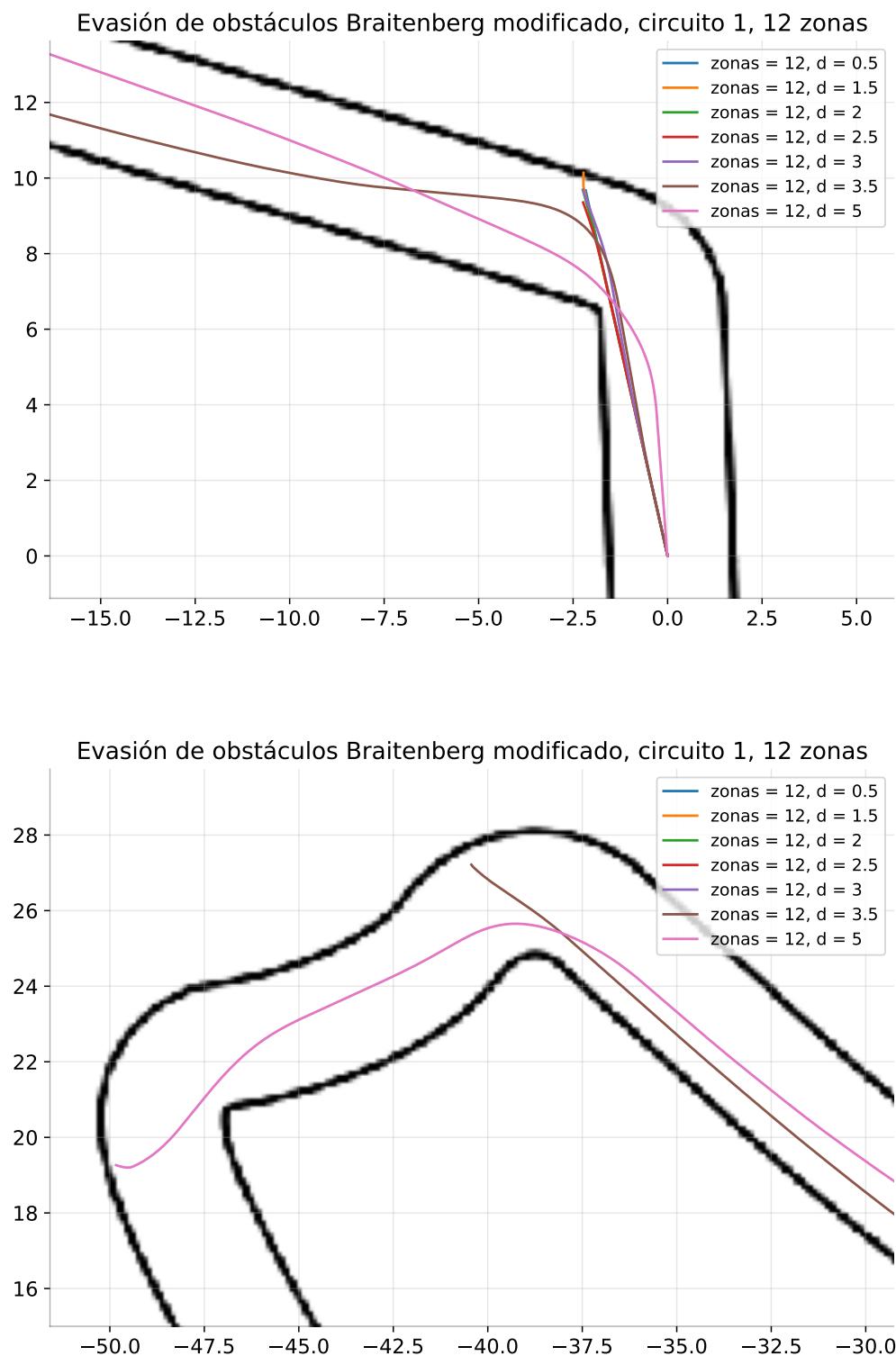


Figura 5.20: Simulación de Braitenberg Modificado: Miedo para circuito 1. n = 12

Círculo de pruebas 2: Catalunya

Este circuito representó una mayor cantidad de iteraciones para los parámetros debido a su complejidad. En este caso se agregaron simulaciones para 4, 6 y 12 zonas de medición. Para la figura 5.21, con 4 zonas de medición, se observó que únicamente 2 parámetros lograron completar vueltas completas al circuito sin colisionar, estos corresponden con $d_{umbral} = 2$ y $d_{umbral} = 2.5$. Para el resto distancias de umbral para 4 zonas de medición, las pruebas fallaron.

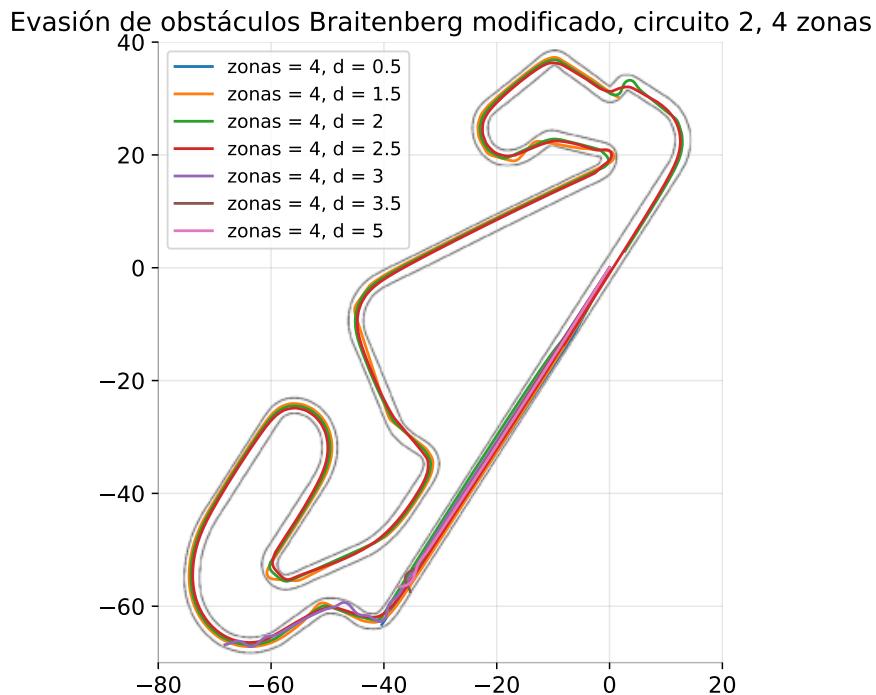


Figura 5.21: Simulación de Braitenberg Modificado: Miedo para circuito 2: Catalunya. $n = 4$

Al observar con detenimiento la ampliación mostrada en la figura 5.23 se puede extraer de la información gráfica que al seleccionar parámetros menores de distancia de umbral, el vehículo tendió a realizar curvas más amplia y a su vez bruscas, con respecto a la $d_{umbral} = 2.5$, un ejemplo claro de esto se muestra en la última curva de la figura inferior en 5.20, donde a pesar de que ambos parámetros, $d_{umbral} = 2$ y $d_{umbral} = 2.5$, logra, completar el objetivo de vueltas completas, la selección de $d_{umbral} = 2$ (verde), supuso un giro más amplio y brusco en comparación a $d_{umbral} = 2$ (rojo).

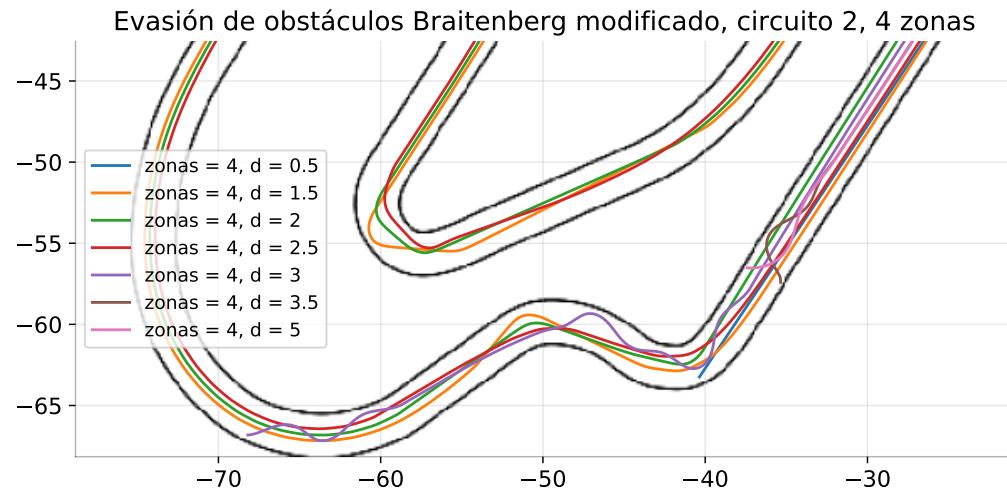


Figura 5.22: Simulación de Braitenberg Modificado: Miedo, circuito 2: Catalunya. n = 4. Ampliación 1.

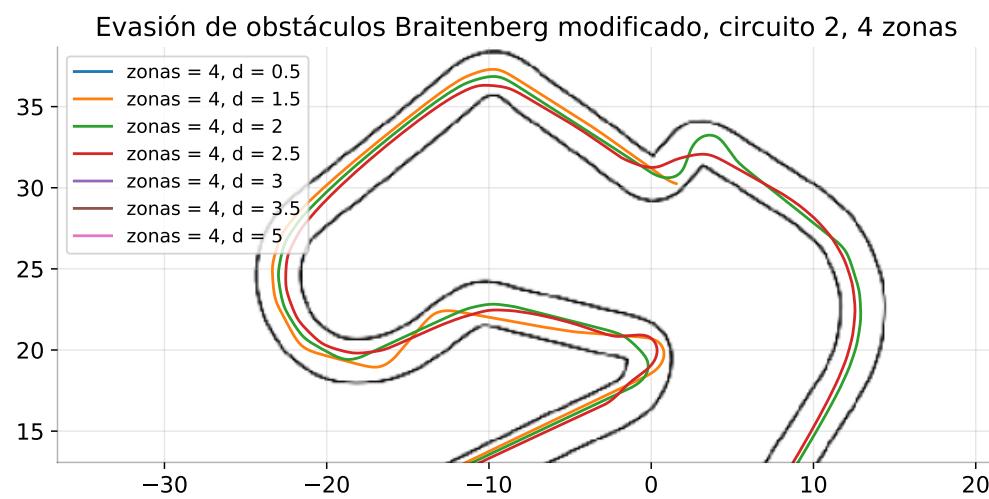


Figura 5.23: Simulación de Braitenberg Modificado: Miedo, circuito 2: Catalunya. n = 4. Ampliación 2.

En cuanto a la implementación para 6 zonas de medición se observa que ningún parámetro seleccionado para d_{umbral} logra completar vueltas de manera efectiva. Y que en comparación con las simulaciones para 4 segmentos de medición, menos trazados lograron superar las primeras curvas del circuito.

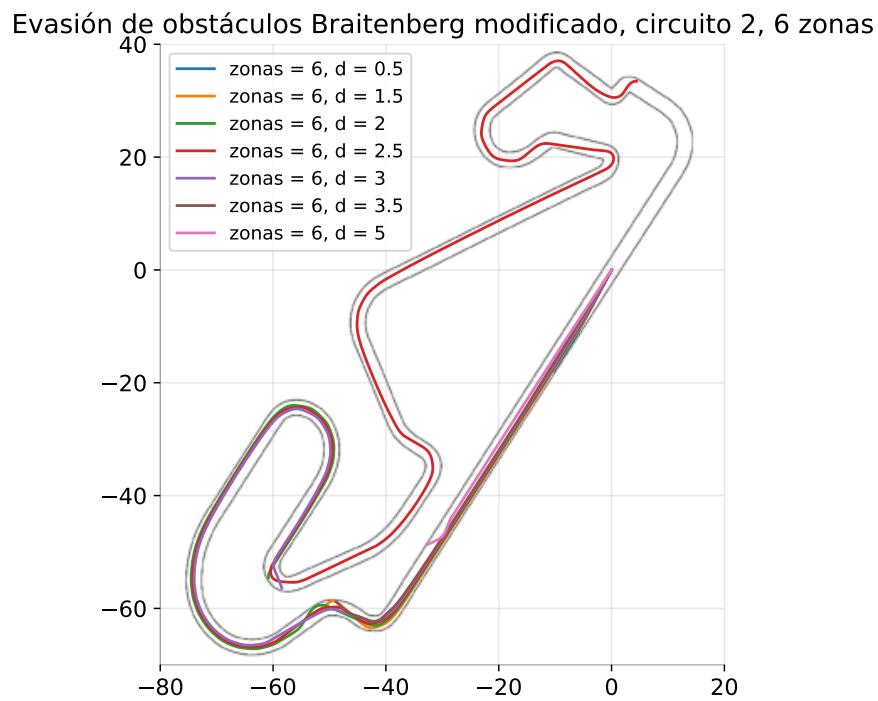


Figura 5.24: Simulación de Braitenberg Modificado: Miedo para circuito 2: Catalunya. $n = 6$

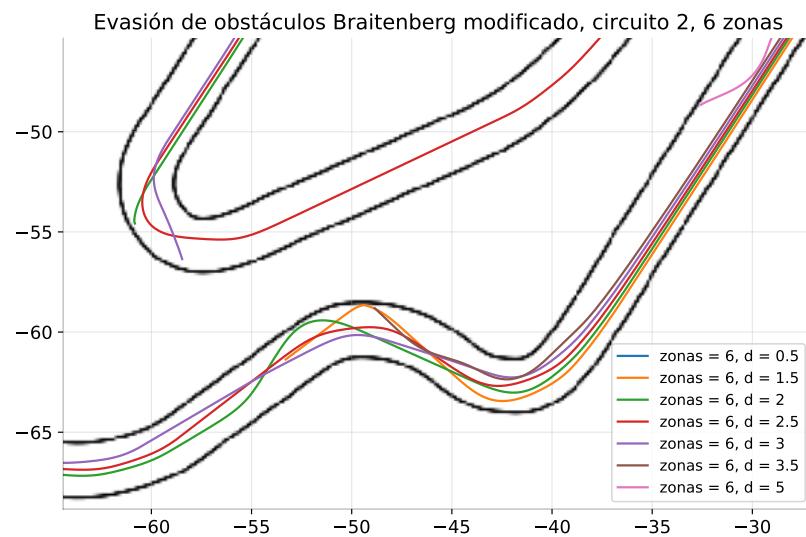


Figura 5.25: Simulación de Braatenberg Modificado: Miedo, circuito 2: Catalunya. n = 6. Ampliación 1.

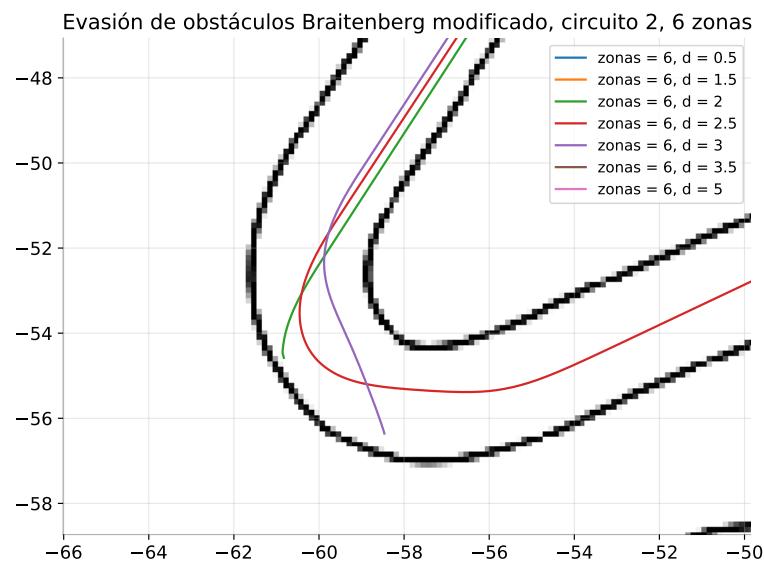


Figura 5.26: Simulación de Braatenberg Modificado: Miedo, circuito 2: Catalunya. n = 6. Ampliación 2.

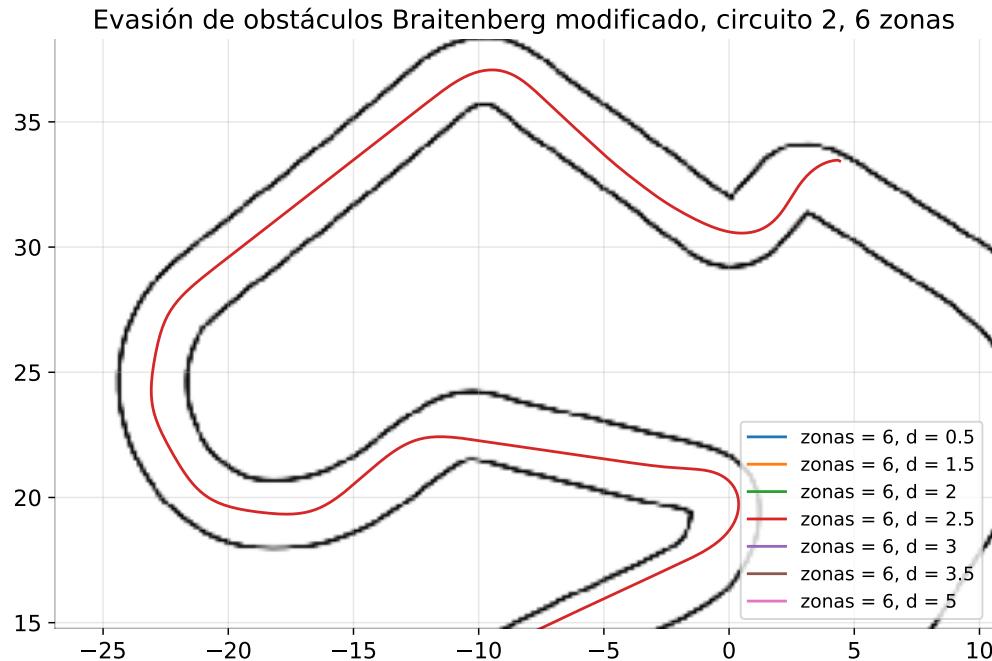


Figura 5.27: Simulación de Braitenberg Modificado: Miedo para circuito 2: Catalunya. $n = 6$. Ampliación 3

Al analizar el comportamiento de las figuras que muestran 12 zonas de medición respecto a las de 4 vemos que, al igual que para 6 segmentos de medición, menos trazados logran alcanzar una mayor distancia en el circuito dado. Al comparar lo observado en las figuras superiores de 5.30 y 5.26, se logró apreciar claramente cómo la cantidad de segmentos afecta el comportamiento para distancias de umbral iguales, se notó que para $d_{umbral} = 2$, en ambas figuras el vehículo colisiona, sin embargo, para las 12 zonas la colisión ocurre de manera temprana en comparación con la selección de 6 zonas de medición.

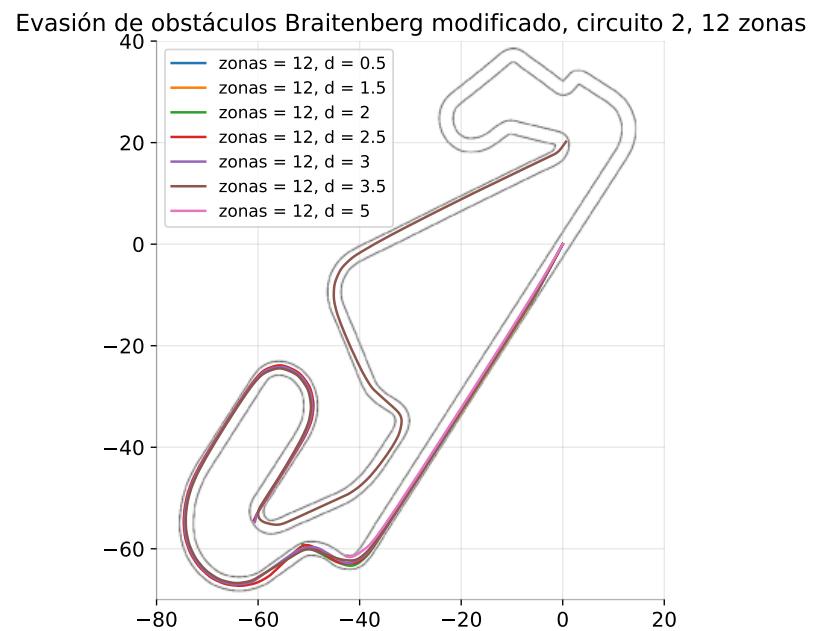


Figura 5.28: Simulación de Braitenberg Modificado: Miedo para circuito 2: Catalunya. n = 12

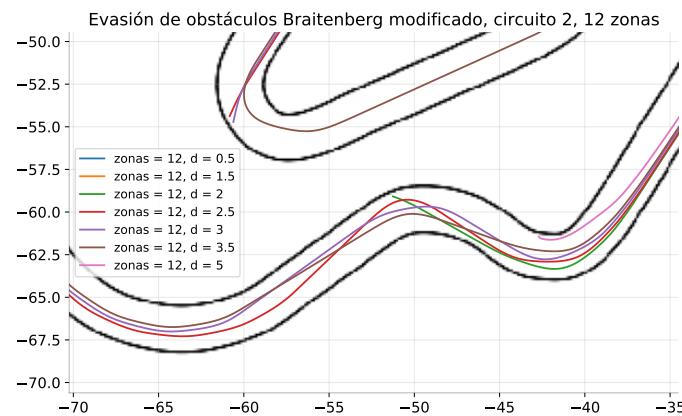


Figura 5.29: Simulación de Braitenberg Modificado: Miedo, circuito 2: Catalunya. n = 12. Ampliación 1.

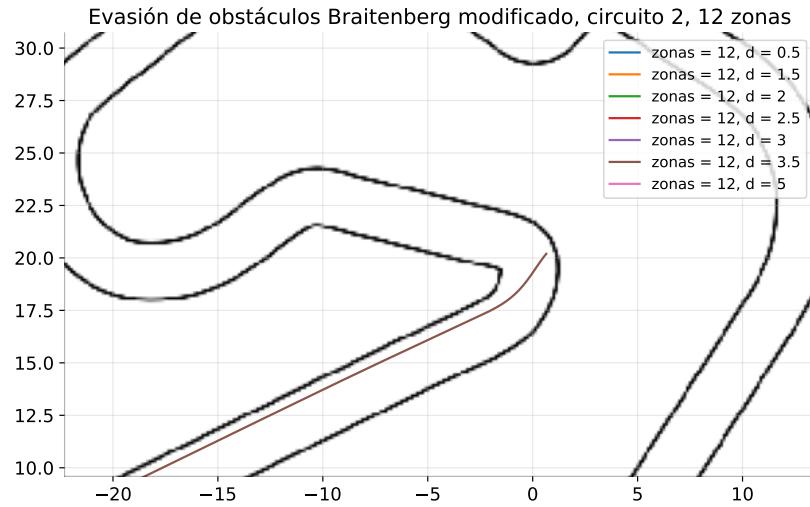


Figura 5.30: Simulación de Braitenberg Modificado: Miedo, circuito 2: Catalunya. $n = 12$. Ampliación 1.

5.3.2. Resultados preliminares para evasión de obstáculos. Braitenberg Modificado: Miedo

En cuando a los resultados preliminares para la implementación del Braitenberg Modificado: Miedo, bajo las condiciones de alta velocidad y la selección de parámetros realizada para las zonas de detección y la distancia de umbral d_{umbral} , se logró finalizar con éxito vueltas completas al circuito, aún presentando derrapes, en ambos circuitos, y curvas cerradas, en el caso de Catalunya. En el caso de la selección de 4 zonas de medición fue donde se obtuvieron resultados exitosos para la finalización del circuito sin colisiones, específicamente para una distancia de umbral $d_{umbral} = 3$, tal y como se mostró en la figura 5.16.

Para la selección 6 y 12 zonas de medición se destaca que, para el límite superior del rango distancias de umbral simulado, se lograron los recorridos más largos sin colisionar. Lo anterior se muestra en detalle en las figuras 5.20 y 5.26.

Las combinaciones de parámetros que logran finalizar vueltas completas se encontraron en la simulación de la figura 5.16 para el circuito de pruebas 1 y en la figura 5.21 para el circuito de pruebas 2: Catalunya. Por tanto, se puede decir que dividir el rango de visión en 4 segmentos en una manera efectiva de finalizar vueltas bajo las condiciones de la prueba.

Como ya se mencionó, a pesar de que no todas las configuraciones de parámetros lograron el objetivo de completar vueltas completas, se encontraron configuraciones de parámetros que si cumplieron con el mismo. Por tanto, el algoritmo de Braitenberg Modificado: Miedo cumple con su objetivo bajo las condiciones pertinentes e inherentes a una competición *F1TENTH* y este proyecto. En la tabla 5.2 se

resumen los resultados para el algoritmo de Braitenberg Modificado: Miedo obtenido en esta sección.

Circuito de pruebas	Parámetros del algoritmo		Vuelta completa finalizada
	Zonas	Distancia Umbral	
Circuito de pruebas 1	4	0,5	No
	4	1,5	No
	4	2	No
	4	2,5	No
	4	3	Si
	4	3,5	No
	4	5	No
	12	0,5	No
	12	1,5	No
	12	2	No
Circuito de pruebas 1	12	2,5	No
	12	3	No
	12	3,5	No
	12	5	No
	4	0,5	No
	4	1,5	No
	4	2	Si
	4	2,5	Si
	4	3	No
	4	3,5	No
Circuito de pruebas 2: Catalunya	4	5	No
	6	0,5	No
	6	1,5	No
	6	2	No
	6	2,5	No
	6	3	No
	6	3,5	No
	6	5	No
	12	0,5	No
	12	1,5	No
	12	2	No
	12	2,5	No
	12	3	No
	12	3,5	No
	12	5	No

Tabla 5.2: Tabla resumen de resultados para Braitenberg Modificado: Miedo.

5.4. Mezcla de comportamientos: Seguimiento de trayectorias + evasión de obstáculos en alta velocidad

En esta sección se muestran los resultados obtenidos de las simulaciones que se realizaron para la mezcla de algoritmo de evasión de obstáculos y seguimiento de trayectorias implementados y descritos en la sección 4.3 y la tabla 4.7. El circuito en este caso presenta obstáculos variados distribuidos a lo largo del circuito para probar como se comporta el algoritmo ante la presencia de obstáculos estáticos. La combinación de los algoritmos se logró mediante la ponderación de sus salidas para velocidad lineal y ángulo de dirección, lo anterior se logró utilizando las fórmulas 4.3 y 4.4, correspondientemente. Cabe recordar que los pesos de ponderación son GSW^1 , LSW^2 , $GSTRW^3$ y $LSTRW^4$.

5.4.1. Persecución Pura + Braitenberg modificado: Miedo

Círculo de pruebas con obstáculos estáticos 2: Catalunya

En la figura 5.31 se observa con detalle la distribución de obstáculos a lo largo del circuito de pruebas 2. Esta mezcla de algoritmos mediante ponderación utilizando pesos para los comandos que cada algoritmo envía de manera independiente supone una parametrización que requiere múltiple iteración para lograr un comportamiento completamente optimizado, sin embargo, logrando demostrar que el algoritmo cumple el objetivo de finalizar vueltas sin colisionar, no se procedió a mostrar en detenimiento el proceso de iteración, ya que este proceso se encuentra fuera del enfoque del proyecto. Sin embargo, se debe de destacar que en el caso de la trayectoria naranja de la figura 5.31 en la ecuación 4.3, se deseaba eliminar por completo la influencia de la velocidad lineal dada por el algoritmo de Braitenberg Modificado: Miedo, puesto que la velocidad que se deseaba utilizar como referencia fue la dada por el seguimiento de trayectoria con Persecución Pura, por este motivo es que LSW es igual a cero. En esta misma línea, respecto al ángulo de dirección dado por la ecuación 4.4, hay que recalcar que el peso mayor se otorgó al seguimiento de trayectoria, ya que lo que se desea es la prevalencia de sel seguimiento de trayectoria y que reactivamente se evadan obstáculos frente al vehículo. Se muestra únicamente 3 selecciones de parámetros.

¹ GSW : Global Speed Weight $\in [0, 1]$.

² LSW : Local Speed Weight $\in [0, 1]$.

³ $GSTRW$: Global Steering Weight $\in [0, 1]$.

⁴ $LSTRW$: Local Steering Weight $\in [0, 1]$.

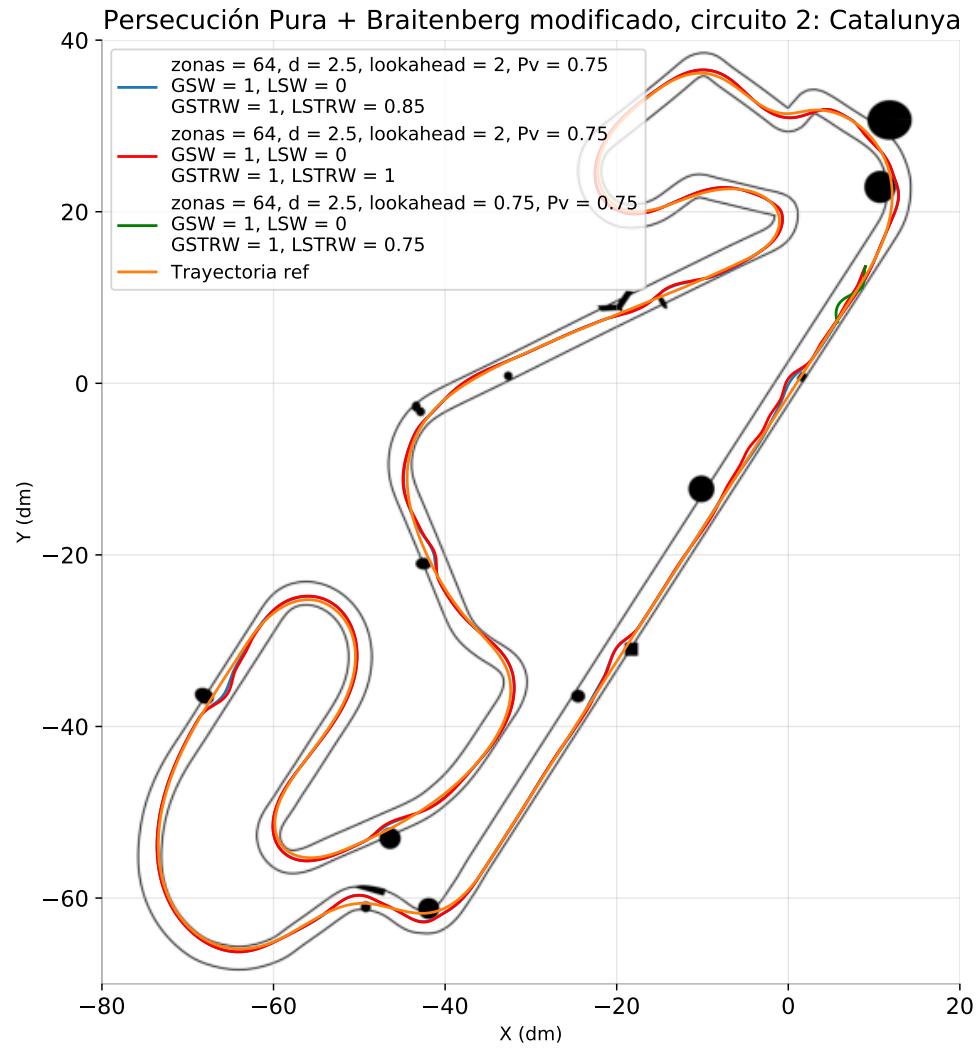


Figura 5.31: Simulación de mezcla de comportamientos, circuito 2: Catalunya.

Al observar la figura 5.32, se notó que si se disminuye demasiado la distancia *lookahead*, el seguimiento de trayectoria se descontrola (trayectoria verde oscuro) y se da una colisión. Dicho comportamiento concuerda con lo demostrado en la sección 5.1 para el algoritmo de Persecución Pura.

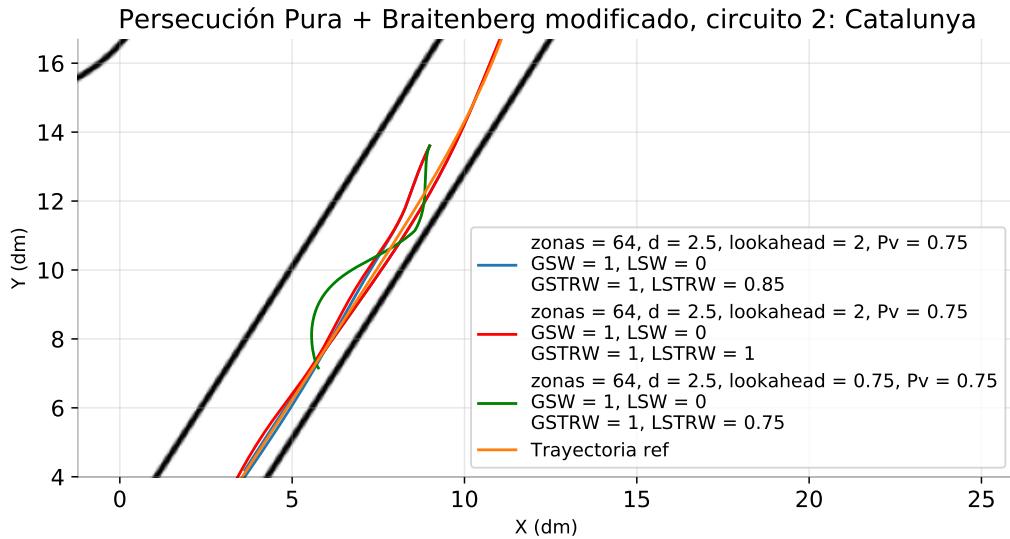


Figura 5.32: Simulación de mezcla de comportamientos, circuito 2: Catalunya. Ampliación 1.

En las figuras 5.33 y 5.34 se pudo observar cómo la selección de parámetros que se realizó, tanto para los pesos como para los parámetros independientes de cada algoritmo, culmina en la evasión efectiva de obstáculos. Se destaca que al otorgarle pesos iguales los comandos de dirección de ambos algoritmos en la ponderación de dirección de mezcla, el algoritmo de evasión influye más en el trazado y es visible que se pierde en cierta medida el seguimiento de trayectoria que se logra cuando se da menos peso a la evasión en la ponderación de dirección.

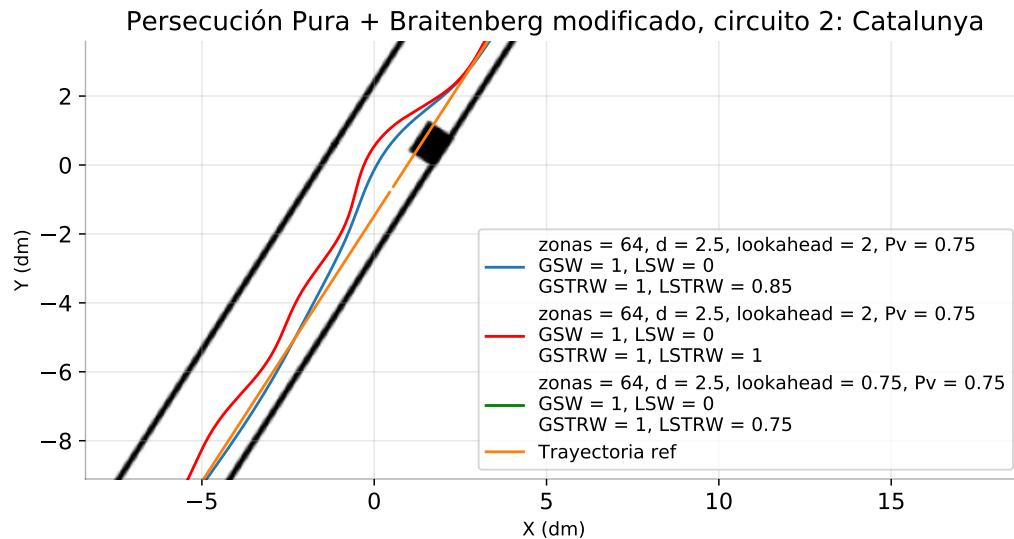


Figura 5.33: Simulación de mezcla de comportamientos, circuito 2: Catalunya. Ampliación 2.

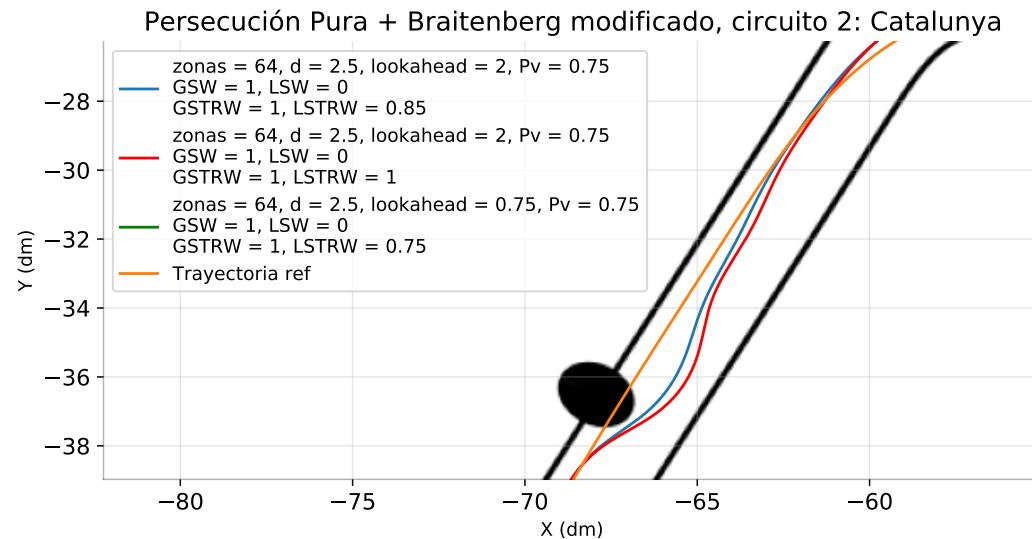


Figura 5.34: Simulación de mezcla de comportamientos, circuito 2: Catalunya. Ampliación 3.

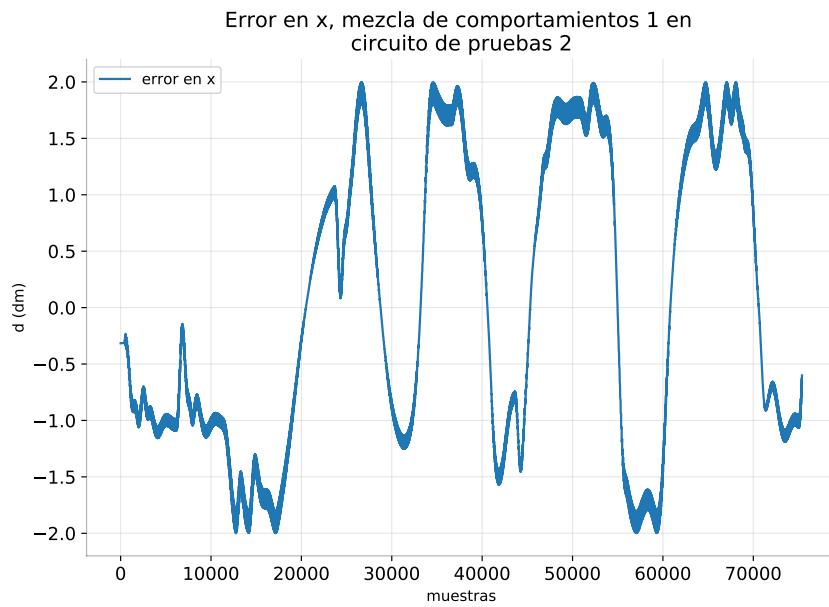


Figura 5.35: Error en distancia para x. Mezcla de comportamientos

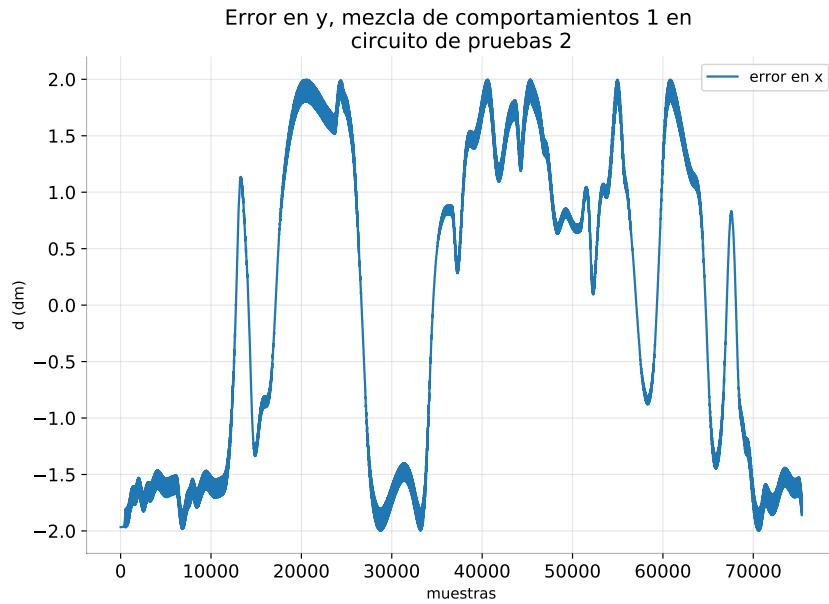


Figura 5.36: Error en distancia para y. Mezcla de comportamientos

5.4.2. Resultados preliminares de mezcla de comportamientos

En esta sección se logró demostrar que con la selección de parámetros realizada se logran completar vueltas al circuito de manera efectiva y realizando evasión de obstáculos. Se debe destacar que la cantidad de zonas de medición para la evasión de obstáculos seleccionada fue 64 y se seleccionaron únicamente las zonas frente al vehículo, ya que cuando se viaja a alta velocidad siguiendo una trayectoria predefinida, la incidencia de obstáculos que se desea evitar se encuentra al frente del vehículo, para que la naturaleza reactiva del algoritmo de Braitenberg no afecte el seguimiento de trayectorias cuando hay obstáculos fuera de la línea frontal del vehículo.

Se destaca que al otorgar mayores pesos a la evasión de obstáculos, se puede observar gráficamente que el seguimiento de trayectorias se vuelve menos efectivo, sin embargo, bajo la parametrización adecuada se sigue alcanzando el objetivo de finalizar vueltas sin colisiones.

5.5. Resumen de resultados y análisis final

5.5.1. Tabla comparativa de resultados

Esta sección incluye una comparación entre los algoritmos implementados, con la intención de comprender mejor el desempeño de cada uno respecto a los demás. Se toma en consideración los aspectos de tiempo de vuelta, velocidad máxima y velocidad promedio durante el recorrido, circuito completado y ángulo de dirección promedio (absoluto). Los resultados para los índices de desempeño asociados por ponderación que se muestran en la tabla 5.3 fueron calculados mediante la siguiente ponderación:

$$\sim ndice = \frac{tiempo\ de\ vuelta}{0,5} + circuito\ completado * 1000 + Velocidad\ Max * 5 + \\ + |Ang.\ de\ dir.\ promedio| * 1 + Velocidad\ Promedio * 200 \quad (5.1)$$

Dichos índices presentados en esta tabla 5.3 están asociados a las simulaciones realizadas en el circuito de pruebas 2: Catalunya con y sin obstáculos. El recorrido con obstáculos está asociado a las iteraciones para el algoritmo de mezcla de comportamiento.

Algoritmo	Parametros usados	Indices Evaluados						Índice de desempeño asociado por ponderación	IAE distancia
		Tiempo de vuelta (s)	Círculo completadado	Velocidad Máxima (m/s)	Velocidad Promedio (m/s)	Angulo de dirección promedio, absoluto (rad)			
Persecución Pura	lookahead = 2, Pvgain = 0.75	88.257	1	6	5.4725	0.006863	2291.02	9.926x10 ⁵	
Braitenberg Modificado	zonas = 4, dumbral = 2	119.102	1	4	3.99	0.005441	1877.56	-	
Braitenberg Modificado	zonas = 4, dumbral = 2.4	117.638	1	4	3.99	0.005244	1876.82	-	
Braitenberg Modificado	zonas = 6, dumbral = 2.5	94.193	0	4	3.99	0.005121	865.1	-	
Braitenberg Modificado	zonas = 12, doumbral = 3.5	77.389	0	4	3.99	0.010414	856.7	-	
Mezcla de comportamientos	zonas = 64, dumbral = 2.5, lookahead = 2, Pvgain = 0.75, GSW = 1, LSW = 0, GSTRW = 1, LSTRW = 0.85	88.901	1	6	5.4767	0.007229	2169.8	3.574x10 ⁶	
Mezcla de comportamientos	zonas = 64, dumbral = 2.5, lookahead = 2, Pvgain = 0.75, GSW = 1, LSW = 0, GSTRW = 1, LSTRW = 1	91.113	1	6	5.4352	0.008709	2162.61	3.781x10 ⁶	
Peso asociado a índice		0,5	1000	5	200	1			

Tabla 5.3: Tabla comparativa de índices de desempeño

Analizando en detalle la tabla 5.3, se observó que el algoritmo con mejor desempeño en el circuito de pruebas 2: Catalunya sin obstáculos fue el de Persecución Pura. En el caso del circuito de pruebas 2: Catalunya con obstáculos, se observó que los resultados no son tan diferentes entre sí, sin embargo, el mejor índice de desempeño se obtuvo para el conjunto de parámetros: $zonas = 64, d_{umbral} = 2m, lookahead = 2m, Pv gain = 0.75, GSW = 1, LSW = 0, GSTRW = 1, LSTRW = 0.85$.

Cabe Mencionar que uno de los aportes más importantes de este capítulo se encuentra en la tabla 5.3, ya que a partir de los resultados ahí mostrados se puede concluir que, al comparar los resultados para los 3 algoritmos, se destaca que la Mezcla de comportamientos sería el algoritmo más indicado para poder competir, ya que, a pesar de que este obtuvo índices de desempeño ligeramente menores que el algoritmo de Persecución Pura, la Mezcla de comportamientos tiene la capacidad de evadir obstáculos de manera efectiva, siempre y cuando no se sobrepasen sus capacidades físicas ni cinemáticas. Lo anterior también se pudo notar en la última columna, la cual muestra el valor de la Integral de Error Absoluto para la distancia. Estos valores confirmaron que la Mezcla de comportamientos de la penúltima fila es el algoritmo que obtuvo un mejor desempeño en este aspecto. Como la evasión de obstáculos no implica el seguimiento de trayectorias no se procedió a calcular el IAE⁵ de distancia para sus simulaciones.

⁵ $IAE = \sum [\sqrt{(ref_x - Pose_x)^2 + (ref_y - Pose_y)^2}]$

Capítulo 6

CONCLUSIONES Y RECOMENDACIONES

6.1. Conclusiones

- Se describieron las características físicas, así como el modelado matemático y la parametrización de la plataforma Ackermann *F1TENTH*.
- Gracias a la investigación realizada e información recopilada para el marco teórico, se logró centralizar y referenció a fuentes relevantes relacionadas con algoritmos de navegación autónoma que podrían ser implementados en futuros proyectos e investigaciones del CERLab. Además, dicha información incluye descripciones de la competencia de *F1TENTH* y requerimientos técnicos que se deben de tener en cuenta si se desea participar en la misma. Adicionalmente, se abarcó de manera resumida el funcionamiento de ROS, RViz y el entorno de simulación utilizado para generar los resultados del proyecto.
- Como se apreció en el capítulo 4, se realizó de manera exitosa la implementación algoritmos de navegación autónoma para el robot *F1TENTH* bajo el entorno de simulación virtual y en 2 distintas pistas para finalizar al menos una vuelta completa a dichos circuitos de prueba.
- En este proyecto se logró implementar un algoritmo de seguimiento de trayectoria, uno de evasión de obstáculos y finalmente uno que mezcla estos comportamientos para concluir vueltas a un circuito realizando evasión de obstáculos y seguimiento de trayectorias en condiciones de plano horizontal y con el modelo dinámico del vehículo implementado en el simulador.
- A pesar de que el algoritmo Persecución Pura no toma en consideración los comportamientos dinámicos inherentes a la alta velocidad, se concluye que dicho algoritmo funciona de manera efectiva para el seguimiento de trayectorias en alta velocidad para las condiciones establecidas en este proyecto.
- En cuanto al algoritmo de Braitenberg, se destaca que fue necesario realizar una modificación para que su funcionamiento fuera el deseado, ya que en entornos cerrados la evasión del algoritmo original no funcionó efectivamente. Esta modificación se implementó gracias a una recomendación realizada en un proyecto eléctrico anterior. Adicionalmente, se debe destacar que en las

pruebas de su funcionamiento independiente, de las 35 combinaciones de parámetros probadas, únicamente se logró completar circuitos en 3 casos. Sin embargo, a la hora de mezclar los comportamientos, su naturaleza reactiva funciona muy bien evadiendo los obstáculos estáticos, tal y como se demostró los resultados de la sección 4.3.

- De acuerdo a los resultados obtenidos para la mezcla de comportamientos, se concluye que la combinación de las señales de control del algoritmo de Persecución Pura y del algoritmo de Braitenberg Modificado mediante ponderación, representa una manera efectiva de realizar el seguimiento de trayectoria y la evasión de obstáculos en el formato de la competencia F1TENTH. Se concluye mediante el análisis de los índices de desempeño que, para los parámetros de la mezcla de comportamientos, $zonas = 64$, $d_{umbral} = 2m$, $lookahead = 2m$, $P_{vgain} = 0,75$, $GSW = 1$, $LSW = 0$, $GSTRW = 1$, $LSTRW = 0,85$, el vehículo logra el objetivo de la competencia con mejores resultados que el resto de selecciones de parámetros.
- Se debe de destacar que la selección adecuada de los parámetros para cada algoritmo fue vital para obtener el comportamiento deseado. En el algoritmo de Persecución Pura se concluye que una distancia *lookahead* pequeña funciona bien para mantener una trayectoria, mientras que un *lookahead* grande es mejor para retomar trayectorias cuando el vehículo no se encuentra próximo a ellas en las arenas del F1TENTH.
- Se realizó la validación de los algoritmos de forma iterativa, y su comparación utilizando índices de desempeño para determinar su capacidad de cumplir con el objetivo planteado; el realizar vueltas completas a los circuitos de prueba sin colisionar. Esto incluyó la navegación sin obstáculos para Persecución Pura y Braitenberg Modificado, y con obstáculos para la mezcla de comportamientos. Al comparar los resultados para los 3 algoritmos se llega a la conclusión de que la Mezcla de comportamientos sería el algoritmo más indicado para poder competir, ya que, a pesar de que se obtuvieron índices de desempeño ligeramente menores que para el caso de Persecución Pura, este tiene la capacidad de evadir obstáculos de manera efectiva, siempre y cuando no se superen sus capacidades físicas ni cinemáticas.

6.2. Recomendaciones

- Para futuros proyectos eléctricos se recomienda revisar las referencias a repositorios que se encuentran a lo largo de este proyecto, ya que son una compilación de códigos, implementaciones e información relevante para el tema de la navegación autónoma y la competición F1TENTH.
- Es importante que cuando se realicen simulaciones se tenga seguridad de que la capacidad de procesamiento sea suficiente, ya que este factor afecta directamente la simulación y, por tanto, supondría una variación en los parámetros seleccionados.
- Se insta que se tome como base este proyecto para futuros proyectos eléctricos para que se logren implementar algoritmos de seguimiento de trayectorias y evasión de obstáculos que tomen en consideración las características dinámicas dentro de su algoritmo y no solamente dentro del simulador.
- Realizar simulaciones con los mismos parámetros para los algoritmos, pero cambiando el modelo cinemático y dinámico del vehículo utilizado a lo interno del simulador para entender con mayor claridad la afectación del modelado interno del simulador a los algoritmos.
- Implementar mejoras al cálculo de la velocidad para rutas pre calculadas, donde se modifiquen los comandos de velocidad de acuerdo a la variación de la pendiente entre dos puntos más adelante en el camino, de tal manera que se pueda optimizar la velocidad de conducción.
- Simular exhaustivamente los algoritmos para ver si después de una determinada cantidad de vueltas consecutiva su comportamiento cae en detrimento y no logra el objetivo de completar las vueltas.
- Se recomienda revisar la herramienta desarrollada por la comunidad F1TENTH se encuentra el repositorio [F1TENTH_SVL-Simulator](#), la cual puede ser de utilidad para futuros proyectos e investigaciones, ya que es un simulador altamente depurado y que se integra con otras herramientas sofisticadas de visualización y manejo físico/gráfico.
- Se recomienda encontrar alguna manera de optimizar las parametrizaciones de los algoritmos para así, poder obtener combinaciones de parámetros aún mejores que las encontradas. Ojalá realizándolo de una manera metodológica y no por método iterativo.
- Se recomienda que además se validen los algoritmos en circuitos en los cuales los obstáculos sean agregados de forma aleatoria buscando la robustez de los algoritmos. Lo anterior, dado que los obstáculos se agregaron en posiciones fijas para todas las pruebas de los algoritmos con evasión.
- Se recomienda consultar [F1TENTHBuild](#) y [F1TENTH - Course Documentation](#), en donde está disponible una wiki completa para iniciar con la construcción, configuración, instalación y simulación del proyecto de la comunidad F1TENTH.

6.3. Trabajos futuros

Se Pueden realizar mejoras al los algoritmos que se implementaron en este proyecto, entre estas se puede mencionar que, para la mezcla de comportamientos, se podría trabajar con un ajuste de parámetros dinámico decida activamente cuál algoritmo debería de tener más influencia sobre las salidas de control. Se podría emplear la mezcla de comportamientos por ponderación, pero con pesos dinámicos. Otra posible mejora a los algoritmos implementados es la implementación de un planeador que calcule y optimice en tiempo real la trayectoria a seguir, esto para no tener que utilizar datos pre calculados para la trayectoria.

Por otra parte, a lo largo del trabajo se hizo referencia a gran cantidad de algoritmos de navegación que podrían ser implementados, como el seguimiento de trayectorias por MPC, *Dynamic Window Approach*, *Time Elastic Band*, *SLAM* y *Scan Matching*.

Adicionalmente, se podrían implementar pruebas para determinar si los algoritmos presentados en este proyecto puede lograr la evasión de obstáculos dinámicos y además la implementación de algoritmos especializados para dicha evasión dinámica. Para trabajos futuros se recomienda la revisión de:

- Métodos reactivos: *Follow the gap* [40] y variantes.
- Mapeo y localización: *Scan Matching* [16], filtro de partículas [24] y SLAM [46].
- planificación: Persecución Pura [11,39,44], RRT (*Rapidly-exploring Random Tree*) [21], generación y seguimiento de trayectorias por MPC [10, 23] y optimización de velocidad [19] y la línea de carrera [36].
- Visión, percepción y detección: detección y estimación de pose [3], expansión de simple vista por geometría multi vista [22] y detección de objetos con YOLO (*You Only Look Once*) [12].
- Métodos de control [18, 43, 45].

En esas fuentes podrá encontrar información para temas importantes como la obtención de un trazado óptimo y la optimización del perfil de velocidad.

Bibliografía

- [1] Commonroad: Composable benchmarks for motion planning on roads. *2017 IEEE Intelligent Vehicles Symposium (IV)*, 2017.
- [2] Ahmad Abbadi and Vaclav Prenosil. Safe path planning using cell decomposition approximation. 05 2015.
- [3] Mary B Alatise and Gerhard P Hancke. Pose estimation of a mobile robot based on fusion of imu data and vision data using an extended kalman filter. *Sensors*, 17(10):2164, 2017.
- [4] V. M. Alfaro. *Manual de usuario de la clase e ieproyecto - IE-0499 Proyecto eléctrico*. Escuela de Ingeniería Eléctrica, Universidad de Costa Rica, febrero 2013.
- [5] M Althoff and G .W ursching. *CommonRoad: Vehicle Models*. Technische Universitat Munchen, 2020.
- [6] Monica Arias and Nataliya Nechyporenko. Generalized voronoi diagram application to turtlebot navigation in ros, 06 2016.
- [7] A.O. Baturone. *Robótica: Manipuladores y Robots Móviles*. Marcombo, 2005.
- [8] K. Berns and E. Puttkamer. *Autonomous Land Vehicles: Steps towards Service Robots*. Vieweg+Teubner Verlag, 2009.
- [9] V. Braitenberg. *Vehicles: Experiments in Synthetic Psychology*. Bradford Books. MIT Press, 1986.
- [10] Ardashir Bulsara, Adhiti Raman, Srivatsav Kamarajugadda, Matthias Schmid, and Venkat Krovi. Obstacle avoidance using model predictive control: An implementation and validation study using scaled vehicles. 04 2020.
- [11] R. Craig Coulter. Implementation of the pure pursuit path tracking algorithm. 1992.
- [12] Douglas Henke Dos Reis, Daniel Welfer, Marco Antonio De Souza Leite Cuadros, and Daniel Fernando Tello Gamarra. Mobile robot navigation using an object recognition software with rgbd images and the yolo algorithm. *Applied Artificial Intelligence*, 33(14):1290–1305, 2019.
- [13] Syed Abdullah Fadzli, Sani Iyal Abdulkadir, Mokhairi Makhtar, and Azrul Amri Jamal. Robotic indoor path planning using dijkstra's algorithm with multi-layer dictionaries. In *2015 2nd International Conference on Information Science and Security (ICISS)*, pages 1–4, 2015.

- [14] Dave Ferguson, Thomas M. Howard, and Maxim Likhachev. Motion planning in urban environments: Part ii. In *2008 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 1070–1076, 2008.
- [15] Open Source Robotics Foundation. About ros.
- [16] J.-S. Gutmann and C. Schlegel. Amos: comparison of scan matching approaches for self-localization in indoor environments. In *Proceedings of the First Euromicro Workshop on Advanced Mobile Robots (EUROBOT '96)*, pages 61–67, 1996.
- [17] Thomas Hellström and Ola Ringdahl. Follow the past: a path-tracking algorithm for autonomous vehicles. *Int. J. Vehicle Autonomous Systems*, 4:2–4, 01 2006.
- [18] Yutaka Kanayama, Yoshihiko Kimura, Fumio Miyazaki, and Tetsuo Noguchi. A stable tracking control method for an autonomous mobile robot. In *Proceedings., IEEE International Conference on Robotics and Automation*, pages 384–389. IEEE, 1990.
- [19] Liuwang Kang, Haiying Shen, and Ankur Sarker. Velocity optimization of pure electric vehicles with traffic dynamics consideration. In *2017 IEEE 37th International Conference on Distributed Computing Systems (ICDCS)*, pages 2206–2211. IEEE, 2017.
- [20] Oguzhan Kose. Real-world application of various trajectory planning algorithms on mit racecar. *arXiv preprint arXiv:2109.00890*, 2021.
- [21] James J Kuffner and Steven M LaValle. Rrt-connect: An efficient approach to single-query path planning. In *Proceedings 2000 ICRA. Millennium Conference. IEEE International Conference on Robotics and Automation. Symposia Proceedings (Cat. No. 00CH37065)*, volume 2, pages 995–1001. IEEE, 2000.
- [22] Abhijit Kundu, K Madhava Krishna, and Jayanthi Sivaswamy. Moving object detection by multi-view geometric techniques from a single camera mounted robot. In *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 4306–4312, 2009.
- [23] F Künhe, J Gomes, and W Fetter. Mobile robot trajectory tracking using model predictive control. In *II IEEE latin-american robotics symposium*, volume 51. Citeseer, 2005.
- [24] Julio Labora Gómez. Análisis comparativo de algoritmos de localización de robots móviles basados en filtros de partículas. 2020.
- [25] Edward Lambert, Richard Romano, and David Watling. Optimal path planning with clothoid curves for passenger comfort. pages 609–615, 05 2019.
- [26] J.P. Laumond and M. Overmars. *Algorithms for Robotic Motion and Manipulation: WAFR 1996*. CRC Press, 1997.

- [27] Pablo Marin-Plaza, Ahmed Hussein, David Martin, and Arturo de la Escalera. Global and local path planning study in a ros-based research platform for autonomous vehicles. *Journal of Advanced Transportation*, 2018:6392697, Feb 2018.
- [28] F.R. Más, Q. Zhang, and A.C. Hansen. *Mechatronics and Intelligent Systems for Off-road Vehicles*. Springer London, 2010.
- [29] P. Ogren and N.E. Leonard. A convergent dynamic window approach to obstacle avoidance. *IEEE Transactions on Robotics*, 21(2):188–195, 2005.
- [30] Brian Paden, Michal Čáp, Sze Zheng Yong, Dmitry Yershov, and Emilio Frazzoli. A survey of motion planning and control techniques for self-driving urban vehicles. *IEEE Transactions on Intelligent Vehicles*, 1, 04 2016.
- [31] Morgan Quigley, Ken Conley, Brian Gerkey, Josh Faust, Tully Foote, Jeremy Leibs, Rob Wheeler, Andrew Y Ng, et al. Ros: an open-source robot operating system. In *ICRA workshop on open source software*, volume 3, page 5. Kobe, Japan, 2009.
- [32] S. Quinlan and O. Khatib. Elastic bands: connecting path planning and control. In [1993] *Proceedings IEEE International Conference on Robotics and Automation*, pages 802–807 vol.2, 1993.
- [33] K. Rebai, O. Azouaoui, M. Benmami, and A. Larabi. Car-like robot navigation at high speed. In *2007 IEEE International Conference on Robotics and Biomimetics (ROBIO)*, pages 2053–2057, 2007.
- [34] F. Reyes. *Robótica - control de robots manipuladores*. Alfaomega Grupo Editor, 2011.
- [35] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. Kinodynamic trajectory optimization and control for car-like robots. *2017 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 5681–5686, 2017.
- [36] Alessandro Rucco, Giuseppe Notarstefano, and John Hauser. An efficient minimum-time trajectory generation strategy for two-track car vehicles. *IEEE Transactions on Control Systems Technology*, 23(4):1505–1519, 2015.
- [37] Christoph Rösmann, Frank Hoffmann, and Torsten Bertram. Integrated online trajectory planning and optimization in distinctive topologies. *Robotics and Autonomous Systems*, 88:142–153, 2017.
- [38] Marcelo Saavedra Alcoba¹, Mariela Gutierrez Callejas², and Luigi EnrÃquez Paz³. AnÃ¡lisis de Comportamientos de VehÃculos de Braitenberg para la bÃrobÃüsando El Robot LEGO EV3. *Fides et Ratio - Revista de DifusiÃcultural y cientÃfica de la Universidad La Salle en Bolivia*, 12:155 – 166, 09 2016.
- [39] Moveh Samuel, Mohamed Hussein, and Maziah Binti Mohamad. A review of some pure-pursuit based path tracking techniques for control of autonomous vehicle. *International Journal of Computer Applications*, 135(1):35–38, 2016.

- [40] Volkan Sezer and Metin Gokasan. A novel obstacle avoidance algorithm:“follow the gap method”. *Robotics and Autonomous Systems*, 60(9):1123–1134, 2012.
- [41] R. Siegwart and I.R. Nourbakhsh. *Introduction to Autonomous Mobile Robots*. A Bradford book. Bradford Book, 2004.
- [42] Tim Stahl, Alexander Wischnewski, Johannes Betz, and Markus Lienkamp. Ros-based localization of a race vehicle at high-speed using lidar. *E3S Web of Conferences*, 95:04002, 01 2019.
- [43] Spyros G Tzafestas. Mobile robot control and navigation: A global overview. *Journal of Intelligent & Robotic Systems*, 91(1):35–58, 2018.
- [44] Rui Wang, Ying Li, Jiahao Fan, Tan Wang, and Xuetao Chen. A novel pure pursuit algorithm for autonomous vehicles based on salp swarm algorithm and velocity controller. *IEEE Access*, 8:166525–166540, 2020.
- [45] Keigo Watanabe, Jun Tang, Masatoshi Nakamura, Shinji Koga, and Toshio Fukuda. A fuzzy-gaussian neural network and its application to mobile robot control. *IEEE transactions on control systems technology*, 4(2):193–199, 1996.
- [46] Zhang Xuexi, Lu Guokun, Fu Genping, Xu Dongliang, and Liang Shiliu. Slam algorithm analysis of mobile robot based on lidar. In *2019 Chinese Control Conference (CCC)*, pages 4739–4745, 2019.
- [47] Kaiyu Zheng. Ros navigation tuning guide. In *Robot Operating System (ROS)*, pages 197–226. Springer, 2021.

Apéndice A

CÓDIGOS DE IMPLEMENTACIÓN

En este capítulo se encuentran recopilados archivos importantes para la implementación realizada. Se dividirá en 3 secciones: Persecución Pura, Braitenberg Modificado: Miedo y mezcla de comportamientos. Además, podrá encontrar todo el código en [este repositorio](#).

A.1. Persecución Pura

A.1.1. Nodo de algoritmo: Persecución Pura

`pure_pursuit.py`

```
1 #!/usr/bin/env python3
2 from pp_utils import get_actuation, load_waypoints, get_current_waypoint
3 import rospy
4 import numpy as np
5 from geometry_msgs.msg import PoseStamped
6 from sensor_msgs.msg import LaserScan
7 from ackermann_msgs.msg import AckermannDriveStamped
8 from nav_msgs.msg import Odometry
9 import tf
10
11 wheel_base = rospy.get_param("wheelbase")
12 lookahead_distance = rospy.get_param("lookahead_distance")
13 P_GAIN = rospy.get_param("P_GAIN")
14 track = rospy.get_param("track")
15
16 drive_topic = rospy.get_param("drive_global")
17 odom_topic = rospy.get_param("odom_topic")
18
19 class PurePursuitPlanner():
20     """
21     Example Planner
22     """
23     def __init__(self):
24
```

```

25     self.wheelbase      =  wheel_base
26     self.current_x     =  0.0
27     self.current_y     =  0.0
28     self.current_heading =  0.0
29     self.current_vx    =  0.0
30     self.curvature     =  0.0
31
32     self.actuation      =  AckermannDriveStamped()
33
34     self.waypoints = load_waypoints()
35
36     self.in_planning_frt_pass = True
37
38     # Publishers:
39     self.drive_pub   =  rospy.Publisher(drive_topic, AckermannDriveStamped,
queue_size=100)
40
41     # Subscribers:
42     self.odom_sub    =  rospy.Subscriber(odom_topic, Odometry, self.
odom_callback, queue_size=100)
43
44 def odom_callback(self, odom_msg):
45
46     if(self.in_planning_frt_pass):
47         self.in_planning_frt_pass = False
48         print("\n\tNavigating path now! !")
49
50     orientation_list = [odom_msg.pose.pose.orientation.x, odom_msg.pose.pose.
orientation.y,
51                         odom_msg.pose.pose.orientation.z, odom_msg.pose.pose.
orientation.w]
52
53     (roll, pitch, theta) = tf.transformations.euler_from_quaternion(
orientation_list)
54
55     self.current_x = odom_msg.pose.pose.position.x
56     self.current_y = odom_msg.pose.pose.position.y
57     self.current_heading = theta
58     self.current_vx = odom_msg.twist.twist.linear.x
59
60     speed , steering = self.plan(self.current_x,self.current_y,self.
current_heading, lookahead_distance, P_GAIN, self.waypoints)
61
62     self.actuation.drive.acceleration      =  0.0
63     self.actuation.drive.jerk             =  0.0
64     self.actuation.drive.speed            =  speed
65     self.actuation.drive.steering_angle   =  steering
66     self.actuation.drive.steering_angle_velocity =  0.0
67
68     self.drive_pub.publish(self.actuation)
69

```

```

70         return 0
71
72     def plan(self, pose_x, pose_y, pose_theta, lookahead_distance, vgain, waypoints):
73         :
74         position = np.array([pose_x, pose_y])
75         lookahead_point = get_current_waypoint(waypoints, lookahead_distance,
76         position, pose_theta)
77
78         if lookahead_point is None:
79             return 4.0, 0.0
80
81         speed, steering_angle = get_actuation(pose_theta, lookahead_point, position,
82         lookahead_distance, self.wheelbase)
83         speed = P_GAIN*speed
84
85     def main():
86         rospy.init_node('pure_pursuit_node')
87         pure_pursuit_planner = PurePursuitPlanner()
88         print("\n\tPure pursuit node working!")
89         print("\tFollowing path of ",track," track now.")
90         print("\n\t\tAckerman actuations are being published to /drive topic!!")
91
92         rospy.spin()
93
94 if __name__ == '__main__':
95     main()

```

pp_util.py

```

1 #!/usr/bin/env python3
2
3 import numpy as np
4 from numba import njit
5 from numba.core.target_extension import current_target
6 import rospy
7
8 WAYPOINTS_FILE = rospy.get_param("wpts_path")
9 X_IDX = rospy.get_param("X_IDX")
10 Y_IDX = rospy.get_param("Y_IDX")
11 THEHA_IDX = rospy.get_param("THEHA_IDX")
12 VEL_IDX = rospy.get_param("VEL_IDX")
13 SKIP_ROWS = rospy.get_param("SKIP_ROWS")
14 DELIMITER = rospy.get_param("DELIMITER")
15 max_reacquire = rospy.get_param("max_reacquire")
16
17 """
18 Planner Helpers
19 """
20

```

```

21 # TODO: handle this script as a class
22
23 def load_waypoints():
24     waypoints = np.loadtxt(WAYPOINTS_FILE, delimiter=DELIMITER, skiprows=SKIP_ROWS)
25     return waypoints
26
27 @njit(fastmath=True, cache=True)
28 def nearest_point_on_trajectory(point, trajectory):
29     """
30         Return the nearest point along the given piecewise linear trajectory.
31
32         Same as nearest_point_on_line_segment, but vectorized. This method is quite fast
33         , time constraints should
34         not be an issue so long as trajectories are not insanely long.
35
36         Order of magnitude: trajectory length: 1000 --> 0.0002 second computation
37         (5000fps)
38
39         point: size 2 numpy array
40         trajectory: Nx2 matrix of (x,y) trajectory waypoints
41             - these must be unique. If they are not unique, a divide by 0 error will
42             destroy the world
43             """
44
45         diffs = trajectory[1:,:] - trajectory[:-1,:]
46         l2s = np.linalg.norm(diffs, axis=1)**2
47         # this is equivalent to the elementwise dot product
48         # dots = np.sum((point - trajectory[:-1,:]) * diffs[:, :], axis=1)
49         dots = np.empty((trajectory.shape[0]-1, ))
50         for i in range(dots.shape[0]):
51             dots[i] = np.dot((point - trajectory[i, :]), diffs[i, :])
52         t = dots / l2s
53         t[t<0.0] = 0.0
54         t[t>1.0] = 1.0
55         # t = np.clip(dots / l2s, 0.0, 1.0)
56         projections = trajectory[:-1,:] + (t*diffs.T).T
57         # dists = np.linalg.norm(point - projections, axis=1)
58         dists = np.empty((projections.shape[0], ))
59         for i in range(dists.shape[0]):
60             temp = point - projections[i]
61             dists[i] = np.sqrt(np.sum(temp*temp))
62         min_dist_segment = np.argmin(dists)
63         return projections[min_dist_segment], dists[min_dist_segment], t[
64             min_dist_segment], min_dist_segment
65
66 def get_current_waypoint(waypoints, lookahead_distance, position, theta):
67     wpts = np.vstack((waypoints[:, X_IDX], waypoints[:, Y_IDX])).T
68
69     nearest_point, nearest_dist, t, i = nearest_point_on_trajectory(position, wpts)
70
71     if nearest_dist < lookahead_distance:

```

```

68     lookahead_point, i2, t2 = first_point_on_trajectory_intersecting_circle(
69         position, lookahead_distance, wpts, i+t, wrap=True)
70     if i2 == None:
71         return None
72     current_waypoint = np.empty((3, ))
73     # x, y
74     current_waypoint[0:2] = wpts[i2, :]
75     # speed
76     current_waypoint[2] = waypoints[i, VEL_IDX]
77     return current_waypoint
78 elif nearest_dist < max_reacquire:
79     return np.append(wpts[i, :], waypoints[i, VEL_IDX])
80 else:
81     return None
82
83 @njit(fastmath=True, cache=True)
84 def first_point_on_trajectory_intersecting_circle(point, radius, trajectory, t=0.0,
85 wrap=False):
86     ''' starts at beginning of trajectory, and find the first point one radius away
87     from the given point along the trajectory.
88
89     Assumes that the first segment passes within a single radius of the point
90
91     http://codereview.stackexchange.com/questions/86421/line-segment-to-circle-
92     collision-algorithm
93     '''
94     start_i = int(t)
95     start_t = t % 1.0
96     first_t = None
97     first_i = None
98     first_p = None
99     trajectory = np.asarray(trajectory)
100    for i in range(start_i, trajectory.shape[0]-1):
101        start = trajectory[i, :]
102        end = trajectory[i+1, :]+1e-6
103        V = np.asarray(end - start)
104
105        a = np.dot(V,V)
106        b = 2.0*np.dot(V, start - point)
107        c = np.dot(start, start) + np.dot(point,point) - 2.0*np.dot(start, point) -
108        radius*radius
109        discriminant = b*b-4*a*c
110
111        if discriminant < 0:
112            continue
113            # print "NO INTERSECTION"
114        # else:
115        # if discriminant >= 0.0:
116            discriminant = np.sqrt(discriminant)
117            t1 = (-b - discriminant) / (2.0*a)

```

```

114     t2 = (-b + discriminant) / (2.0*a)
115     if i == start_i:
116         if t1 >= 0.0 and t1 <= 1.0 and t1 >= start_t:
117             first_t = t1
118             first_i = i
119             first_p = start + t1 * v
120             break
121         if t2 >= 0.0 and t2 <= 1.0 and t2 >= start_t:
122             first_t = t2
123             first_i = i
124             first_p = start + t2 * v
125             break
126     elif t1 >= 0.0 and t1 <= 1.0:
127         first_t = t1
128         first_i = i
129         first_p = start + t1 * v
130         break
131     elif t2 >= 0.0 and t2 <= 1.0:
132         first_t = t2
133         first_i = i
134         first_p = start + t2 * v
135         break
136 # wrap around to the beginning of the trajectory if no intersection is found
137 if wrap and first_p is None:
138     for i in range(-1, start_i):
139         start = trajectory[i % trajectory.shape[0], :]
140         end = trajectory[(i+1) % trajectory.shape[0], :]+1e-6
141         V = end - start
142
143         a = np.dot(V,V)
144         b = 2.0*np.dot(V, start - point)
145         c = np.dot(start, start) + np.dot(point,point) - 2.0*np.dot(start, point
146 ) - radius*radius
147         discriminant = b*b-4*a*c
148
149         if discriminant < 0:
150             continue
151         discriminant = np.sqrt(discriminant)
152         t1 = (-b - discriminant) / (2.0*a)
153         t2 = (-b + discriminant) / (2.0*a)
154         if t1 >= 0.0 and t1 <= 1.0:
155             first_t = t1
156             first_i = i
157             first_p = start + t1 * v
158             break
159         elif t2 >= 0.0 and t2 <= 1.0:
160             first_t = t2
161             first_i = i
162             first_p = start + t2 * v
163             break

```

```

164     return first_p, first_i, first_t
165
166 @njit(fastmath=True, cache=True)
167 def get_actuation(pose_theta, lookahead_point, position, lookahead_distance,
168   wheelbase):
169   waypoint_y = np.dot(np.array([np.sin(-pose_theta), np.cos(-pose_theta)]),
170     lookahead_point[0:2]-position)
171   speed = lookahead_point[2]
172   if np.abs(waypoint_y) < 1e-6:
173     return speed, 0.
174   radius = 1/(2.0*waypoint_y/lookahead_distance**2)
175   steering_angle = np.arctan(wheelbase/radius)
176   return speed, steering_angle
177
178
179 def anti_windup(self, input, output, max_out, min_out):
180   if(input >= max_out):
181     output = max_out
182   elif(input<=min_out):
183     output = min_out
184   else:
185     output = input
186
187   return output

```

A.2. Braitenberg Modificado: Miedo

A.2.1. Nodo de algoritmo: Braitenberg Modificado: Miedo

braitenberg_modified.py

```

1 #!/usr/bin/env python3
2
3 import rospy
4 import numpy as np
5
6 from ackermann_msgs.msg import AckermannDriveStamped
7 from sensor_msgs.msg import LaserScan
8 from nav_msgs.msg import Odometry
9
10
11 drive = AckermannDriveStamped()          # Ackerman msgs initialize
12
13 # Catalunya:
14 TURN_DISTANCE = 2.5
15 DELTA_TURN  = 0.0075
16
17 MAX_DRIVE_SPEED = 4
18 MAX_TURN_SPEED = MAX_DRIVE_SPEED/2
19 TURN_STEERING_ANGLE = 0.220
20 n = 6

```

```

21
22 class Braitenberg(object):
23     def __init__(self):
24         self.dir = ""
25         # Publishers:
26         self.drive_pub = rospy.Publisher('/drive', AckermannDriveStamped, queue_size
27         =1)
28
29         # Subscribers:
30         self.scan_sub = rospy.Subscriber('/scan', LaserScan, self.scan_callback,
31         queue_size=100)
32
33     def scan_callback(self, scan_msg):
34         # NOTE: scan measurements are taken in counter-clockwise direction
35         scan_measures = scan_msg.ranges                                     # 10
36         each, zones to split scan measurements
37
38         # Splits ranges in n:
39         LR = np.array_split(scan_measures, n)      # L and R split (n Zones)
40
41         # Separation on L and R avg measures:
42         R_zone = int(n/2) - 1
43         L_zone = int(n/2)
44         R = round( (sum(LR[R_zone])) ) / (len(LR[0])),6)
45         L = round( (sum(LR[L_zone])) ) / (len(LR[0])),6)
46         Center = scan_measures[540]
47
48
49         # ----- GOOOOOOO: -----
50         # +steering: left | -steering: right
51
52         if(L < TURN_DISTANCE):
53             self.dir = L
54             # turn right:
55             drive.drive.steering_angle = (drive.drive.steering_angle - DELTA_TURN)
56             print("\t\t\t\t\tTurning R")
57
58         elif(R <= TURN_DISTANCE):
59             if(self.dir == L):
60                 self.dir = R
61                 drive.drive.steering_angle = 0.0
62
63                 drive.drive.steering_angle = (drive.drive.steering_angle + DELTA_TURN)
64                 print("\t\t\t\t\tTurning L")
65
66         else:
67             drive.drive.steering_angle = 0
68
69         drive.drive.speed = MAX_DRIVE_SPEED
70         # BOUNDING ACTUATION:
71         # Drive speed bounded output upper limit:

```

```

69     if(drive.drive.speed >= MAX_DRIVE_SPEED):
70         drive.drive.speed = MAX_DRIVE_SPEED
71     # Drive speed bounded output lower limit:
72     elif(drive.drive.speed <= MAX_TURN_SPEED):
73         drive.drive.speed = MAX_TURN_SPEED
74
75     # Bounding drive steering upper limit when turnig:
76     if(drive.drive.steering_angle >= TURN_STEERING_ANGLE):
77         drive.drive.steering_angle = TURN_STEERING_ANGLE
78     # Bounding drive steering upper limit when turning:
79     elif(drive.drive.steering_angle <= -TURN_STEERING_ANGLE):
80         drive.drive.steering_angle = -TURN_STEERING_ANGLE
81
82
83     print("\n Velocity: \t\t\t", round(drive.drive.speed, 4),
84           "\n Steering: \t\t\t", round(drive.drive.steering_angle, 4),
85           "\n Left obstacle avg distance: \t", L,
86           "\n Rigth obstacle avg distance: \t", R,
87           "\n Free Space:\t\t\t", round(Center, 4))
88
89
90     # Publishing drive msgs:
91     self.drive_pub.publish(drive)
92     return 0
93
94
95 if __name__ == '__main__':
96     rospy.init_node('braitenberg_modified_node')
97     f1tenths_CERLab_agent = Braitenberg()
98     rospy.spin()

```

A.3. Mezcla de comportamientos

A.3.1. Planificador Local

local_braitenberg_modified.py

```

1 #!/usr/bin/env python3
2
3 import rospy
4 import numpy as np
5
6 from ackermann_msgs.msg import AckermannDriveStamped
7 from rospy.client import get_param
8 from sensor_msgs.msg import LaserScan
9 from nav_msgs.msg import Odometry
10
11 drive_topic      =    rospy.get_param("drive_local")
12 odom_topic       =    rospy.get_param("odom_topic")
13 track            =    rospy.get_param("track")
14 n                =    rospy.get_param("sectors")

```

```

15
16 # Catalunya:
17 TURN_DISTANCE = rospy.get_param("turn_dist")
18 DELTA_TURN = rospy.get_param("delta_turn")
19
20 drive = AckermannDriveStamped()           # Ackerman msgs initialize
21
22
23
24 MAX_DRIVE_SPEED = 4
25 MAX_TURN_SPEED = MAX_DRIVE_SPEED/2
26 TURN_STEERING_ANGLE = rospy.get_param("max_steering_angle")
27
28
29 class BraitenbergMod(object):
30     def __init__(self):
31         self.dir = ""
32         # Publishers:
33         self.drive_pub = rospy.Publisher(drive_topic, AckermannDriveStamped,
queue_size=100)
34
35         # Subscribers:
36         self.scan_sub = rospy.Subscriber('/scan', LaserScan, self.scan_callback,
queue_size=100)
37
38     def scan_callback(self, scan_msg):
39         # NOTE: scan measurements are taken in counter-clockwise direction
40         scan_measures = scan_msg.ranges          # 22.5
41         each, zones to split scan measurements
42
43         # Splits ranges in n:
44         LR = np.array_split(scan_measures, n)      # L and R split (n Zones)
45
46         # Separation on L and R avg measures:
47         R_zone = int(n/2) - 1
48         L_zone = int(n/2)
49         R = round( (sum(LR[R_zone])) ) / (len(LR[0])),6)
50         L = round( (sum(LR[L_zone])) ) / (len(LR[0])),6)
51         Center = scan_measures[540]
52
53         # ----- GOOOOOOO: -----
54         # +steering: left | -steering: right
55
56         if(L < TURN_DISTANCE):
57
58             # turn right:
59             drive.drive.steering_angle = (drive.drive.steering_angle - DELTA_TURN)
60             print("\t\t\t\t\tTurning R")
61
62         elif(R <= TURN_DISTANCE):

```

```

63
64
65     drive.drive.steering_angle = (drive.drive.steering_angle + DELTA_TURN)
66     print("\tTurning L")
67
68 else:
69     drive.drive.steering_angle = 0
70
71 drive.drive.speed = MAX_DRIVE_SPEED
72 # BOUNDING ACTUATION:
73 # Drive speed bounded output upper limit:
74 if(drive.drive.speed >= MAX_DRIVE_SPEED):
75     drive.drive.speed = MAX_DRIVE_SPEED
76 # Drive speed bounded output lower limit:
77 elif(drive.drive.speed <= MAX_TURN_SPEED):
78     drive.drive.speed = MAX_TURN_SPEED
79
80 # Bounding drive steering upper limit when turnig:
81 if(drive.drive.steering_angle >= TURN_STEERING_ANGLE):
82     drive.drive.steering_angle = TURN_STEERING_ANGLE
83 # Bounding drive steering upper limit when turning:
84 elif(drive.drive.steering_angle <= -TURN_STEERING_ANGLE):
85     drive.drive.steering_angle = -TURN_STEERING_ANGLE
86
87
88 print("\n Velocity: \t\t\t", round(drive.drive.speed, 4),
89      "\n Steering: \t\t\t", round(drive.drive.steering_angle, 4),
90      "\n Left obstacle avg distance: \t", L,
91      "\n Rigth obstacle avg distance: \t", R,
92      "\n Free Space:\t\t\t", round(Center, 4))
93
94
95 # Publishing drive msgs:
96 self.drive_pub.publish(drive)
97 return 0
98
99
100 def main():
101     rospy.init_node('braitenbergPlus_node')
102     braitenberg_mod = BraitenbergMod()
103     print("\n\tbraitenberg node working!")
104     print("\tFollowing path of ",track ," track now.")
105     print("\n\t\tAckerman actuations are beeing published to /drive topic!!")
106
107     rospy.spin()
108
109 if __name__ == '__main__':
110     main()

```

A.3.2. Planificador Global

global_pure_pursuit.py

```

1 #!/usr/bin/env python3
2 from pp_utils import get_actuation, load_waypoints, get_current_waypoint
3 import rospy
4 import numpy as np
5 from geometry_msgs.msg import PoseStamped
6 from sensor_msgs.msg import LaserScan
7 from ackermann_msgs.msg import AckermannDriveStamped
8 from nav_msgs.msg import Odometry
9 import tf
10
11 wheel_base      =  rospy.get_param("wheelbase")
12 lookahead_distance =  rospy.get_param("lookahead_distance")
13 P_GAIN          =  rospy.get_param("P_GAIN")
14 track           =  rospy.get_param("track")
15
16 drive_topic     =  rospy.get_param("drive_global")
17 odom_topic       =  rospy.get_param("odom_topic")
18
19 class PurePursuitPlanner():
20     """
21     Example Planner
22     """
23     def __init__(self):
24
25         self.wheelbase      =  wheel_base
26         self.current_x      =  0.0
27         self.current_y      =  0.0
28         self.current_heading =  0.0
29         self.current_vx     =  0.0
30         self.curvature      =  0.0
31
32         self.actuation       =  AckermannDriveStamped()
33
34         self.waypoints = load_waypoints()
35
36         self.in_planning_frt_pass = True
37
38     # Publishers:
39     self.drive_pub   =  rospy.Publisher(drive_topic, AckermannDriveStamped,
40                                         queue_size=100)
41
42     # Subscribers:
43     self.odom_sub    =  rospy.Subscriber(odom_topic, Odometry, self.
44                                         odom_callback, queue_size=100)
45
46     def odom_callback(self, odom_msg):
47

```

```

46     if(self.in_planning_frt_pass):
47         self.in_planning_frt_pass = False
48         print("\n\tNavigating path now! !")
49
50     orientation_list = [odom_msg.pose.pose.orientation.x, odom_msg.pose.pose.
51                         orientation.y,
52                         orientation.z, odom_msg.pose.pose.orientation.w]
53
54     (roll, pitch, theta) = tf.transformations.euler_from_quaternion(
55                           orientation_list)
56
57     self.current_x = odom_msg.pose.pose.position.x
58     self.current_y = odom_msg.pose.pose.position.y
59     self.current_heading = theta
60     self.current_vx = odom_msg.twist.twist.linear.x
61
62     speed , steering = self.plan(self.current_x,self.current_y,self.
63                                   current_heading, lookahead_distance, P_GAIN, self.waypoints)
64
65     self.actuation.drive.acceleration = 0.0
66     self.actuation.drive.jerk = 0.0
67     self.actuation.drive.speed = speed
68     self.actuation.drive.steering_angle = steering
69     self.actuation.drive.steering_angle_velocity = 0.0
70
71     self.drive_pub.publish(self.actuation)
72
73     return 0
74
75 def plan(self, pose_x, pose_y, pose_theta, lookahead_distance, vgain, waypoints):
76 :
77     position = np.array([pose_x, pose_y])
78     lookahead_point = get_current_waypoint(waypoints, lookahead_distance,
79                                           position, pose_theta)
80
81     if lookahead_point is None:
82         return 4.0, 0.0
83
84     speed, steering_angle = get_actuation(pose_theta, lookahead_point, position,
85                                           lookahead_distance, self.wheelbase)
86     speed = P_GAIN*speed
87
88     return speed, steering_angle
89
90
91 def main():
92     rospy.init_node('pure_pursuit_node')
93     pure_pursuit_planner = PurePursuitPlanner()
94     print("\n\tPure pursuit node working!")
95     print("\tFollowing path of ",track ," track now.")

```

```

90     print("\n\t\tAckerman actuations are being published to /drive topic!!")
91
92     rospy.spin()
93
94 if __name__ == '__main__':
95     main()

```

A.3.3. Nodo de algoritmo: mezcla de comportamientos

PP_Brait_modified_nav_node.py

```

1 #!/usr/bin/env python3
2
3 from sys import path
4 import rospy
5 from ackermann_msgs.msg import AckermannDriveStamped
6 from nav_msgs.msg import Odometry
7
8 PLANNER_TYPE = rospy.get_param("PLANNER_TYPE")
9 track          = rospy.get_param("track")
10 drive_global  = rospy.get_param("drive_global")
11 drive_local   = rospy.get_param("drive_local")
12
13 drive = rospy.get_param("drive")
14 odom_topic = rospy.get_param("odom_topic")
15
16 GSW = rospy.get_param("global_speed_weight")
17 LSW = rospy.get_param("local_speed_weight")
18 GSTRW = rospy.get_param("global_steer_weight")
19 LSTRW = rospy.get_param("local_steer_weight")
20
21 class PathPlanningNode:
22     def __init__(self):
23         self.drv_global_speed      = 0
24         self.drv_global_steering   = 0
25         self.drv_local_speed       = 0
26         self.drv_local_steering   = 0
27         rospy.init_node("path_planner_node")
28         rospy.loginfo("Starting PathPlanningNode as path_planner_node.")
29
30         self.global_drive_sub =    rospy.Subscriber(drive_global,
31 AckermannDriveStamped, self.global_drive_callback)
32         self.local_drive_sub =   rospy.Subscriber(drive_local, AckermannDriveStamped,
33 self.local_drive_callback)
34
35         # Subscribers:
36         self.drive_pub =    rospy.Publisher(drive, AckermannDriveStamped, queue_size
37 =100)
38
39     def switch_planner(self):

```

```

39
40     return 0
41
42 def dynamic_weight(self):
43     global_speed_weight      = GSW
44     local_speed_weight       = LSW
45     global_steer_weight      = GSTRW
46     local_steer_weight       = LSTRW
47
48
49     return global_speed_weight, local_speed_weight, global_steer_weight,
50     local_steer_weight
51
52 def global_drive_callback(self, g_msg):
53     self.drv_global_speed      = g_msg.drive.speed
54     self.drv_global_steering    = g_msg.drive.steering_angle
55
56     self.drive_handler()
57
58     return 0
59
60 def local_drive_callback(self, l_msg):
61     self.drv_local_speed        = l_msg.drive.speed
62     self.drv_local_steering      = l_msg.drive.steering_angle
63     return 0
64
65 def drive_handler(self):
66     drive_msg = AckermannDriveStamped()
67
68     global_speed_weight, local_speed_weight, global_steer_weight,
69     local_steer_weight = self.dynamic_weight()
70
71     speed_overall = global_speed_weight * self.drv_global_speed +
72     local_speed_weight * self.drv_local_speed
73     steer_overall = global_steer_weight * self.drv_global_steering +
74     local_steer_weight * self.drv_local_steering
75
76     drive_msg.drive.speed = speed_overall
77     drive_msg.drive.steering_angle = steer_overall
78
79     self.drive_pub.publish(drive_msg)
80     return 0
81
82 def main():
83     pathPlanningNode = PathPlanningNode()
84     print("\n\tpathPlanning node working!")
85     print("\tFollowing path of ",track," track now.")
86     print("\n\t\tAckerman actuations are being published to /drive topic!!")
87
88     rospy.spin()

```

```

86
87 if __name__ == '__main__':
88     main()

```

A.3.4. Archivos de configuración y launch file

PP_Brait_modified_planner_integration.launch

```

1 <?xml version="1.0"?>
2 <launch>
3   <!--
4     =====
5       Launch node for pure pursuit algorithm. You can choose between track params
6       depending
7       on map your simulation is using.
8       -> Change TRACK default value for desired track. List of available tracks:
9           Example
10          Catalunya
11          IMS
12          MoscowRaceway
13          Monza
14          Oschersleben
15          Skirk
16          Zandvoort
17          Silverstone
18          Blank
19
20   -> Change ALGORITHM default value for desired navigation. List of available
21       algorithms:
22           PP : pure_pursuit
23           BTG : braitenberg
24           BTGP : baitenberg+
25           PathPlanner : path_planner -> pure_pursuit_&
26           _braitenberg -> (Path tracking + Obstacle avoidance)
27
28
29   -->
30   <!--
31   -->
32   <!-- track settings for algorithm params: -->
33   <arg name="TRACK" default="Example"/>
34
35   <!-- algorithm settings: -->
36   <arg name="ALGORITHM_ID" default="PP"/>
37   <arg name="ALGORITHM" default="pure_pursuit"/>
38
39   <!--
40   -->

```

```

34      <rosparam file="$(find f110_race_nodes)/config/track_algorithm_params/$(arg
35 TRACK)_$(arg ALGORITHM_ID)_params.yaml" command="load"/>
36      <rosparam file="$(find f110_race_nodes)/config/planners_config_$(arg TRACK).yaml"
37      command="load"/>
38      <!-- ===== -->
39      <!--
40      <node pkg="f110_race_nodes" type="gl_nodes.sh" name="gl_nodes"/>
41      -->
42      <node pkg="f110_race_nodes" type="raceline_viz.py" name="raceline_viz_node"/>
43      >
44      <node pkg="f110_race_nodes" type="path_plotter.py" name="path_plotter_node"/>
45      >
46      <node pkg="f110_race_nodes" type="local_braitenberg_modified.py" name="local_braitenberg_modified"/>
        <node pkg="f110_race_nodes" type="global_pure_pursuit.py" name="global_pure_pursuit"/>
        <node pkg="f110_race_nodes" type="PP_Brait_modified_nav_node.py" name="PP_Brait_modified_nav_node"/>
    </launch>

```

config.yaml

```

1 # placeholder for config parameters
2
3 drive_global      : "/drive"          # path planner: pure pursuit
4 drive_local       : "/drive_local"     # obstacle avoider: braitenberg
5 drive             : "/drive"          # path planner drive
6 odom_topic        : "/odom"           # odometry topic
7 path_topic        : "/path"           # path plot topic
8 raceline_path     : "/raceline_path"   # raceline plot topic
9 steer_curvature  : "/steer_curvature" # curvature plot topic

```

planners_config_Catalunya.yaml

```

1 # placeholder for config parameters
2
3 PLANNER_TYPE      : "weighted"        # could be: "weighted" or "switched"
4
5 drive_global      : "/drive_global"    # path planner: pure pursuit
6 drive_local       : "/drive_local"     # obstacle avoider: braitenberg
7 drive             : "/drive"          # path planner drive
8 odom_topic        : "/odom"           # odometry topic
9 scan_topic        : "/scan"           # LIDAR reading topic
10 path_topic        : "/path"           # path plot topic
11 raceline_path     : "/raceline_path"   # raceline plot topic
12 steer_curvature  : "/steer_curvature" # curvature plot topic
13
14
15 # Brait params:
16 sectors          : 64 #64 #22
17 turn_dist         : 2.5 #2.5 #2.5

```

```

18 delta_turn      : 0.01 #0.01
19
20 # Pure pursuit params:
21 lookahead_distance : 2 #1.0 # good 1.5 # 3 FOR LONG lad
22 P_GAIN           : 0.75 #0.75 # good
23
24
25 # G + L planner:
26   #speed:
27 global_speed_weight : 1.0 #1.0
28 local_speed_weight  : 0.0 #0.0
29   # steering:
30 global_steer_weight : 1 #1
31 local_steer_weight  : 0.85 #0.85
32
33 #global_speed_weight : 1
34 #local_speed_weight  : 1
35 #global_steer_weight : 1
36 #local_steer_weight  : 1

```

planners_config_Example.yaml

```

1 # placeholder for config parameters
2
3 PLANNER_TYPE      : "weighted"          # could be: "weighted" or "switched"
4
5 drive_global       : "/drive_global"     # path planner: pure pursuit
6 drive_local        : "/drive_local"      # obstacle avoider: braitenberg
7 drive              : "/drive"            # path planner drive
8 odom_topic         : "/odom"             # odometry topic
9 scan_topic         : "/scan"             # LIDAR reading topic
10 path_topic         : "/path"              # path plot topic
11 raceline_path      : "/raceline_path"    # raceline plot topic
12 steer_curvature   : "/steer_curvature" # curvature plot topic
13
14 #####33
15 # Braint params:
16 sectors           : 64 #64 #22
17 turn_dist          : 2.5 #2.5 #2.5
18 delta_turn         : 0.01 #0.01
19
20 # Pure pursuit params:
21 lookahead_distance : 2 #2 ok # 3 FOR LONG lad
22 P_GAIN             : 0.75 #0.75 # good
23
24
25 # G + L planner:
26   #speed:
27 global_speed_weight : 1.0 #1.0
28 local_speed_weight  : 0.0 #0.0
29   # steering:
30 global_steer_weight : 1.0 #1

```

```
31 local_steer_weight      : 0.85 #0.85
```

A.4. Misceláneos generales, launch files y archivos de configuración

A.4.1. path_plotter.py

```

1 #!/usr/bin/env python3
2 import rospy
3
4 from geometry_msgs.msg import PoseStamped
5 from nav_msgs.msg import Odometry
6 from nav_msgs.msg import Path
7
8 path           = rospy.get_param("path_topic")
9 odom_topic     = rospy.get_param("odom_topic")
10
11 class OdomPathPlotter(object):
12     def __init__(self):
13         self.pth = Path()
14         self.odom_sub = rospy.Subscriber(odom_topic, Odometry, self.
15                                         odom_to_path, queue_size=1)
16         self.path_pub = rospy.Publisher(path, Path, queue_size=1)
17
18     def odom_to_path(self, data):
19         self.pth.header = data.header
20         pose = PoseStamped()
21         pose.header = data.header
22         pose.pose = data.pose.pose
23         self.pth.poses.append(pose)
24
25         self.path_pub.publish(self.pth)
26
27         if(self.pth.header.seq > 1):
28             self.pth.poses.pop(0)
29
30         return 0
31
32 def main():
33     rospy.init_node('path_plotter_node')
34     plotter = OdomPathPlotter()
35     print("\n\tPlotting to Path node working!")
36     print("\tVisualizing Odometry data on RViz now.")
37     print("\n\t\tOdometry path is being published to /path topic!!\n")
38     rospy.spin()
39
40
41 if __name__ == '__main__':
42     main()
```

A.4.2. raceline_viz.py

```

1 #!/usr/bin/env python3
2
3 from geometry_msgs.msg import PoseStamped
4 from nav_msgs.msg import Odometry
5 from nav_msgs.msg import Path
6 import numpy as np
7 import rospy
8
9 track          =    rospy.get_param("track")
10 odom_topic     =    rospy.get_param("odom_topic")
11 raceline_path  =    rospy.get_param("raceline_path")
12
13 class RacelineDataPublisher(object):
14     def __init__(self):
15
16         self.wpts          =    rospy.get_param("wpts_path")
17         self.delimiter      =    rospy.get_param("DELIMITER")
18         self.skip           =    rospy.get_param("SKIP_ROWS")
19         self.X_IDX          =    rospy.get_param("X_IDX")
20         self.Y_IDX          =    rospy.get_param("Y_IDX")
21         self.i = 0
22
23         self.raceline_path = PoseStamped()
24         self.pth = Path()
25
26         self.odom_sub       =    rospy.Subscriber(odom_topic, Odometry, self.
27 racelice_viz_odomCallback, queue_size=10)
28         self.raceline_path_pub = rospy.Publisher(raceline_path, Path, queue_size=10)
29
30         self.wpts_raceline = np.loadtxt(self.wpts, delimiter=self.delimiter,
31 skiprows=self.skip)
32         self.e = len(self.wpts_raceline)
33         self.zeros = np.zeros( ( self.e , 1 ) , dtype=float)
34
35         self.raceline_arr = np.vstack((self.wpts_raceline[:, self.X_IDX], self.
36 wpts_raceline[:, self.Y_IDX])).T
37         self.raceline_arr = np.append(self.raceline_arr, self.zeros, axis=1)
38
39     def racelice_viz_odomCallback(self, data):
40
41         path_stamped = PoseStamped()
42         path_stamped.header = data.header
43         # pose position ( x,y,z ):
44         path_stamped.pose.position.x = self.raceline_arr[self.i][0]
45         path_stamped.pose.position.y = self.raceline_arr[self.i][1]
46         path_stamped.pose.position.z = self.raceline_arr[self.i][2]
47
48         # pose orientation:
49         path_stamped.pose.orientation.x = 0

```

```

47     path_stamped.pose.orientation.y = 0
48     path_stamped.pose.orientation.z = 0
49     path_stamped.pose.orientation.w = 1
50
51     # path data to publish:
52     self.pth.header = data.header
53     self.pth.poses.append(path_stamped)
54
55
56     self.raceline_path_pub.publish(self.pth)
57     #print("\n ----- \n", self.pth)
58
59     if(self.i < len(self.raceline_arr)-3):
60         self.i = self.i + 1
61     else:
62         self.i = 0
63
64     if(self.pth.header.seq > 1):
65         self.pth.poses.pop(0)
66
67     return 0
68
69 def main():
70     rospy.init_node('raceline_plotter_node')
71     plotter = RacelineDataPublisher()
72     print("\n\tRaceline Visualization node working!")
73     print("\tVisualizing raceline for path tracking on RViz now.")
74     print("\tPlotted track raceline:\t", track)
75     print("\n\t\tRaceline is being published to /raceline_path topic!!\n")
76     rospy.spin()
77     return 0
78
79
80 if __name__ == '__main__':
81     main()

```

A.4.3. raceline_viz.py

```

1 #!/usr/bin/env python3
2
3 from geometry_msgs.msg import PoseStamped
4 from nav_msgs.msg import Odometry
5 from nav_msgs.msg import Path
6 import numpy as np
7 import rospy
8
9 track        =    rospy.get_param("track")
10 odom_topic   =    rospy.get_param("odom_topic")
11 raceline_path =    rospy.get_param("raceline_path")
12
13 class RacelineDataPublisher(object):
14     def __init__(self):

```

```

15     self.wpts          = rospy.get_param("wpts_path")
16     self.delimiter     = rospy.get_param("DELIMITER")
17     self.skip          = rospy.get_param("SKIP_ROWS")
18     self.X_IDX         = rospy.get_param("X_IDX")
19     self.Y_IDX         = rospy.get_param("Y_IDX")
20     self.i = 0
21
22     self.raceline_path = PoseStamped()
23     self.pth = Path()
24
25     self.odom_sub      = rospy.Subscriber(odom_topic, Odometry, self.
26 racelice_viz_odomCallback, queue_size=10)
27     self.raceline_path_pub = rospy.Publisher(raceline_path, Path, queue_size=10)
28
29     self.wpts_raceline = np.loadtxt(self.wpts, delimiter=self.delimiter,
30 skiprows=self.skip)
31     self.e = len(self.wpts_raceline)
32     self.zeros = np.zeros( ( self.e , 1 ) , dtype=float)
33
34     self.raceline_arr = np.vstack((self.wpts_raceline[:, self.X_IDX], self.
35 wpts_raceline[:, self.Y_IDX])).T
36     self.raceline_arr = np.append(self.raceline_arr, self.zeros, axis=1)
37
38 def racelice_viz_odomCallback(self, data):
39
40     path_stamped = PoseStamped()
41     path_stamped.header = data.header
42     # pose position ( x,y,z):
43     path_stamped.pose.position.x = self.raceline_arr[self.i][0]
44     path_stamped.pose.position.y = self.raceline_arr[self.i][1]
45     path_stamped.pose.position.z = self.raceline_arr[self.i][2]
46
47     # pose orientation:
48     path_stamped.pose.orientation.x = 0
49     path_stamped.pose.orientation.y = 0
50     path_stamped.pose.orientation.z = 0
51     path_stamped.pose.orientation.w = 1
52
53     # path data to publish:
54     self.pth.header = data.header
55     self.pth.poses.append(path_stamped)
56
57     self.raceline_path_pub.publish(self.pth)
58     #print("\n ----- -----\n", self.pth)
59
60     if(self.i < len(self.raceline_arr)-3):
61         self.i = self.i + 1
62     else:
63         self.i = 0

```

```

63         if(self.pth.header.seq > 1):
64             self.pth.poses.pop(0)
65
66             return 0
67
68 def main():
69     rospy.init_node('raceline_plotter_node')
70     plotter = RacelineDataPublisher()
71     print("\n\tRaceline Visualization node working!")
72     print("\tVisualizing raceline for path tracking on RViz now.")
73     print("\tPlotted track raceline:\t", track)
74     print("\n\t\tRaceline is being published to /raceline_path topic!!\n")
75     rospy.spin()
76     return 0
77
78
79
80 if __name__ == '__main__':
81     main()

```

A.4.4. autonomous_navigation_nodes.launch

```

1  <?xml version="1.0"?>
2  <launch>
3      <!--
4          =====
5          Launch node for pure pursuit algorithm. You can choose between track params
6          depending
7          on map your simulation is using.
8          -> Change TRACK default value for desired track. List of available tracks:
9              Example *
10             Catalunya **
11             Zandvoort ***
12             IMS
13             MoscowRaceway
14             Monza
15             Oschersleben
16             Skirk
17             Silverstone
18             Blank
19
20             -> Change ALGORITHM default value for desired navigation. List of available
21             algorithms:
22                 PP      :    pure_pursuit
23                 BTG     :    braitenberg
24                 BTGP    :    baitennberg+
25
26             ->
27             <!--
28             =====

```

```

-->
24     <!-- track settings for algorithm params: -->
25     <arg name="TRACK" default="Catalunya"/>
26
27     <!-- algorithm settings: -->
28     <arg name="ALGORITHM_ID" default="PP"/>
29     <arg name="ALGORITHM" default="pure_pursuit"/>
30
31     <!--
=====
-->
32     <rosparam file="$(find f110_race_nodes)/config/track_algorithm_params/$(arg
33 TRACK)_$(arg ALGORITHM_ID)_params.yaml" command="load"/>
34     <rosparam file="$(find f110_race_nodes)/config/config.yaml" command="load"/>
35     <!-- ===== -->
36     <node pkg="f110_race_nodes" type="path_plotter.py" name="path_plotter_node"/
37     >
38     <node pkg="f110_race_nodes" type="raceline_viz.py" name="raceline_viz_node"/
39     >
40     <node pkg="f110_race_nodes" type="path_plotter.py" name="path_plotter_node"/
41     >
42     <node pkg="f110_race_nodes" type="$(arg ALGORITHM).py" name="$(arg ALGORITHM
43 )_node"/>
44     -->
45
46     <node pkg="f110_race_nodes" type="$(arg ALGORITHM_ID)_nodes.sh" name="$(arg
47 ALGORITHM)_nodes" /> <!-- you can launch executables as nodes :) -->
48
49 </launch>

```

A.4.5. raceline_draw.launch

```

1 <?xml version="1.0"?>
2 <launch>
3     <!--
=====
4         Launch node for pure pursuit algorithm. You can choose between track params
5         depending
6         on map your simulation is using.
7         -> Change TRACK default value for desired track. List of available tracks:
8             Example
9                 Catalunya
10                IMS
11                MoscowRaceway
12                Monza
13                Oschersleben
14                Skirk
15                Zandvoort
16                Silverstone
17                Blank

```

```

17      -> Change ALGORITHM default value for desired navigation. List of available
18      algorithms:
19          PP      :      pure Pursuit
20          BTG     :      Braitenberg
21          BTGP    :      baitenbergs+
22          PP&BTG  :      pure Pursuit & braitenberg -> (Path tracking +
23          Obstable avoidance)
24
25      =====
26      -->
27      <!-->
28      -->
29      <!-- track settings: -->
30      <arg name      = "TRACK"    default      = "Catalunya"/>
31
32      <!-- algorithm settings: -->
33      <arg name      = "ALGORITHM" default      = "PP"/>
34
35      <!-->
36      =====
37
38      <rosparam file="$(find f110_race_nodes)/config/$(arg TRACK)_$(arg ALGORITHM)
39      _params.yaml" command="load"/>
40      <rosparam file="$(find f110_race_nodes)/config/config.yaml" command="load"/>
41      <!-- ===== -->
42
43      <node pkg="f110_race_nodes" type="raceline_viz.py" name="raceline_viz" />
44      <!-- you can lauch executables as nodes :) -->
45
46  </launch>

```

A.4.6. Archivos de parámetros para algoritmos de Persecución Pura y Braitenberg Modificado: Miedo

```

1 # Configuration file for Braitenberg in Catalunya map circuit for
2 # parameter loading
3
4 X_IDX           : 1
5 Y_IDX           : 2
6 THEHA_IDX       : 3
7 VEL_IDX         : 5
8 SKIP_ROWS       : 1
9 DELIMITER        : ";"
10 max_reacquire   : 20.0
11

```

```

12 ackerman_wheel_base : 0.3302
13
14 lookahead_distance : 1.0 # good 1.5 # 3 FOR LONG lad
15 P_GAIN : 1.0 # good 1

1 # Configuration file for pure pursuit in Catalunya map circuit for
2 # parameter loading
3
4 X_IDX : 1
5 Y_IDX : 2
6 THEHA_IDX : 3
7 VEL_IDX : 5
8 SKIP_ROWS : 1
9 DELIMITER : ";""
10 max_reacquire : 20.0
11
12 ackerman_wheel_base : 0.3302
13
14 lookahead_distance : 2 #1.0 # good 1.5 # 3 FOR LONG lad
15 P_GAIN : 0.75 #0.75 # good

1 # Configuration file for Braitenberg in Catalunya map circuit for
2 # parameter loading
3
4 X_IDX : 1
5 Y_IDX : 2
6 THEHA_IDX : 3
7 VEL_IDX : 5
8 SKIP_ROWS : 1
9 DELIMITER : ";""
10 max_reacquire : 20.0
11
12 ackerman_wheel_base : 0.3302
13
14 lookahead_distance : 2 #2 #1.0 # good 1.5 # 3 FOR LONG lad
15 P_GAIN : 0.75 #0.75 # good

```