



## Trabajo Fin de Grado

# **Implementación en Python del modelo DOVS para la navegación en entornos dinámicos**

Python implementation of the DOVS model for  
navigation in dynamic environments

Autor

**Jorge Martínez Gil**

Directores

Diego Martínez Baselga

Luis Montano Gella

ESCUELA DE INGENIERÍA Y ARQUITECTURA  
2023





## **DECLARACIÓN DE AUTORÍA Y ORIGINALIDAD**

(Este documento debe acompañar al Trabajo Fin de Grado (TFG)/Trabajo Fin de Máster (TFM) cuando sea depositado para su evaluación).

D./Dª. \_\_\_\_\_,

con nº de DNI \_\_\_\_\_ en aplicación de lo dispuesto en el art. 14 (Derechos de autor) del Acuerdo de 11 de septiembre de 2014, del Consejo de Gobierno, por el que se aprueba el Reglamento de los TFG y TFM de la Universidad de Zaragoza,

Declaro que el presente Trabajo de Fin de (Grado/Máster) \_\_\_\_\_, (Título del Trabajo)

---

---

---

---

---

---

---

es de mi autoría y es original, no habiéndose utilizado fuente sin ser citada debidamente.

Zaragoza, \_\_\_\_\_

Fdo: \_\_\_\_\_



## **AGRADECIMIENTOS**

Quiero agradecer la ayuda que me han prestado en la realización de este trabajo de fin de Grado varias personas. Entre ellas, y en primer lugar, a mi director Diego Martínez, al codirector Luis Montano y a Luis Riazuelo por todo lo que me han ayudado y enseñado durante la realización de este trabajo, siempre han estado dispuestos a dedicarme una parte de su tiempo, y sin su ayuda me hubiera sido muy difícil la realización del mismo.

También a mis compañeros, especialmente a Hugo, que me han aportado alguna idea en momentos de duda, y a mi familia que siempre me han apoyado.

Y por último agradecer a la Universidad de Zaragoza, y a la Escuela de Ingeniería la oportunidad que me ha brindado de obtener una formación de calidad.



# RESUMEN

En la actualidad los robots deben coexistir o cooperar con los seres humanos u otros vehículos en movimiento. Estas interacciones se dan en escenarios que cambian constantemente. Por ello la investigación en las aplicaciones de los sistemas robóticos autónomos es un campo prioritario.

La planificación del movimiento en escenarios dinámicos requiere la capacidad de predecir la evolución futura de los obstáculos, a fin de planificar y ejecutar los movimientos más seguros y rápidos posibles para el robot.

El *Dynamic Object Velocity Space* (DOVS) [1] es un modelo que representa el dinamismo de un entorno. El DOVS mapea el entorno en el espacio de velocidad del robot y determina las velocidades seguras y factibles para él en cada momento dentro del horizonte temporal considerado, permitiendo así la planificación de movimientos libres de colisiones.

El presente trabajo de fin de grado implementa en Python el modelo DOVS, poniendo mucha importancia en que el código que sea fácilmente entendible y extensible. Esto se debe a las posibilidades de investigación que esta implementación abre sobre este modelo por la claridad, modularidad, extensibilidad y simplicidad que aporta Python y un buen diseño. Esto permite a personas ajenas a la implementación aprovecharse del modelo, pudiendo escalar el proyecto. Este trabajo es una base para ampliar en el futuro el modelo DOVS a una navegación de drones en 3-D y navegación multi-robot con toma de decisiones compartida.

El correcto funcionamiento del código se validó mediante una evaluación exhaustiva del modelo DOVS en una variedad de escenarios dinámicos. Se realizaron pruebas en entornos con diferentes configuraciones de obstáculos en movimiento, lo que permitió obtener resultados satisfactorios y validar la funcionalidad del modelo.

Con el modelo DOVS implementado y validado, se abre un amplio abanico de posibilidades para desarrollar estrategias de planificación y navegación de robots. Estas estrategias pueden aprovechar la capacidad del modelo para predecir la evolución de los obstáculos y generar trayectorias seguras y eficientes. Como resultado, se logra una navegación autónoma eficiente y segura, evitando colisiones en entornos en constante cambio.



# Índice

<b>1. Introducción y objetivos</b>	<b>1</b>
1.1. Motivación y contexto . . . . .	1
1.2. Objetivos y tareas . . . . .	2
1.3. Contenido de la memoria . . . . .	4
<b>2. DOVS</b>	<b>5</b>
2.1. Definición . . . . .	5
2.2. El robot . . . . .	6
2.2.1. Definición . . . . .	7
2.2.2. Trayectorias . . . . .	7
2.2.3. Ventana de velocidad . . . . .	8
2.3. Obstáculos . . . . .	10
2.3.1. Trayectorias de los obstáculos . . . . .	13
2.4. Cálculo del DOVS . . . . .	16
2.4.1. Cálculo de los puntos de colisión . . . . .	18
2.4.2. Cálculo de las velocidades . . . . .	21
<b>3. Implementación de la librería DOVS</b>	<b>27</b>
3.1. Diseño . . . . .	27
3.1.1. Diagrama de paquetes . . . . .	28
3.1.2. Diagrama de clases . . . . .	31
3.2. Implementación del DOVS . . . . .	31
<b>4. Validación experimental</b>	<b>33</b>
4.1. Experimento 1. Pasar por detrás de un obstáculo . . . . .	33
4.2. Experimento 2. Pasar por delante de dos obstáculos . . . . .	34
4.3. Experimento 3. Colisión con obstáculo, trayectoria curvilínea . . . . .	36
4.4. Experimento 4. Colisión con obstáculo, trayectoria rectilínea . . . . .	37
4.5. Experimento 5. Pasar por delante de 2 obstáculos y detrás de 1 . . . . .	39
4.6. Experimento 6. Pasar por detrás con 3 obstáculos . . . . .	41

4.7. Experimento 7. Llega a la meta con trayectoria curvilínea pasando por delante de un obstáculo y por detrás de otro . . . . .	42
4.8. Experimento 8. Cinco obstáculos . . . . .	44
4.9. Experimento 9. Colisión por detrás del robot . . . . .	45
4.10. Experimento 10. Colisión frontal . . . . .	46
4.11. Evaluación de resultados . . . . .	47
4.12. Problemas encontrados y soluciones . . . . .	49
<b>5. Conclusiones</b>	<b>55</b>
5.1. Conclusiones técnicas . . . . .	55
5.2. Trabajo futuro . . . . .	55
<b>6. Bibliografía</b>	<b>57</b>
<b>Lista de Figuras</b>	<b>59</b>
<b>Lista de Tablas</b>	<b>63</b>
<b>Anexos</b>	<b>64</b>
<b>A. Gestión del trabajo</b>	<b>67</b>
A.1. Horas invertidas . . . . .	67
A.2. Diagrama de Gantt . . . . .	67
<b>B. Código</b>	<b>69</b>
B.1. Calculo del DOVS . . . . .	69
B.2. Calculo de los puntos de colision . . . . .	70
B.3. Calculo de las velocidades . . . . .	71
<b>C. Diagramas</b>	<b>75</b>
C.1. Diagrama de clases . . . . .	75
C.2. Diagrama de secuencias del cálculo del DOVS . . . . .	78
C.3. Diagrama de secuencias del cálculo de los puntos de colisión . . . . .	79
C.4. Diagrama de flujo, cálculo de las velocidades de colisión . . . . .	80

# Capítulo 1

## Introducción y objetivos

### 1.1. Motivación y contexto

Las aplicaciones de los sistemas robóticos autónomos son una prioridad de investigación reconocida, especialmente en contextos relacionados con el rescate, la vigilancia, los hogares, la orientación, los museos y las fábricas.

En estos contextos, los robots deben coexistir o cooperar con seres humanos u otros vehículos en movimiento en escenarios que cambian constantemente. Los planificadores tradicionales para escenarios estáticos y las técnicas de evasión puramente reactivas ya no son válidos en estos casos, o funcionan de manera degradada.

La planificación del movimiento en escenarios dinámicos requiere la capacidad de predecir la evolución futura de los obstáculos, a fin de planificar y ejecutar los movimientos más seguros y rápidos posibles para el robot. Además, es necesario tener en cuenta las restricciones cinemáticas y dinámicas para obtener trayectorias factibles.

Para esto se necesita una técnica de planificación y navegación centrada en el robot para este tipo de entornos, que considera la seguridad, la maniobrabilidad y las restricciones tanto del robot como del entorno.

El *Dynamic Object Velocity Space* (DOVS) [1] es un modelo para representar el dinamismo de un entorno, diseñado para robots no holonómicos, que mapea el entorno en el espacio de velocidad del robot y el cual se explica en más detalle en el Capítulo 2.

El modelo DOVS es de gran utilidad para ser luego utilizado por planificadores. Por ejemplo, D3QN-DOVS [2], utiliza la abstracción profunda del entorno proporcionada por el DOVS para comprender el escenario circundante del robot. La información detallada proporcionada por el DOVS se utiliza como entrada para el aprendizaje profundo por refuerzo (DRL), lo que permite al algoritmo seleccionar las mejores acciones para cada situación y mejorar la navegación en entornos dinámicos. Otro planificador de movimiento, S-DOVS (Strategy-based Dynamic Object Velocity Space) utiliza la información sobre la posición y velocidad de los objetos en el entorno para

crear el DOVS y planificar una trayectoria segura y eficiente para el robot en entornos dinámicos. Esto permite al robot navegar en su entorno evitando las colisiones con objetos en movimiento que pudiera encontrar a su alrededor adaptándose a los cambios en el entorno de manera efectiva. Este planificador tiene limitaciones cuando el mapa es muy grande o los obstáculos muy complejos. Para solucionar este problema existe una variante, Full-Stack S-DOVS [3], una versión modificada del planificador S-DOVS, que incluye un sistema de localización, un rastreador de obstáculos y un generador de puntos de ruta. Otra variante es RL-DOVS [4], el cual utiliza aprendizaje por refuerzo (RL), utilizando como entrada para el aprendizaje la abstracción del entorno proporcionada por el DOVS, para seleccionar las mejores acciones para cada situación.

El presente trabajo de fin de grado implementa en Python y evalúa exhaustivamente el modelo DOVS en diferentes escenarios dinámicos. En el Capítulo 3 se exponen las razones de la utilización de Python, siendo estas básicamente su claridad y su fácil extensibilidad y diseño modular.

Además, destacar que la decisión de re implementar el modelo DOVS desde cero fue necesaria. El código original había pasado por múltiples desarrolladores y dependía de numerosas librerías, lo que dificultaba su instalación y modificación. Por lo tanto, se determinó que la mejor opción era comenzar desde cero, permitiendo así una implementación más eficiente y mantenible del modelo.

Sobre este modelo se pueden desarrollar diferentes estrategias de planificación y de navegación de robots.

En un futuro, y gracias a la implementación clara y sencilla que nos permite el lenguaje y un buen diseño, se esperan abrir líneas de investigación desde esta librería, que permitan escalar el proyecto, por ejemplo, incorporando una tercera dimensión, añadiendo toma de decisiones compartidas en entornos multi-robot o incorporar diferentes métodos de navegación.

## 1.2. Objetivos y tareas

El objetivo del proyecto es la implementación en Python del algoritmo DOVS para la navegación en entornos dinámicos.

En concreto en este trabajo se busca implementar el modelado del entorno cambiante en el que se tiene que desenvolver el robot. Este modelo permite que el planificador del robot dote al mismo de las instrucciones necesarias para permitirle una navegación segura evitando las colisiones en un entorno cambiante.

En la sección 2.1 se explica con detalle en qué consiste el DOVS y la forma de calcularlo.

Una de las prioridades del trabajo es implementar un código que sea fácilmente entendible y extensible. Es por esto por lo que el diseño correcto de las clases es una parte fundamental del trabajo.

En la siguiente lista se describen las tareas concretas que se han realizado durante la totalidad del trabajo.

- **Estudio del DOVS:** Se ha estudiado el contenido del artículo que define el DOVS[1].
- **Diseño de la función principal:** Con el conocimiento adquirido sobre el DOVS. Se ha propuesto una función principal que implementa la generación del modelo.
- **Diseño las clases de clases:** Basándose en la información que necesitaba la función que genera el DOVS se ha planteado un diagrama de clases que permitiese resolver el problema.
- **Implementación:** Una vez que se refina el diseño se ha pasado a la implementación, la cual ha tenido las siguientes fases.
  - Función principal: Se ha implementado la función principal que genera el modelo.
  - Objetos del DOVS: Se ha procedido a implementar el robot y los obstáculos que forman el entorno que se pretende modelar.
  - Trayectorias de los objetos: Se ha implementado la trayectoria tanto del robot como de los obstáculos.
  - Cálculo de los puntos de colisión: Se ha implementado el cálculo de los puntos de colisión generado por la intersección de las trayectorias del robot con la de los obstáculos.
  - Cálculo de la velocidad: Se calculan las velocidades que producen una colisión en base a los puntos calculados.
  - Generación del DOVS: Con toda esa información se genera el DOVS y se muestra.
- **Pruebas:** Se han generado una gran cantidad de entornos que han permitido probar el correcto funcionamiento del código implementado. Corrigiendo aquellas partes del código en la que se encontraban errores
- **Memoria:** Se ha generado toda la documentación del proyecto.

Para más información en cuanto a la organización y duración de las tareas, se puede consultar el apéndice A, donde se habla de cómo se ha gestionado internamente la carga de las diferentes fases del trabajo, acompañado del diagrama de Gantt A.2. También se encuentra disponible el repositorio utilizado a nivel personal para gestionar las versiones del proyecto, donde se puede extraer información detallada de las tareas desarrolladas y ver el código realizado [5].

### **1.3. Contenido de la memoria**

A continuación, se resume brevemente el contenido de cada capítulo y de cada anexo de este documento.

En el capítulo 2 se describe el DOVS y los objetos que componen el entorno. Además, se explica cómo se calcula el modelo y las fórmulas matemáticas necesarias para ello.

En el capítulo 3 se expone todo lo relacionado con el diseño y la implementación, se muestran los diagramas de clases y paquetes y las decisiones tomadas a la hora de diseñar la funcionalidad que se iba a implementar. A continuación, se muestra mediante diagramas las partes principales de la implementación.

En el capítulo 4, se muestran las pruebas de validación realizadas, los resultados obtenidos, los principales problemas encontrados durante el proyecto, y las soluciones adoptadas.

Finalmente, en el capítulo 5 se muestran las conclusiones del proyecto, y las posibilidades de desarrollo y mejora.

En el apéndice A se incluye información sobre cómo se ha gestionado el proyecto, como el número de horas aproximadas y el diagrama de Gantt. En el apéndice B se encuentran las partes más importantes del código. En el apéndice C se encuentran los diagramas de la implementación.

# Capítulo 2

## DOVS

En este capítulo se describe el modelo DOVS utilizado, así como los objetos que lo componen, en concreto el robot, los obstáculos y las propiedades de cada uno de ellos.

También se aborda el procedimiento seguido para su modelización, explicando con detalle cada una de las ecuaciones utilizadas.

### 2.1. Definición

El Dynamic Object Velocity Space (DOVS) es un modelo utilizado en la navegación autónoma de robots para calcular comandos de movimiento seguros dentro de un horizonte temporal. El DOVS aborda el desafío de modelar el entorno dinámico al representar la dinámica del sistema robótico y la evolución del movimiento de los obstáculos en movimiento para ser utilizada en la planificación de trayectorias de los robots.

El DOVS se basa en el concepto del *Dynamic Object Velocity* (DOV). El DOV representa el conjunto de las velocidades lineal y angular del robot que, partiendo de un estado actual de un robot con tracción diferencial, provocarían una colisión con un obstáculo en movimiento dentro de un horizonte temporal.

Al considerar el DOV y combinarlo con las restricciones cinemático-dinámicas del robot, el DOVS es capaz de determinar las velocidades seguras y factibles para el robot en cada momento para el horizonte temporal considerado, teniendo en cuenta las posibles colisiones futuras con los obstáculos en movimiento.

Esto permite representar de manera explícita en el espacio de control del robot tanto los comandos libres de colisión como los que podrían causar colisiones. Al seleccionar comandos libres de colisión, cualquier planificador de movimiento basado en el modelo DOVS generará trayectorias libres de colisiones.

Esto garantiza que el robot pueda planificar y ejecutar movimientos que sean tanto seguros como eficientes, permitiéndole navegar de manera autónoma evitando colisiones

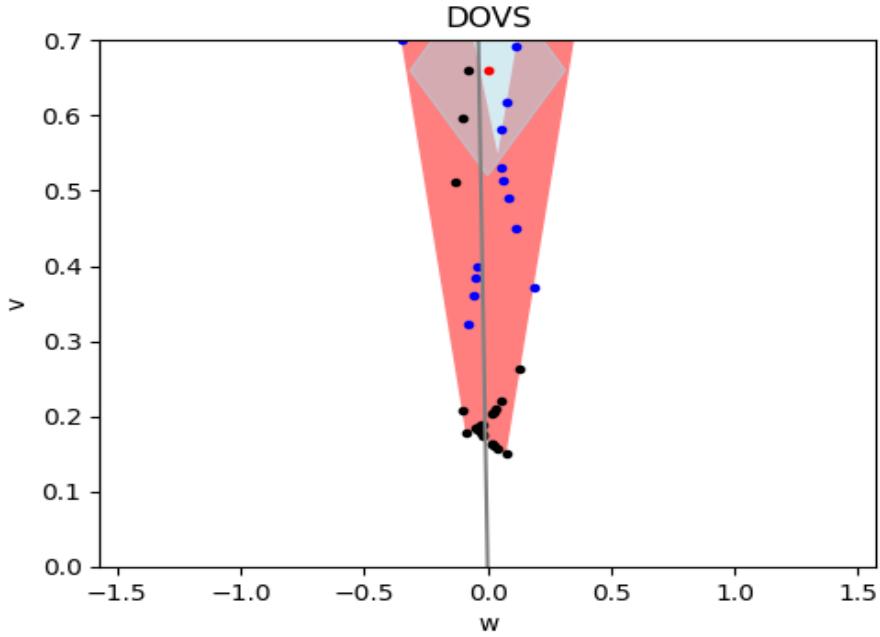


Figura 2.1: Ejemplo DOVS. Las zonas rojas representan las velocidades que llevan a colisión y las blancas las libres/seguras. El punto rojo representa la velocidad actual y el rombo azul todas las velocidades alcanzables en un periodo de muestreo.

en un entorno en constante cambio.

En resumen, el Espacio de Velocidad de Objetos Dinámicos (DOVS) es un modelo para representar y calcular comandos de movimiento seguros dentro de un horizonte temporal en entornos dinámicos. Abstacta el entorno dinámico al modelar las velocidades seguras e inseguras que el robot puede elegir en cada período de muestreo, representándolas directamente en el espacio de control del robot, en este caso el de velocidad lineal y angular de un robot con tracción diferencial.

Al considerar la dinámica del entorno y las restricciones del robot, el DOVS permite una navegación autónoma eficiente y segura, evitando colisiones en un entorno en constante cambio.

## 2.2. El robot

En esta sección se define el robot 2.2.1 sobre el que se genera el modelo DOVS y se explican las partes que lo componen, es decir, sus trayectorias (2.2.2) y su ventana de velocidad (2.2.3).

### 2.2.1. Definición

El objetivo final es la navegación autónoma del robot por un entorno dinámico, es por tanto imprescindible describir el robot y como se representa en el modelo a implementar.

Consideramos un robot con tracción diferencial  $R$  moviéndose en un entorno dinámico, donde debe llegar de manera segura a su objetivo evitando colisiones con los objetos en movimiento a su alrededor.

El estado del robot se define por su localización (posición y orientación) y su velocidad en el instante  $t$ ,  $R_t = (x, y, \theta, \omega, v)$ . En nuestro caso, los controles son la velocidad angular ( $\omega$ ) y la velocidad lineal ( $v$ ) para un robot de tracción diferencial. El modelo de movimiento para el robot puede expresarse mediante las conocidas ecuaciones (2.1):

$$\dot{x} = v \cdot \cos(\theta); \quad \dot{y} = v \cdot \sin(\theta); \quad \dot{\theta} = \omega \quad (2.1)$$

En resumen, un robot es considerado como un sistema no holonómico que se mueve en un entorno dinámico y tiene la tarea de llegar de manera segura a un objetivo definido por su localización, posición y orientación  $(x_o, y_o, \theta_o)$ , evitando colisiones con objetos en movimiento a su alrededor.

### 2.2.2. Trayectorias

Las trayectorias se generan mediante un conjunto discretizado de trayectorias circulares factibles para robot, asumiendo que este mantiene una velocidad lineal y angular constante en el horizonte temporal considerado. Esta asunción no es problemática debido a que, en cada instante temporal, el modelo entero se recalcula teniendo en cuenta los cambios.

Los centros de las circunferencias que forman las trayectorias, se calculan mediante las siguientes ecuaciones (2.2, 2.3):

$$\forall r \in [-max_r, max_r], \quad center_x = x + r \cdot \cos\left(\theta + \frac{\pi}{2}\right) \quad (2.2)$$

$$center_y = y + r \cdot \sin\left(\theta + \frac{\pi}{2}\right) \quad (2.3)$$

donde  $center_x$  y  $center_y$  son las coordenadas del centro de las circunferencias en el sistema de referencia del robot,  $r$  es el radio,  $max_r$  es el radio máximo de las circunferencias,  $x$  e  $y$  es la ubicación del robot en el sistema de referencia del robot, y  $\theta$  la orientación del robot.

Una vez que se obtienen los centros, se generan los círculos que forman las trayectorias con la siguiente ecuación (2.4):

$$\forall r \in [-max_r, max_r], \quad center_x^2 + center_y^2 = r^2 \quad (2.4)$$

Donde:

- $center_x$  y  $center_y$  son las coordenadas del centro de las circunferencias, en el sistema de referencia del robot. Las cuales se calcula mediante las ecuaciones 2.2 y 2.3.
- $r$  es el radio,  $max_r$  es el radio máximo de las circunferencias.

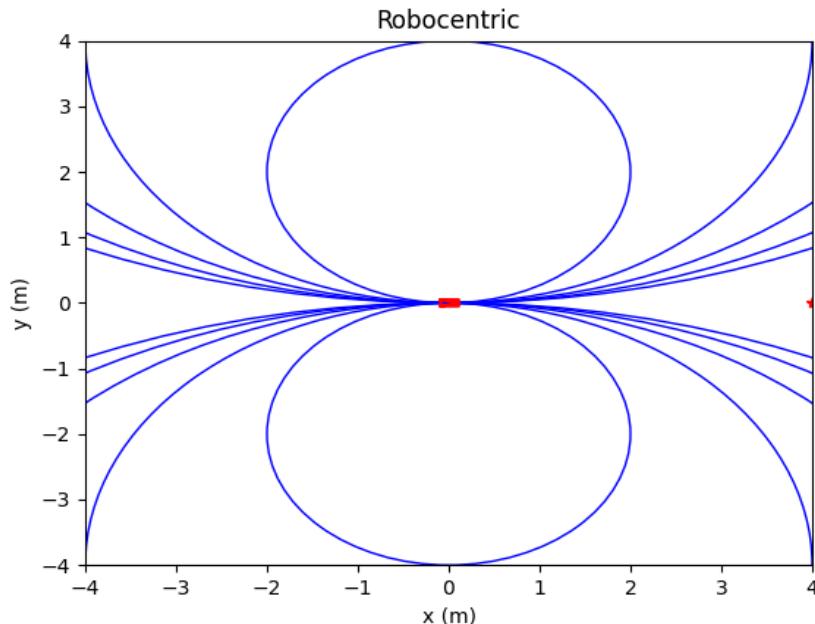


Figura 2.2: Ejemplo posibles trayectorias del robot en coordenadas robocéntricas

Estas trayectorias, junto con la de los obstáculos son las que nos permiten obtener los puntos de colisión mediante los cuales se crea el modelo.

En la figura 2.2, puede observarse un ejemplo de posibles trayectorias factibles del robot, que pueden utilizarse para el cálculo del DOVS.

### 2.2.3. Ventana de velocidad

Debido a las restricciones cinemático-dinámicas existe una aceleración máxima del robot, además de otras restricciones. Esto hace que no todas las velocidades posibles del robot sean alcanzables en un periodo de muestreo. Esto se refleja en el modelo DOVS con un rombo de la manera reflejada en la figura 2.3. El círculo rojo central es

la velocidad ( $\omega, v$ ) actual y el contorno del rombo representa las velocidades máximas y mínimas que el robot puede alcanzar para el siguiente instante. Las fórmulas que calculan los extremos del rombo ( $v_{max}, v_{min}, w_{max}, w_{min}$ ) son las siguientes (2.5, 2.6, 2.7, 2.8 para cada extremo respectivamente):

$$v_{max} = max\_av_{robot} \cdot timestep + v_{robot} \quad (2.5)$$

$$v_{min} = -max\_av_{robot} \cdot timestep + v_{robot} \quad (2.6)$$

$$w_{max} = max\_aw_{robot} \cdot timestep + w_{robot} \quad (2.7)$$

$$w_{min} = -max\_aw_{robot} \cdot timestep + w_{robot} \quad (2.8)$$

Siendo  $v_{max}, v_{min}$  la velocidad lineal máxima y mínima que el robot puede alcanzar para el siguiente instante,  $w_{max}, w_{min}$  la velocidad angular máxima y mínima que el robot puede alcanzar para el siguiente instante,  $max\_av_{robot}, max\_aw_{robot}$  la aceleración lineal y angular máxima del robot,  $timestep$  el periodo de control, y  $v_{robot}, w_{robot}$  la velocidad lineal y angular actual del robot.

Las velocidades disponibles dentro de la ventana de velocidades son las que se utilizarán por los planificadores para escoger las velocidades adecuadas en cada momento desde la velocidad actual.

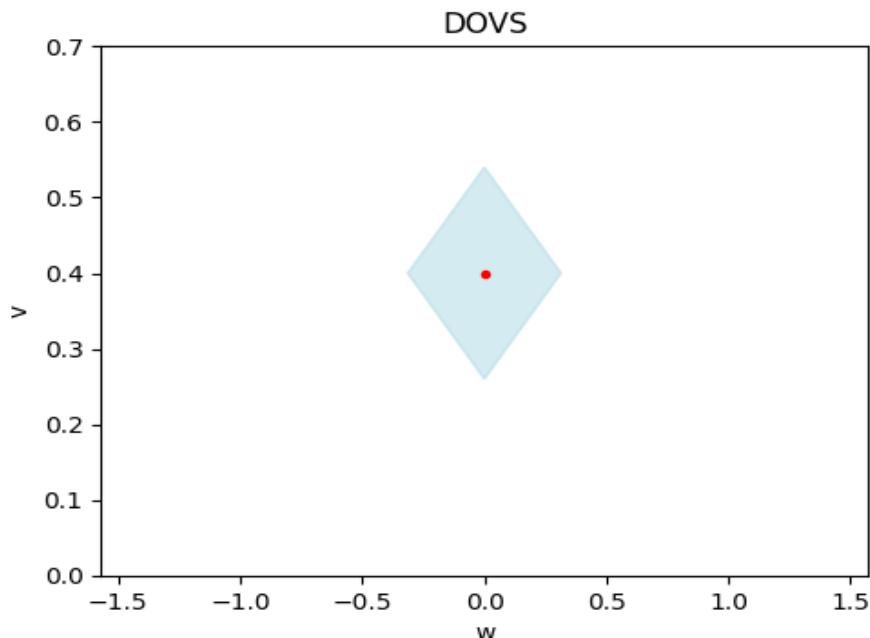


Figura 2.3: Ejemplo ventana de velocidad del robot

## 2.3. Obstáculos

Los obstáculos en el modelo al igual que el robot se definen por su ubicación y velocidad en el instante  $t$ ,  $O_t = (x, y, \theta, \omega, v)$ .

Dado que en su aplicación real el robot observará los obstáculos circundantes mediante los sensores embarcados en él, las variables correspondientes a los obstáculos se representarán en el sistema de referencia robocéntrico  $R$ . Es decir, se quiere situar el objeto ( $O$ ) respecto a la referencia del robot ( $R$ ), siendo ( $W$ ) la referencia absoluta o del mundo (*World*). Para ello es necesario calcular la transformación entre las referencia  $O$  y  $R$ , utilizando matrices homogéneas. Esto se consigue mediante las ecuaciones (2.9, 2.10):

$${}^R T_O = {}^W T_R^{-1} \cdot {}^W T_O \quad (2.9)$$

$${}^R T_O = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & d_x \\ \sin(\theta) & \cos(\theta) & d_y \\ 0 & 0 & 1 \end{bmatrix} \rightarrow {}^R x_O = (d_x, d_y, \theta) \quad (2.10)$$

donde  ${}^R x_O$  es la localización (posición y orientación) del obstáculo en la referencia del robot,  ${}^R T_O$  es la matriz homogénea de traslación y rotación del obstáculo respecto del robot,  ${}^W T_R^{-1}$  es la matriz homogénea de traslación y rotación inversa del robot respecto la referencia absoluta y  ${}^W T_O$  es la matriz homogénea de traslación y rotación del obstáculo respecto la referencia absoluta.

En el modelo, se utiliza el Espacio de Configuraciones (*CS, Configuration Space*), en el que el robot se reduce a un punto, y la dimensión de los obstáculos se amplía con el radio del robot. Ello facilita el cálculo de trayectorias del robot sin colisión con los obstáculos, al ser necesario sólo calcular la trayectoria para un punto representativo del robot, usualmente su centro.

En este trabajo se van a considerar únicamente obstáculos móviles. A la hora de calcular las velocidades que llevan a colisión con los obstáculos, es necesario marcar los puntos del obstáculo que definen la entrada y salida en la trayectoria del obstáculo (figura 2.5). Es por esto, que se modelan como cuadrados envolventes y se distinguen dos puntos, el que a partir del cual se considera que el obstáculo ha pasado antes de llegar el robot a él, siempre en la parte trasera del obstáculo, y el que punto a partir del cual se considera que el robot pasa por delante del obstáculo, que se representa en la parte superior del obstáculo. Un ejemplo de estos puntos puede observarse en la figura 2.4.

La posición de estos puntos a la izquierda o derecha del obstáculo dependen de la orientación del mismo. Como se puede verse en las ecuaciones 2.11, 2.12:

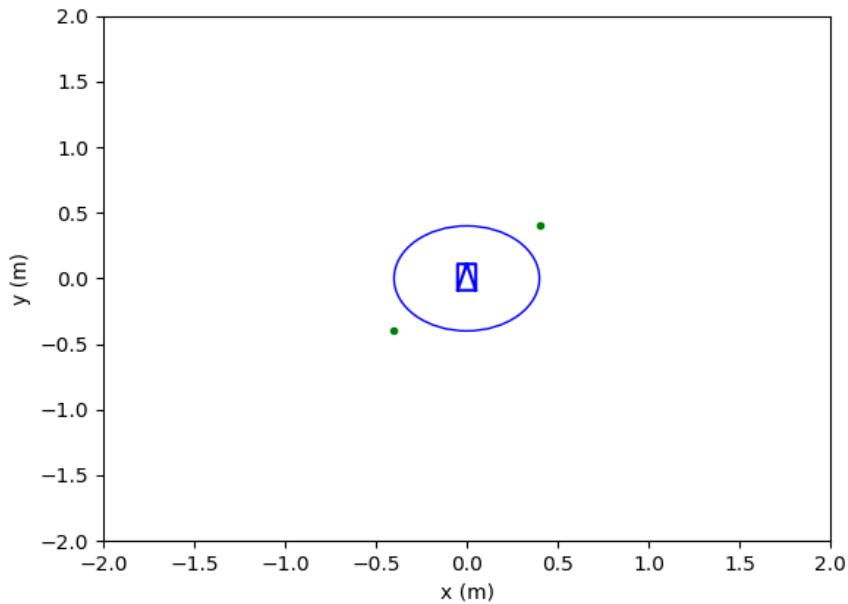


Figura 2.4: Ejemplo puntos colisión con obstáculo

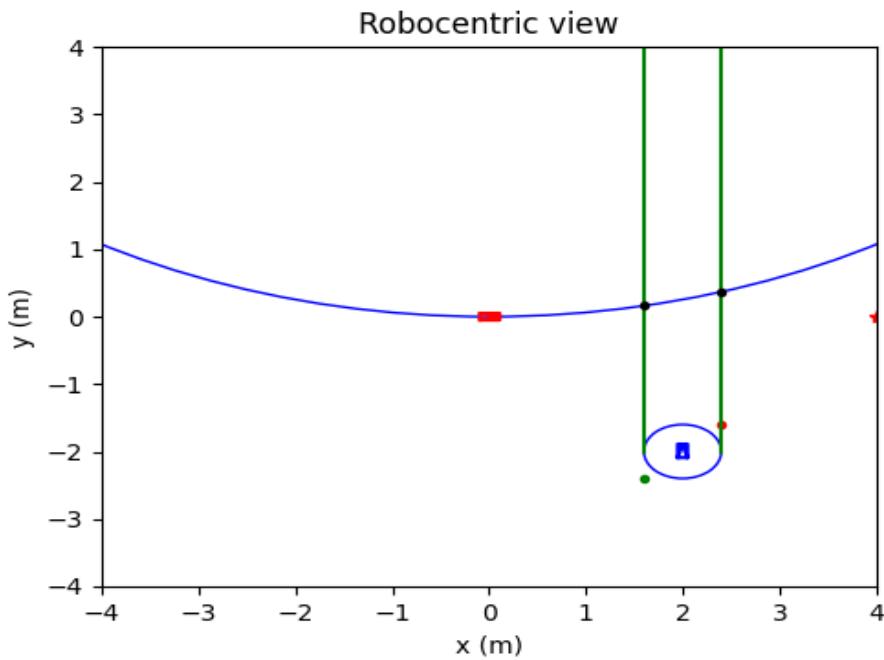


Figura 2.5: Robot con una sola trayectoria, banda de colisión del obstáculo, con los puntos representativos extremos de los obstáculos, para los que se calculan los puntos de colisión. El punto rojo del obstáculo representa el punto  ${}^o x_{CPA}$ , calculado en la ecuación 2.11, el cual marca la posición en el obstáculo a partir de la cual se considera que el robot pasa por delante del obstáculo. El punto verde del obstáculo representa el punto  ${}^o x_{CPD}$ , calculado en la ecuación 2.12, el cual marca la posición en el obstáculo a partir del cual se considera que el obstáculo ha pasado antes de llegar el robot a él.

$${}^O x_{CPA} = \begin{cases} (radius, radius, 0) & \text{si } \theta < 0 \\ (radius, -radius, 0) & \text{si } \theta \geq 0 \end{cases}, \quad (2.11)$$

donde  $radius$  es el radio del obstáculo, aumentado con el radio del robot,  $\theta$  es la orientación del obstáculo en la referencia del robot y  ${}^O x_{CPA}$  es la posición del punto calculado en la referencia del obstáculo  $O$ , el cual marca la posición en el obstáculo a partir de la cual se considera que el robot pasa por delante del obstáculo. Con los puntos considerados para pasar después del obstáculo la ecuación considerada es la siguiente:

$${}^O x_{CPD} = \begin{cases} (-radius, -radius, 0) & \text{si } \theta < 0 \\ (-radius, radius, 0) & \text{si } \theta \geq 0 \end{cases} \quad (2.12)$$

donde  $radius$  es el radio del obstáculo, aumentado con el radio del robot,  $\theta$  es la orientación del obstáculo desde la referencia del robot y  ${}^O x_{CPD}$  es la posición del punto calculado en la referencia del obstáculo  $O$ , el cual indica la posición en el obstáculo a partir del cual se considera que el obstáculo ha pasado antes de llegar el robot a él.

Esta información hay que representarla en la referencia del robot  $R$ , ya que toda la información para la toma de decisiones de navegación se va a representar en esta referencia. Para ello se aplican las siguientes transformaciones, ecuaciones 2.13, 2.14, 2.15, 2.16:

$${}^R T_{CPA} = {}^R T_O \cdot {}^O T_{CPA} \quad (2.13)$$

$${}^R T_{CPD} = {}^R T_O \cdot {}^O T_{CPD} \quad (2.14)$$

$${}^R T_{CPA} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & d_x \\ \sin(\theta) & \cos(\theta) & d_y \\ 0 & 0 & 1 \end{bmatrix} \rightarrow {}^R x_{CPA} = (d_x, d_y, \theta) \quad (2.15)$$

$${}^R T_{CPD} = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & d_x \\ \sin(\theta) & \cos(\theta) & d_y \\ 0 & 0 & 1 \end{bmatrix} \rightarrow {}^R x_{CPD} = (d_x, d_y, \theta) \quad (2.16)$$

Donde:

- ${}^R T_{CPA}$  es la matriz homogénea de traslación y rotación del punto del obstáculo 2.11, respecto del robot,  ${}^R T_O$  es la matriz homogénea de traslación y rotación del obstáculo respecto del robot y  ${}^O T_{CPA}$  es la matriz homogénea de traslación y rotación del punto del obstáculo 2.11 respecto del robot.
- ${}^R T_{CPD}$  es la matriz homogénea de traslación y rotación del punto del obstáculo 2.12, respecto del robot,  ${}^R T_O$  es la matriz homogénea de traslación y rotación del obstáculo respecto del robot y  ${}^O T_{CPD}$  es la matriz homogénea de traslación y rotación del punto del obstáculo 2.12, respecto del robot.

- ${}^R x_{CPA}$  es la posición del punto del obstáculo 2.11 desde la referencia del robot.
- ${}^R x_{CPD}$  es la posición del punto del obstáculo 2.12 desde la referencia del robot.

### 2.3.1. Trayectorias de los obstáculos

En este trabajo se han considerado dos tipos de trayectorias, rectilíneas y curvilíneas. Sería extensible a otras trayectorias más complejas con la misma metodología, pero por simplicidad se han considerado aquí estas trayectorias.

Primero, se calculan los dos puntos en los laterales del obstáculo en la referencia del robot  $R$ , a partir de los cuales se calcula la línea (recta o circular) que define la trayectoria. Esto se consigue mediante las siguientes fórmulas (2.17, 2.18, 2.19, 2.20):

$$x_1 = x + radius \cdot \cos(\theta + \frac{\pi}{2}) \quad (2.17)$$

$$y_1 = y + radius \cdot \sin(\theta + \frac{\pi}{2}) \quad (2.18)$$

$$x_2 = x + radius \cdot \cos(\theta - \frac{\pi}{2}) \quad (2.19)$$

$$y_2 = y + radius \cdot \sin(\theta - \frac{\pi}{2}) \quad (2.20)$$

Donde:

- $x, y$  y  $\theta$  son la ubicación y orientación del obstáculo en la referencia del robot.
- $x_1, y_1$  es la posición en la referencia del robot, de donde parte una de las líneas que forman parte de la trayectoria del obstáculo.
- $x_2, y_2$  es la posición en la referencia del robot, de donde parte la otra de las líneas que forman parte de la trayectoria del obstáculo.

#### Rectilínea

A partir de esos puntos se calculan dos rectas con la orientación del obstáculo en la referencia del robot. Las fórmulas mediante las cuales se calculan las rectas son las siguientes (2.21, 2.22):

$$y_1 = \tan(\theta) \cdot x_1 \quad (2.21)$$

$$y_2 = \tan(\theta) \cdot x_2 \quad (2.22)$$

Donde:

- $x_1, y_1$  es el punto en la referencia del robot, a un lado del obstáculo, el cual se calcula con las ecuaciones 2.17 y 2.18, que marca el inicio de una de las líneas que forma parte de la trayectoria del obstáculo.

- $x_2, y_2$  es el punto en la referencia del robot, al otro lado del obstáculo, el cual se calcula con las ecuaciones 2.19 y 2.20, que marca el inicio de la otra línea que forma parte de la trayectoria del obstáculo.
- $\theta$  es la orientación del obstáculo desde la referencia del robot.

Un ejemplo de la trayectoria rectilínea de un obstáculo puede observarse en la figura 2.6.

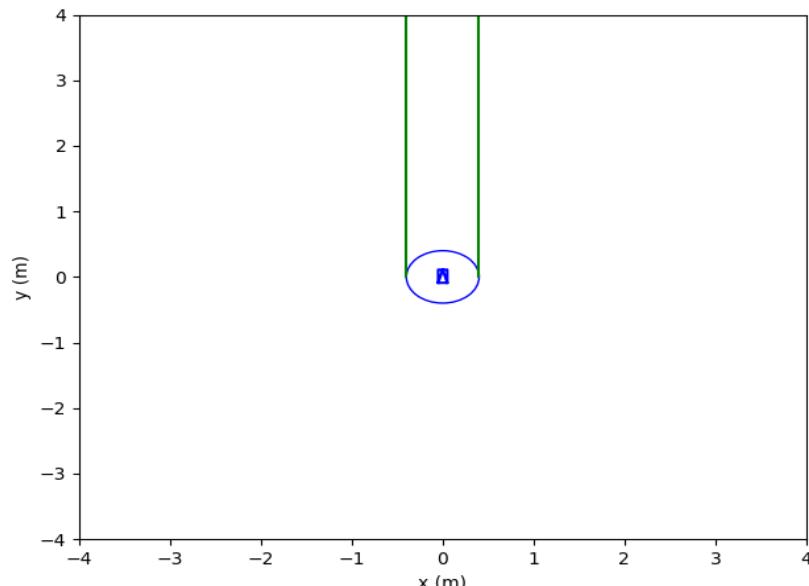


Figura 2.6: Ejemplo de trayectoria rectilínea de un obstáculo. Las rectas verdes delimitan la banda del plano barrida por el obstáculo, en la dirección de movimiento del mismo.

## Curvilíneas

Un ejemplo de la trayectoria curvilínea de un obstáculo puede observarse en la figura 2.7.

Para generar la trayectoria hace falta calcular el centro, en la referencia del robot, y los radios de las circunferencias que la forman. Esto se logra mediante las siguientes ecuaciones (2.23, 2.24, 2.25, 2.26, 2.27):

$$radius = \frac{v}{\omega} \quad (2.23)$$

$$center_x = x + radius \cdot \cos \left( \theta + \frac{\pi}{2} \right) \quad (2.24)$$

$$center_y = y + radius \cdot \sin \left( \theta + \frac{\pi}{2} \right) \quad (2.25)$$

$$radius_1 = \sqrt{(center_x - x_1)^2 + (center_y - y_1)^2} \quad (2.26)$$

$$radius_2 = \sqrt{(center_x - x_2)^2 + (center_y - y_2)^2} \quad (2.27)$$

Donde:

- $v, \omega$  es la velocidad lineal y angular del obstáculo.
- $x, y, \theta$  es la ubicación del obstáculo desde la referencia del robot.
- $x_1, y_1$  es el punto, en la referencia del robot, a un lado del obstáculo, el cual se calcula con las ecuaciones 2.17 y 2.18, que marca el inicio de una de las líneas que forma parte de la trayectoria del obstáculo.
- $x_2, y_2$  es el punto, en la referencia del robot, al otro lado del obstáculo, el cual se calcula con las ecuaciones 2.19 y 2.20, que marca el inicio de la otra linea que forma parte de la trayectoria del obstáculo.
- $radius$  es el radio de la trayectoria con velocidad lineal  $v$  y velocidad angular  $\omega$ .
- $center_x, center_y$  es la posición, en la referencia del robot, del centro de la trayectoria curvilínea.
- $radius_1$  y  $radius_2$  son los dos radios de las líneas que forman la trayectoria.

Con los datos obtenidos, se generan los círculos que forman la trayectoria con las siguientes ecuaciones 2.28, 2.29:

$$center_x^2 + center_y^2 = radius_1^2 \quad (2.28)$$

$$center_x^2 + center_y^2 = radius_2^2 \quad (2.29)$$

Donde:

- $center_x, center_x$  es la posición, en la referencia del robot, del centro de la trayectoria curvilínea. La cual se calcula mediante las ecuaciones 2.24, 2.25.
- $radius_1$  y  $radius_2$  son los dos radios de las líneas que forman la trayectoria. Estos radios se calculan con las ecuaciones 2.26, 2.27.

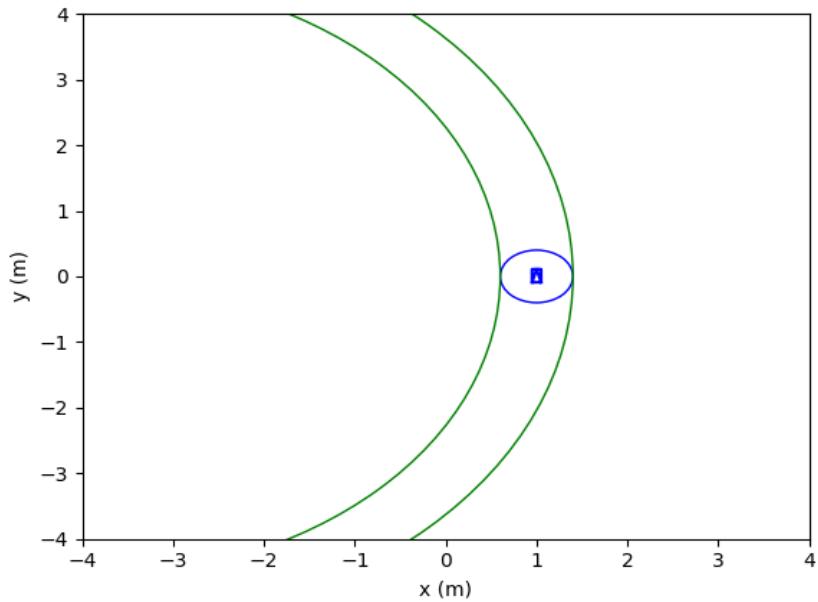


Figura 2.7: Ejemplo de trayectoria curvilínea de un obstáculo

## 2.4. Cálculo del DOVS

El DOVS se construye considerando la trayectoria de los obstáculos y calculando las velocidades máximas y mínimas del robot que evitan la colisión con los obstáculos. Es decir, aquellas velocidades mínimas necesarias para que el robot pase antes que llegue el obstáculo al punto de colisión, y las velocidades máximas que puede llevar el robot para que el obstáculo pase antes de que llegue el robot al punto de colisión. Con esta información, se determinan las velocidades seguras para el robot con las que planificar trayectorias.

El modelo se construye mapeando la información del entorno, sobre el espacio de control del robot, denominado espacio de velocidad ( $V$ ), que es el conjunto de velocidades ( $\omega, v$ ) alcanzables por el robot y limitado por las restricciones máximas y mínimas de velocidad.

El concepto básico para construir el modelo es el tiempo de colisión entre las trayectorias del robot y los obstáculos. Los puntos de intersección se calculan a partir de una representación robocéntrica, esto determina una información asociada en el espacio de velocidad sobre el tiempo y la velocidad, con los instantes en los que un obstáculo alcanza el punto de intersección, y las velocidades para que el robot pase antes de la colisión o después de que el objeto pase, evitando de esta forma la colisión.

El DOV contiene la información calculada de las velocidades de los obstáculos.

Estos datos de tiempo y velocidad se transfieren al espacio de control del robot para

representar los obstáculos en el entorno, conformando el espacio de velocidad-tiempo (DOVTS) del robot. En este trabajo por simplificar no se tiene en cuenta el tiempo.

El mapeo de las velocidades de los obstáculos (DOV) al espacio de velocidades del robot ( $V$ ) es lo que determina el DOVS. El DOVS representa las velocidades seguras e inseguras en el espacio de control del robot, permitiendo la planificación de movimientos libres de colisiones. Este espacio de velocidades permitirá a un planificador de movimientos seleccionar velocidades seguras que generan trayectorias sin colisión (figura 2.8).

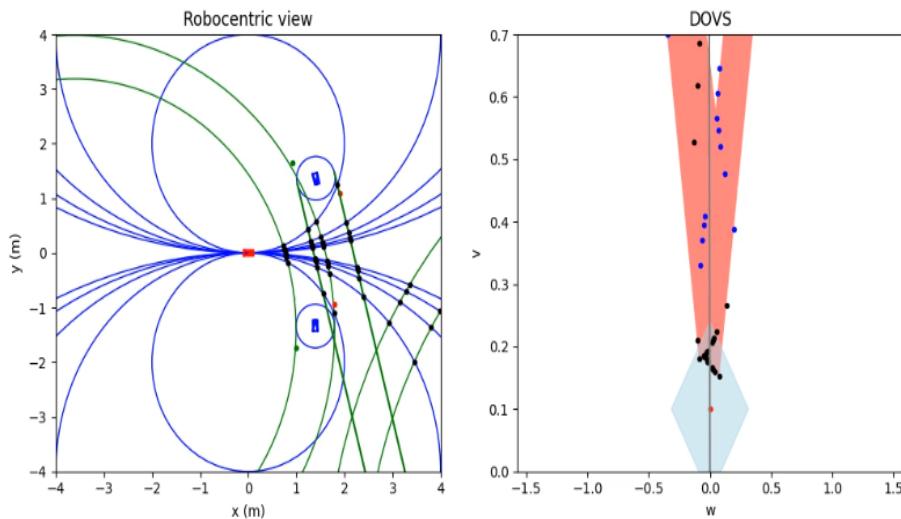


Figura 2.8: Ejemplo de DOVS. Las zonas rojas son las velocidades que llevan a colisión con los obstáculos. El resto de las velocidades son seguras. Los puntos azules corresponden a las velocidades mínimas necesarias para que el robot pase antes que llegue el obstáculo al punto de colisión, y las velocidades máximas que puede llevar el robot para que el obstáculo pase antes de que llegue el robot al punto de colisión.

A continuación, se explica en detalle las dos partes principales del cálculo del modelo:

- **Cálculo de puntos de colisión** (Sec. 2.4.1): Utilizando la representación robocéntrica, se calculan los puntos de colisión entre las trayectorias del robot y los obstáculos. Estos puntos de colisión representan los instantes en los que una colisión ocurriría si el robot siguiera una determinada velocidad y trayectoria. El cálculo de estos puntos implica considerar la dinámica del robot y la evolución de las trayectorias de los obstáculos en movimiento.
- **Determinación de velocidades que llevan a colisión** (Sec. 2.4.2): Una vez que se han calculado los puntos de colisión, se procede a determinar en base a esos puntos las velocidades y tiempos que llevan a colisión.

## 2.4.1. Cálculo de los puntos de colisión

Las coordenadas de los puntos de colisión se han calculado con la intersección geométrica de la trayectoria del obstáculo (ecuaciones 2.21 y 2.22, si el obstáculo sigue una trayectoria rectilínea, y ecuaciones 2.28 y 2.29 si el obstáculo sigue una trayectoria circular) y la trayectoria del robot (ecuación 2.4).

Para cada pareja trayectoria del robot y trayectoria del obstáculo se obtienen dos puntos de colisión, el que marca la colisión que permitirá que el robot pasará antes del obstáculo y la que marcaría que el robot pasa por detrás del obstáculo, es decir uno de los puntos sería la intersección con la línea de la banda de colisión cercana y el otro con la lejana. En las imágenes 2.9, 2.10, 2.12 y 2.13 pueden verse ejemplos de los puntos de colisión obtenidos entre las trayectorias del robot y las de un obstáculo.

Para obtener los puntos de colisión correctos hay que filtrar los puntos de colisión obtenidos mediante la intersección ya que se puede obtener hasta dos puntos entre la intersección de un círculo con una recta o círculo. Esta situación puede observarse en la figura 2.9.

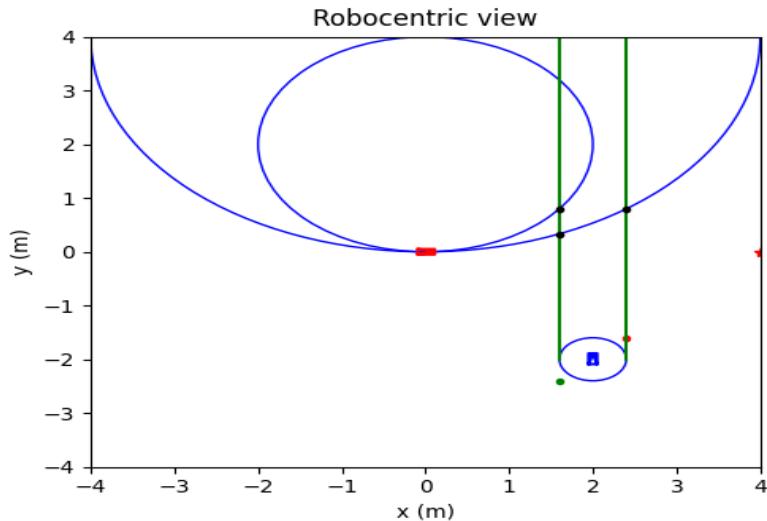


Figura 2.9: Ejemplo de colisiones entre las trayectorias de un robot y la de un obstáculo, se ha limitado el número de trayectorias del robot por ser más claro en qué puntos de colisión se consideran. La linea izquierda de la trayectoria del obstáculo corta dos veces a la trayectoria del robot de menor radio, de entre los dos puntos se selecciona aquel que este más cerca del robot, ya que implica la situación más cercana en el tiempo de esa trayectoria del robot.

Para cada punto de colisión definido por la posición  $(x, y)$ , en la referencia del robot, se calcula lo siguiente:

- Cálculo del ángulo barrido por el robot hasta la posición  $(x, y)$ , en la referencia

del robot:

$$\text{angle} = \arctan\left(\frac{2xy}{x^2 - y^2}\right) \quad (2.30)$$

donde:

- $x, y$  es la posición del punto de colisión en la referencia del robot.
- $\text{angle}$  es el ángulo barrido por el robot hasta la posición  $(x,y)$ , en la referencia del robot.
- Cálculo de la longitud de arco:

$$\text{arclength} = \text{trajectory\_radius} \cdot \text{angle} \quad (2.31)$$

donde:

- $\text{angle}$  es el ángulo barrido por el robot, calculado con la ecuación 2.30.
- $\text{trajectory\_radius}$  es el radio de la trayectoria del robot.
- $\text{arclength}$  es la longitud de arco, medida sobre la trayectoria del robot, desde la posición del robot hasta el punto  $(x,y)$ .
- Cálculo del ángulo en grados del arco:

$$\text{angle}_{\text{aux}} = \arctan\left(\frac{2x|y|}{x^2 - |y|^2}\right) \quad (2.32)$$

$$\text{angle}_{\text{aux}} = (\text{angle}_{\text{aux}} + 2\pi) \pmod{2\pi} \quad (2.33)$$

$$\text{arclength}_{\text{aux}} = \text{trajectory\_radius} \cdot \text{angle}_{\text{aux}} \quad (2.34)$$

$$\text{angulo\_arc} = \frac{\text{arclength}}{\text{trajectory\_radius}} \cdot \frac{180}{\pi} \quad (2.35)$$

donde:

- $x, y$  es la posición del punto de colisión en la referencia del robot.
- $\text{angle}_{\text{aux}}$  es el ángulo barrido por el robot hasta la posición  $(x,|y|)$ , en la referencia del robot.
- $\text{arclength}_{\text{aux}}$  es la longitud de arco, medida sobre la trayectoria del robot, desde la posición del robot hasta el punto  $(x,|y|)$ .
- $\text{angulo\_arc}$  es el ángulo en grados del arco.
- Calculo de la posición del punto de colisión en la referencia del obstáculo, es decir, situar el punto de colisión (C) respecto a la referencia del obstáculo (O), siendo

(R) la referencia del robot. Para ello es necesario calcular la transformación entre las referencias  $C$  y  $R$ , esto se consigue mediante las siguientes ecuaciones:

$${}^R x_C = (x, y, 0) \rightarrow {}^R T_C \quad (2.36)$$

$${}^O T_C = {}^R T_O^{-1} \cdot {}^R T_C \quad (2.37)$$

$${}^O T_C = \begin{bmatrix} \cos(\theta) & -\sin(\theta) & d_x \\ \sin(\theta) & \cos(\theta) & d_y \\ 0 & 0 & 1 \end{bmatrix} \rightarrow {}^O x_C = (d_x, d_y, \theta) \quad (2.38)$$

donde:

- $x, y$  y  ${}^R x_C$  es la posición del punto de colisión en la referencia del robot.
- ${}^O T_C$  es la matriz homogénea de traslación y rotación del punto de colisión respecto del obstáculo,  ${}^R T_O^{-1}$  es la matriz homogénea de traslación y rotación inversa del obstáculo respecto del robot y  ${}^R T_C$  es la matriz homogénea de traslación y rotación del punto de colisión respecto del robot.
- ${}^O x_C$  es la posición del punto de colisión desde la referencia del obstáculo.

A la hora de seleccionar los puntos de colisión válidos se siguen los siguientes principios:

- Solo se consideran aquellos puntos de colisión que vistos desde la referencia del obstáculo tienen un valor positivo de  $x$ . Es decir, dado un punto de colisión definido por la posición  $(x, y)$ , en la referencia del robot, solo se considera valido si

$${}^O x_C = (x_{aux}, y_{aux}, 0) \quad (2.39)$$

$$x_{aux} >= 0 \quad (2.40)$$

donde:  ${}^O x_C$  es el punto de colisión desde la referencia del obstáculo, calculado con la ecuación 2.38 y  $x_{aux}, y_{aux}$  es la posición del punto de colisión desde la referencia del obstáculo.

- En el caso de haber dos puntos de colisión entre la trayectoria del robot y una de las líneas que componen la trayectoria del obstáculo, esto se debe a que si se calcula la intersección entre una recta o circulo (una de las lineas que forma la trayectoria del obstáculo) con un círculo, hay dos posibles puntos de intersección.

Esta situación se puede observar en la figura 2.9. En este caso se selecciona aquel punto cuyo arco tenga menor ángulo, es decir, está más cerca del robot. Se selecciona el punto más cercano al robot, porque el robot llegara en menos tiempo a ese punto.

Es decir, dado dos puntos de colisión, definidos por las posiciones  $(x_1, y_1)$  y  $(x_2, y_2)$ , en la referencia del robot, sean, *angulo\_arc1* el ángulo en grados, calculado con la ecuación 2.35 del punto definido por  $(x_1, y_1)$  y *angulo\_arc2* el ángulo en grados, calculado con la ecuación 2.35 del punto definido por  $(x_2, y_2)$ , suponiendo que *angulo\_arc1* es menor que *angulo\_arc2*, el punto de colisión considerado, es el definido por las coordenadas  $(x_1, y_1)$ .

- El punto de colisión tiene que tener un ángulo en grados del arco entre  $\pm 90^\circ$ , lo que se calcula con la ecuación 2.35, el resto de grados no se tienen en cuenta ya que implican que el robot entra y sale de la banda de colisión u ocurren en una parte demasiado lejana de la trayectoria del robot. Esta situación puede observarse en la figura 2.10.

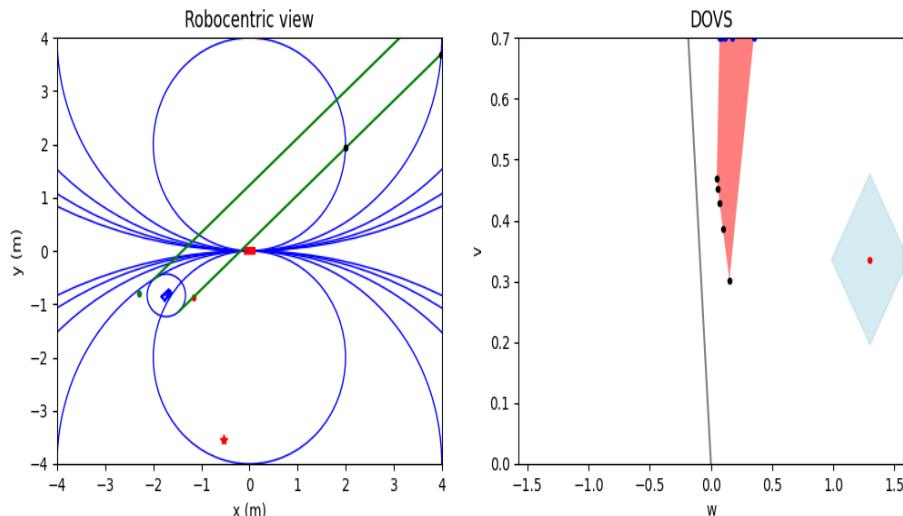


Figura 2.10: Ejemplo de situación con los puntos de colisión que se consideran. Puede observarse como los puntos de colisión cuyos ángulos en grados del arco sean mayor a  $90^\circ$  no son considerados.

#### 2.4.2. Cálculo de las velocidades

Una vez que se han seleccionado los puntos de colisión correctos, el objetivo es calcular el instante de tiempo en los que se produce esa colisión. Estos tiempos son los tiempos estimados de llegada del objeto al punto de colisión, que indican los

momentos de colisión. Por lo tanto, se determinan las velocidades mínimas y máximas que permiten al robot pasar antes o después del obstáculo.

El cálculo de estas velocidades se realiza para los puntos de colisión de cada trayectoria del robot con la trayectoria del obstáculo, en la banda de colisión, por lo que como máximo hay dos puntos de colisión.

La banda de colisión de un obstáculo está delimitada por las dos líneas que forman su trayectoria. Cada una de estas líneas producen un tiempo de colisión distintos.

La intersección de la trayectoria del robot con la línea más cercana define el momento en el que el obstáculo empieza a interseccionar con la trayectoria del robot y el de la más lejana el momento en el que deja de estarlo.

Si el robot está en la banda de colisión entre esos instantes se produciría una colisión. Por lo tanto, tiene dos opciones, pasar antes de que pase el obstáculo o después.

Con el tiempo obtenido con la intersección de la línea lejana con la trayectoria del robot, se obtiene la velocidad mínima que debe tener el robot para pasar antes de que el obstáculo pase.

Con el tiempo obtenido con la intersección de la línea más cercana con la trayectoria del robot, se obtiene la velocidad máxima que puede llevar el robot para no colisionar con el obstáculo.

Las velocidades de en medio son las que llevan al robot a colisión.

El objetivo es calcular los siguientes valores:

-  $(v_{max}, w_{max})$ , velocidad lineal y angular máxima que debe llevar el robot para no colisionar con el obstáculo.

-  $(v_{min}, w_{min})$ , velocidad lineal y angular mínima que debe llevar el robot para no colisionar con el obstáculo y pasar antes que él.

Donde:

- $robot\_v_{max}$  es la velocidad lineal máxima del robot.
- $radio\_trayectoria$  es el radio de la trayectoria del robot.
- $angulo$  es el ángulo barrido por el robot hasta el punto de colisión respecto del robot, calculado mediante la ecuación 2.30.
- $arc length$  es la longitud de arco del punto de colisión, calculado mediante la ecuación 2.31.

Es importante aclarar que  $v_{max}$  es siempre menor que  $v_{min}$ , ya que representan las velocidades que evita al robot entrar en la banda de colisión, y la que permite escapar de la banda de colisión antes de la llegada del obstáculo, respectivamente. Si no hay

ningún punto de colisión, sólo se calculan velocidades de escape en el caso de que el robot se encuentre dentro de la banda de colisión del obstáculo. Esta situación se refleja en la figura 2.11

En este caso, se considera que el obstáculo está muy cerca del robot, por lo tanto, no se han detectado colisiones entre las trayectorias, y se asume que esa trayectoria del robot lleva a colisionar con el obstáculo. Por tanto, las velocidades se calculan de la siguiente manera:

$$\begin{aligned} v_{max} &= 0 \\ w_{max} &= 0 \\ v_{min} &= \text{robot\_}v_{max} \\ w_{min} &= v_{min}/\text{radio\_trayectoria} \end{aligned} \tag{2.41}$$

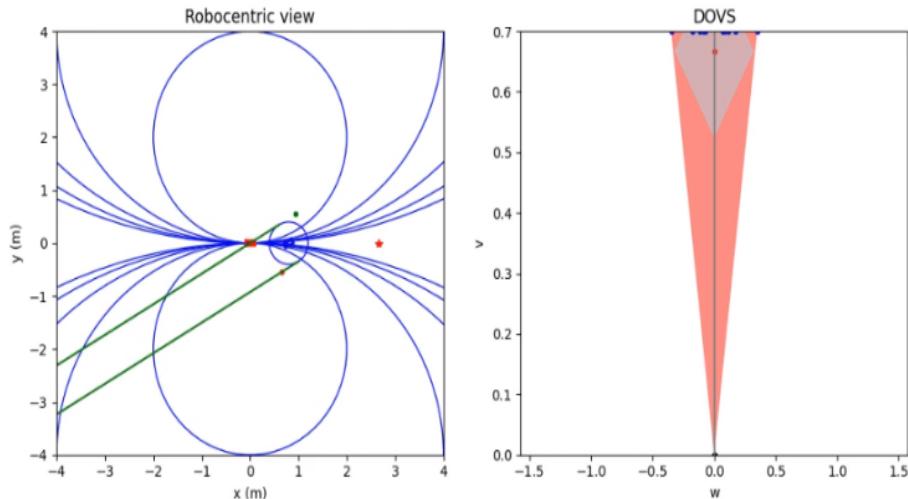


Figura 2.11: Momento justo antes de colisionar, el obstáculo está tan cerca que no se detectan colisiones, por ello se considera que todas las trayectorias llevan a colisión

En el caso de que solo tengamos un punto de colisión, este se considera que establece la velocidad máxima, ya que la velocidad mínima para salir de la banda de colisión no existe al no existir el segundo punto de colisión. Este caso puede observarse en la figura 2.12, en las trayectorias de menor radio se observa que únicamente hay un punto de colisión, ya que la trayectoria del obstáculo no colisiona con la otra línea que forma la trayectoria del obstáculo. Por lo tanto, se le asigna a la velocidad mínima el valor máximo posible, es decir:

$$\begin{aligned} v_{min} &= \text{robot\_}v_{max} \\ w_{min} &= v_{max}/\text{radio\_trayectoria} \end{aligned} \tag{2.42}$$

La velocidad máxima la calculamos utilizando la Ecuación 2.48.

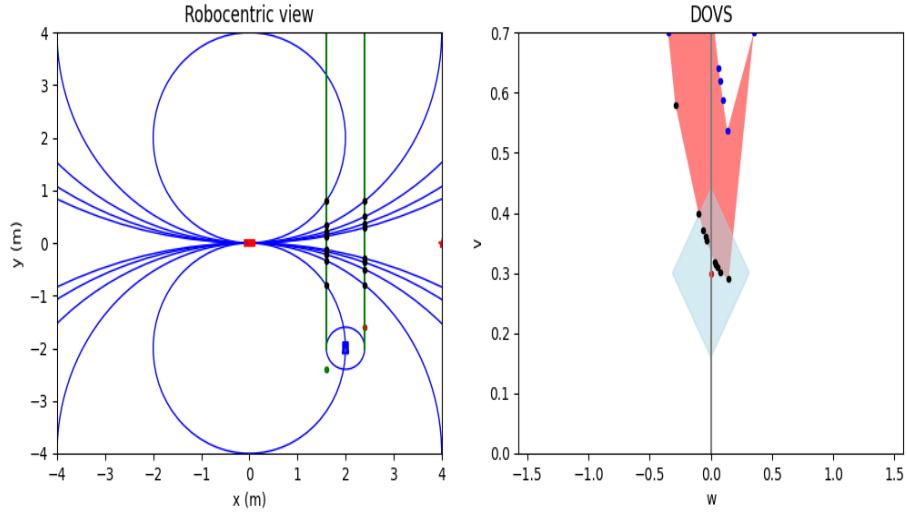


Figura 2.12: Ejemplo de situación con los puntos de colisión que se consideran

Esto cambia en el caso de que el robot se encuentre en la banda de colisión. En este caso, el punto de colisión establece la velocidad mínima que debe llevar para salir de la banda de colisión, este es el caso reflejado en la figura 2.13, por lo tanto, a la velocidad máxima se le asigna 0:

$$\begin{aligned} v_{max} &= 0 \\ w_{max} &= 0 \end{aligned} \tag{2.43}$$

La velocidad mínima la calculamos utilizando la Ecuación 2.48.

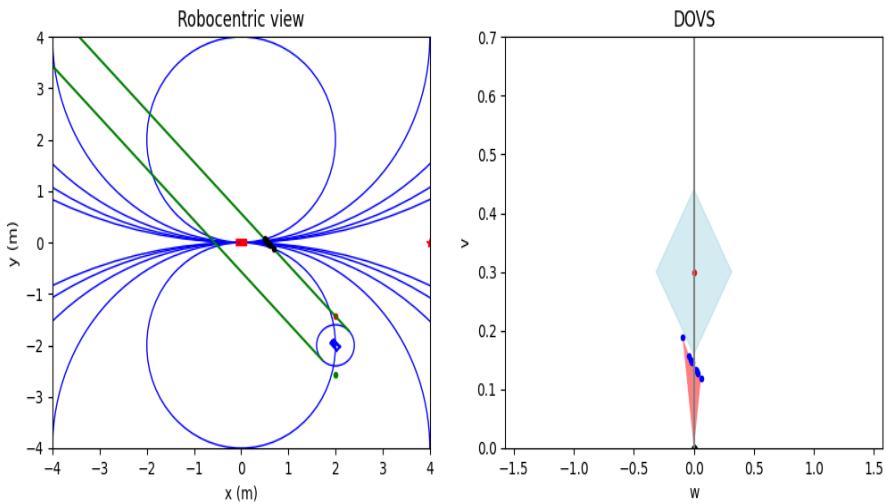


Figura 2.13: Ejemplo de situación con los puntos de colisión que se consideran

Si se tienen ambos puntos de colisión, hay que distinguir cuál asigna valor a la velocidad máxima y cuál a la velocidad mínima. Para ello, se usa la longitud de arco (*arc length*, calculado con 2.31) de ambos puntos de colisión. El punto que tenga un

*arclength* menor marca la velocidad máxima, ya que esta marca la entrada en la banda de colisión, y el punto de colisión más lejano marca la salida y por tanto la velocidad mínima.

Un ejemplo de esta situación puede observarse en la figura 2.12. En este caso la línea izquierda que forma la trayectoria del obstáculo marca la entrada del robot en la banda de colisión, y por tanto marca la velocidad máxima que el robot puede llevar para no colisionar con el obstáculo. La línea derecha marca la de la salida del robot de la banda de colisión, y por tanto marca las velocidades mínimas que el robot debe llevar para pasar antes que el obstáculo.

Estas velocidades pueden verse reflejadas el DOVS, los puntos azules, corresponden a las velocidades mínimas calculadas y los puntos negros a las velocidades máximas.

El cálculo de las velocidades en este caso se hace con la fórmula 2.48.

Para el cálculo de las velocidades, se necesita el cálculo de la distancia entre el punto de colisión y el obstáculo. El punto en el obstáculo sobre el que se calcula la distancia depende de si ese punto de colisión corresponde a pasar por detrás (se calcularía la distancia respecto al punto calculado mediante la ecuación 2.16) o por delante del obstáculo (se calcularía la distancia respecto al punto calculado mediante la ecuación 2.15). Además, el cálculo de la distancia depende del tipo de trayectoria del obstáculo, ya que esta se calcula sobre la trayectoria del obstáculo.

Fórmula para trayectorias rectilíneas:

$$distancia = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (2.44)$$

donde:  $x_1, y_1$  es el punto de colisión en la referencia del robot,  $x_2, y_2$  es el punto de colisión en el obstáculo en la referencia del robot y *distancia* es la distancia calculada sobre la trayectoria rectilínea del obstáculo.

Fórmula para trayectorias curvilíneas:

$$d = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2} \quad (2.45)$$

$$\theta = 2 \cdot \arctan \left( \frac{d}{2 \cdot radius} \right) \quad (2.46)$$

$$distancia = radius \cdot \theta \quad (2.47)$$

donde:  $x_1, y_1$  es el punto de colisión en la referencia del robot,  $x_2, y_2$  es el punto de colisión en el obstáculo en la referencia del robot, *d* es la distancia Euclídea entre  $(x_1, y_1)$  y  $(x_2, y_2)$ , *radius* es el radio de la trayectoria del obstáculo,  $\theta$  es el ángulo entre  $(x_1, y_1)$  y  $(x_2, y_2)$  y *distancia* es la distancia calculada sobre la trayectoria rectilínea del obstáculo.

Una vez que se ha calculado la distancia, las velocidades se calculan de la siguiente manera:

$$\begin{aligned} t &= \frac{\text{distancia}}{v_{obstaculo}} \\ w &= \frac{\text{angulo}}{t} \\ v &= \text{radio\_trayectoria} * w \end{aligned} \tag{2.48}$$

En las ecuaciones 2.48, se muestran las fórmulas para calcular las velocidades y el tiempo en función de la distancia, la velocidad del obstáculo, el ángulo del punto de colisión y el radio de la trayectoria.

# Capítulo 3

## Implementación de la librería DOVS

Este capítulo describe cómo se ha implementado la librería DOVS que permite crear el modelo. Para implementarlo se ha usado el lenguaje de programación Python. La decisión de utilizar este lenguaje se debe principalmente a dos factores:

- Sintaxis clara y legible: Lo cual facilita el mantenimiento y la colaboración. De esta manera, se busca que se puedan modificar y añadir fácilmente nuevas funcionalidades. Además de garantizar la fácil inclusión de un planificador que utilice el modelo creado.
- Amplia disponibilidad de bibliotecas y frameworks: Python cuenta con una gran cantidad de bibliotecas y frameworks que facilitan el desarrollo. De esta manera no se tiene que implementar funcionalidades ya implementadas. Lo cual hace también el código mucho más legible.

La claridad, modularidad, extensibilidad y simplicidad que aportan Python y un buen diseño, permite a personas ajena a la implementación, aprovecharse del modelo para abrir líneas de investigación, pudiendo escalar el proyecto. Siendo este trabajo una base para ampliar en el futuro el modelo DOVS a una navegación de drones en 3-D y navegación multi-robot con toma de decisiones compartida.

La implementación de la librería ha sido la mayor carga de trabajo de este proyecto y por eso se le ha dedicado mucho tiempo a conseguir un correcto diseño.

En la sección de diseño (Sec. 3.1) se explica la arquitectura del sistema software. En la sección 3.2 se abordan las partes más importantes de la implementación.

### 3.1. Diseño

Esta sección describe el diseño del software desarrollado para dar una idea de la estructura del sistema global y cómo se relacionan cada una de las partes.

### 3.1.1. Diagrama de paquetes

En la figura 3.1 se muestra, mediante un diagrama de paquetes, la organización de los diferentes módulos y paquetes de la librería implementada.

Todo el código aquí descrito se encuentra en un repositorio público de GitHub[5].

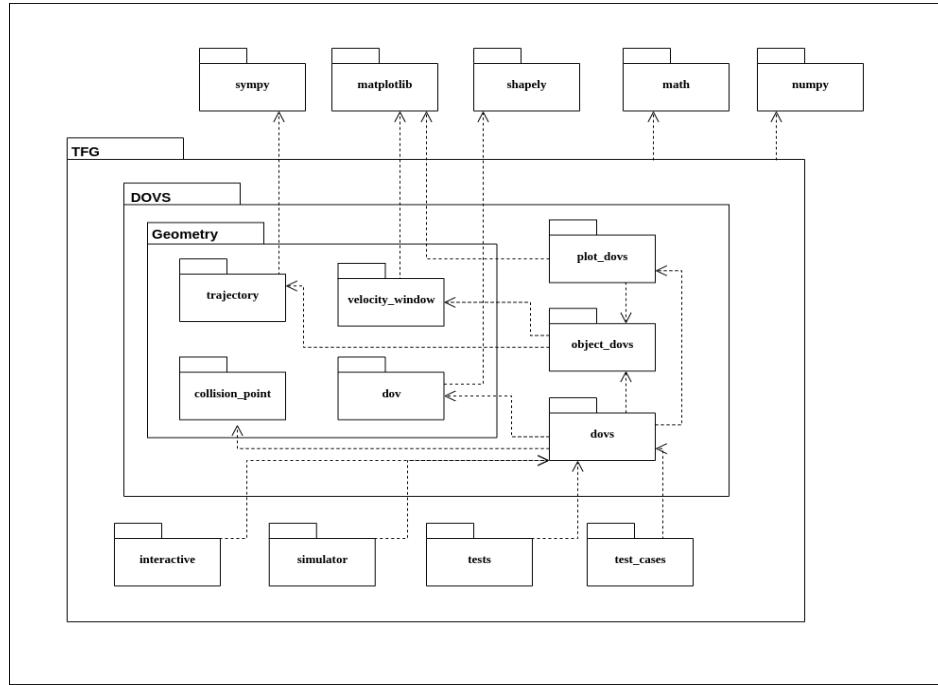


Figura 3.1: Diagrama de paquetes

A continuación, se encuentra una descripción de cada uno de los módulos.

- **dovs**: Es el módulo principal de la librería. Contiene la función que genera todo el DOVS y llama al resto de módulos.
- **object\_dovs**: Contiene todas las clases relacionadas con los objetos que forman el entorno.
- **plot\_dovs**: Contiene las clases que permiten generar gráficas con la información calculada por el resto de los paquetes. Como es el caso del DOVS.
- **trajectory**: Paquete con los distintos tipos de trayectorias de los componentes del modelo.
- **collision\_point**: Paquete con la información relativa a los puntos de colisión que llevan a calcular el DOVS.
- **dov**: Módulo que contiene la información sobre las velocidades prohibidas del robot.

- **velocity\_window**: Define la ventana de velocidad del robot.
- **simulator**: Programa *main*, simula el movimiento de una situación de robots y obstáculos generando el DOVS.
- **interactive**: Programa *main*, dada una situación de los objetos del DOVS permite elegir interactivamente las velocidades del robot en base a la gráfica del DOVS en cada iteración.
- **tests**: Programa *main*, que ejecuta un test definido en el módulo *test\_cases*. El test consiste en un entorno, e información adicional con lo que se espera que pase.
- **test\_cases**: Lista de tests que luego pueden ser utilizados por el módulo *tests*

Este es el código que se ha implementado. Para la implementación de muchas de las partes se ha utilizado una serie de paquetes externos que simplifican la implementación.

Paquetes externos utilizados:

- **matplotlib**: Matplotlib es una librería de visualización de datos en Python que proporciona una amplia gama de herramientas para crear gráficos estáticos, interactivos y de alta calidad con facilidad.

En este proyecto se ha utilizado para dos cosas:

- Visualización de gráficos.

- La ventana de velocidades del robot está definida como *matplotlib.patches.Polygon*. [6]

- **sympy**: SymPy es una librería de Python para matemáticas simbólicas que permite realizar cálculos simbólicos, como manipulación algebraica, diferenciación, integración, resolución de ecuaciones y más. Además, proporciona clases y funciones para trabajar con objetos geométricos, como puntos, líneas, círculos y polígonos.

Se ha utilizado para la implementación de las trayectorias de los objetos. [7]

- **shapely**: Shapely es una librería de Python que proporciona herramientas para la manipulación y análisis de geometrías espaciales, como puntos, líneas y polígonos. Permite realizar operaciones espaciales como intersecciones, uniones y cálculos de áreas y distancias.

Se ha utilizado para la implementación del DOV. [8]

- **numpy**: NumPy es una librería fundamental en Python para el cálculo numérico y la manipulación eficiente de matrices y arreglos multidimensionales. Proporciona una amplia gama de funciones y herramientas para realizar operaciones matemáticas y estadísticas en conjuntos de datos numéricos, incluyendo álgebra lineal, generación de números aleatorios, operaciones de indexación y segmentación, entre otros.
- **math**: La librería "math" es un módulo incorporado en Python que proporciona funciones matemáticas y constantes. Permite realizar operaciones matemáticas comunes como funciones trigonométricas, exponenciales, logarítmicas, redondeo, raíces cuadradas, entre otros.

Otras librerías para trabajar con objetos geométricos que se han planteado:

**PyGeos**: Una librería rápida y eficiente que proporciona una interfaz Python para la biblioteca GEOS (Geometry Engine - Open Source). Permite realizar operaciones geométricas avanzadas como intersecciones, uniones, diferencias y simplificaciones en objetos geométricos.

**GeoPandas**: Librería que combina las funcionalidades de las librerías Pandas (para manipulación de datos tabulares) y Shapely (para geometrías) para ofrecer una forma conveniente de trabajar con datos geoespaciales. Permite leer, escribir, manipular y analizar datos geoespaciales en formatos como Shapefile, GeoJSON y otros.

Al final se ha considerado que tanto Shapely como SymPy se acerca más a las necesidades de este proyecto por las siguientes razones:

**Flexibilidad y facilidad de uso**: Shapely se centra exclusivamente en la manipulación y análisis de geometrías, lo que lo hace más fácil de aprender y utilizar para tareas específicas relacionadas con geometría. Proporciona una amplia gama de operaciones geométricas y su sintaxis es simple y directa.

**Soporte para geometrías complejas**: Shapely permite trabajar con geometrías más complejas, como curvas y superficies, además de los objetos geométricos básicos. Esto puede ser útil si necesitas realizar operaciones avanzadas y modelar formas más complejas en tus datos.

**Integración con otros proyectos de Python**: SymPy se integra bien con otras bibliotecas y proyectos de Python, como SciPy, NumPy y matplotlib. Esto te permite combinar el poder de SymPy con otras herramientas para realizar análisis numéricos y visualizaciones de manera más eficiente.

Además, tanto Shapely como SymPy tienen una base sólida de usuarios y cuentan con soporte activo.

Existen numerosos recursos en línea, como documentación oficial, tutoriales, ejemplos de código y foros de discusión, que ayudan a resolver problemas.

### 3.1.2. Diagrama de clases

La estructura de las clases implementadas se ha definido en el Anexo C.1

## 3.2. Implementación del DOVS

En esta sección se comenta las partes más importantes de la implementación.

La función más importante del trabajo es *compute\_DOVS()* de la clase *DOVS* ya que es esta la que se encarga de crear el modelo.

El diagrama de secuencia de esta función puede encontrarse en el Anexo C.2

El algoritmo en Python de esta función puede encontrarse en el Anexo B.1

Dentro de esta función las dos partes principales son las comentadas en el apartado 2.4, el cálculo de los puntos de colisión y el de las velocidades a colisión.

Para entender mejor cómo funcionan y cómo se han implementado ambas partes, se han incluido unos diagramas.

En el Anexo C.3 se encuentra el diagrama de secuencias que describe como se ha implementado la generación y selección de los puntos de colisión a partir de los cuales luego se han calculado el conjunto de velocidades prohibidas.

El algoritmo en Python de este diagrama (figura C.3) puede encontrarse en el Anexo B.2

En el Anexo C.4, en la figura C.4, puede observarse el diagrama de flujo de la función que se encarga de, dados los puntos de colisión, calcular la velocidad mínima que debe llevar el robot para pasar por delante del obstáculo, y la velocidad máxima que puede llevar el robot para no chocarse.

El algoritmo en Python de este diagrama (figura C.4) puede encontrarse en el Anexo B.3



# Capítulo 4

## Validación experimental

Uno de los objetivos de este trabajo es realizar una evaluación exhaustiva de modelo en diferentes escenarios y situaciones. Se han creado una serie de experimentos que permiten evaluar el correcto funcionamiento del código implementado. En ellos se verifica que la elección de velocidades representadas en el modelo como seguras, son efectivamente velocidades que generan trayectorias sin colisión, bien porque el robot pasa delante de los obstáculos, bien porque los obstáculos pasan antes de que llegue el robot a la banda de colisión. También se evalúa que la elección de velocidades dentro del DOV, es decir velocidades no seguras, conducen a colisión en un horizonte temporal. Se elige un conjunto suficientemente representativo de ambos conjuntos de velocidades, que permite asegurar que el modelo está bien construido.

Los distintos programas implementados, que generan un entorno y lo simulan, para probar la validez del código, se explicaron con anterioridad en la sección 3.1.1.

Las pruebas generadas van en un nivel de dificultad creciente. Y al final se prueban casos específicos sobre los que se quiere asegurar el correcto funcionamiento.

Se ha creado una lista de reproducción en YouTube con vídeos del conjunto de pruebas realizado [9].

### 4.1. Experimento 1. Pasar por detrás de un obstáculo

En esta prueba la idea es ver si el robot, seleccionando velocidades por debajo del polígono de velocidades prohibidas, es capaz de llegar a la meta pasando por detrás del objeto. El objeto sigue una trayectoria rectilínea.

En la figura 4.1 puede verse la situación inicial de los objetos. En la figura 4.3 se puede ver como el tamaño de del área de las velocidades prohibidas ha aumentado respecto a la situación inicial 4.2. En concreto se puede observar cómo toda la parte superior queda cubierta. Esto se debe a que al estar tan cerca, si el robot escogiera

velocidades altas, incluso la máxima, colisionaría, ya que ya no tiene tiempo para pasar por delante del obstáculo.

Puede observarse en la figura 4.4, como una vez que pasa el obstáculo, ya no hay velocidades prohibidas.

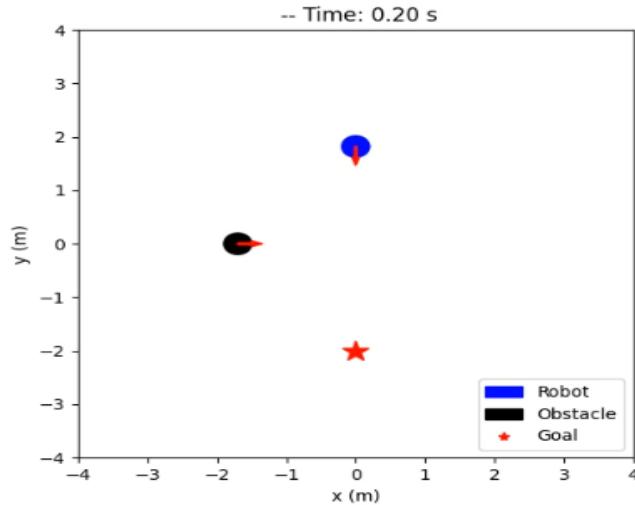


Figura 4.1: Experimento 1, entorno del momento inicial

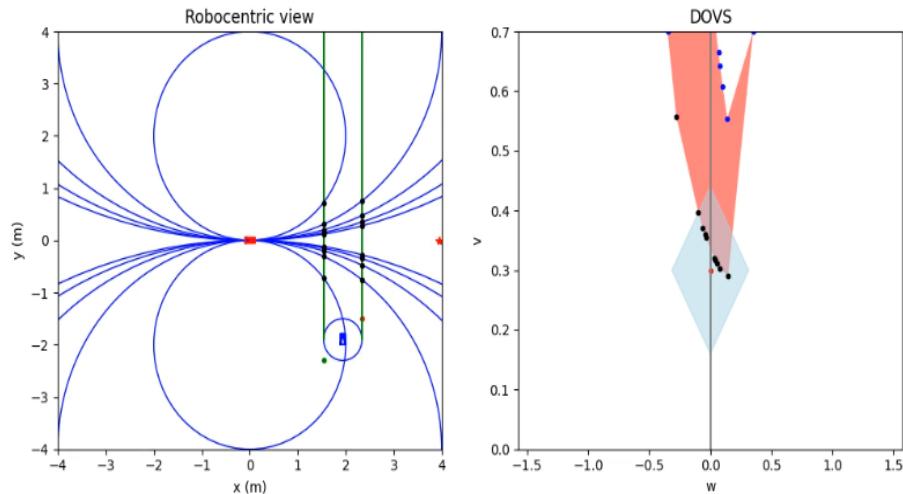


Figura 4.2: Experimento 1, DOVS del momento inicial

## 4.2. Experimento 2. Pasar por delante de dos obstáculos

En esta prueba se han seleccionado velocidades por encima del polígono de velocidades prohibidas que nos permiten llegar a la meta pasando por delante de los obstáculos. En la figura 4.5 puede verse la situación inicial de los objetos.

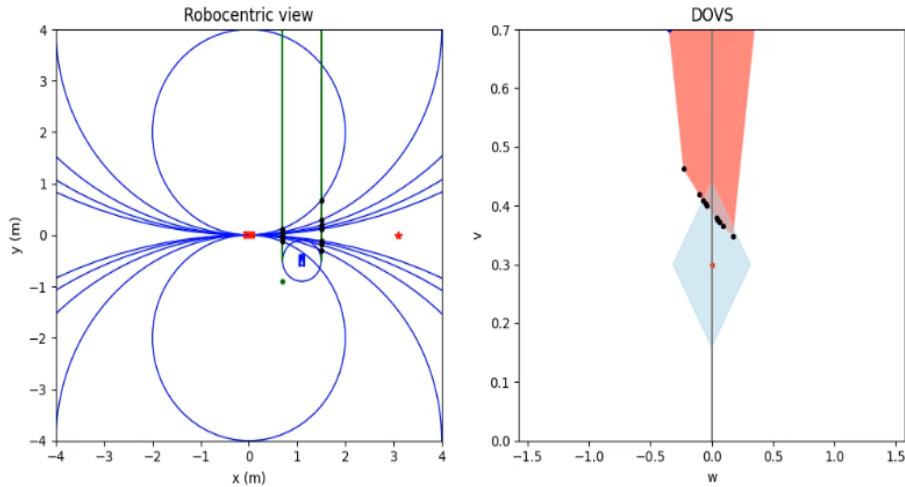


Figura 4.3: Experimento 1, cerca del robot

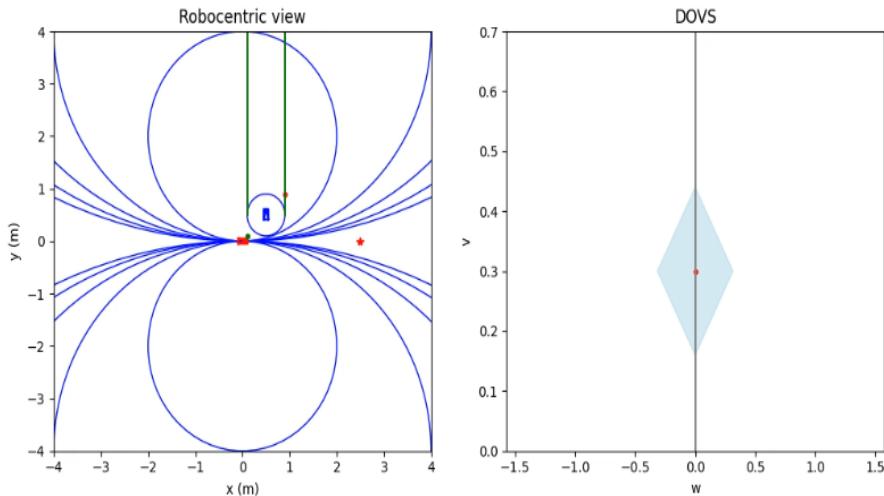


Figura 4.4: Experimento 1, obstáculo ya ha pasado

En la primera figura 4.6, se puede observar el DOV de dos obstáculos combinados. En las gráficas que se muestran los puntos azules corresponden a las velocidades calculadas que permiten pasar por delante del obstáculo y los puntos negros a las velocidades máximas que puede llevar el robot para no colisionar. Fijándose en los puntos se puede observar cómo se han juntado dos polígonos, uno que queda más abajo a la derecha y otro que queda más arriba.

Uno de los obstáculos sigue una trayectoria rectilínea y el otro una curvilínea.

Puede observarse en la figura 4.7 cómo la separación entre los polígonos es más clara encontrándose el del obstáculo con la trayectoria rectilínea en la zona más baja del DOVS. Esto se debe a que el obstáculo va muy lento, por lo que, aun estando relativamente cerca el obstáculo, el robot tendría que ir muy lento para colisionar.

En la figura 4.8 puede observarse que cuando el robot se encuentra dentro de la banda de colisión del obstáculo el robot interpreta esos puntos de colisión como la velocidad mínima que debe llevar para salir de esa banda. Esto implica que las velocidades máximas serán 0, por ello se ven muchos puntos negros y se calculan las velocidades mínimas. Que como puede observarse salen altas ya que el obstáculo está cerca del robot y lleva bastante velocidad.

A esto se deben los puntos altos de velocidades prohibidas con velocidad angular negativa. Ya que si el robot girara rápidamente un poco a la derecha se encontraría con el obstáculo.

En la última figura 4.9 puede observarse como ya una vez que ha pasado el obstáculo, apenas hay velocidades prohibidas. Aún se encuentran puntos de colisión con el obstáculo con trayectoria rectilínea, que, al encontrarse el robot dentro de la banda de colisión, se consideran como velocidades mínimas que debe tener para escapar de la banda. Estas velocidades son muy bajas por lo que ya se ha comentado, la velocidad del obstáculo es muy baja.

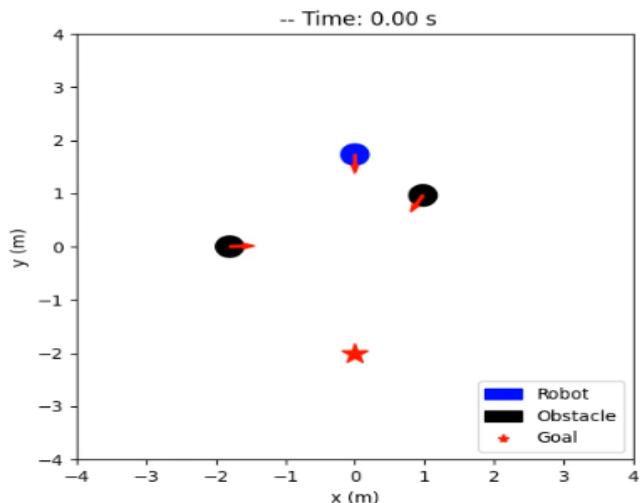


Figura 4.5: Experimento 2, entorno del momento inicial

### 4.3. Experimento 3. Colisión con obstáculo, trayectoria curvilínea

La situación es la misma que en el experimento 4.2, en este caso se va a seleccionar una velocidad dentro del DOV del obstáculo con trayectoria curvilínea de manera que se produzca la colisión.

En la figura 4.10 puede observarse la situación inicial del entorno y la velocidad que se va a seleccionar, que lleva a colisión.

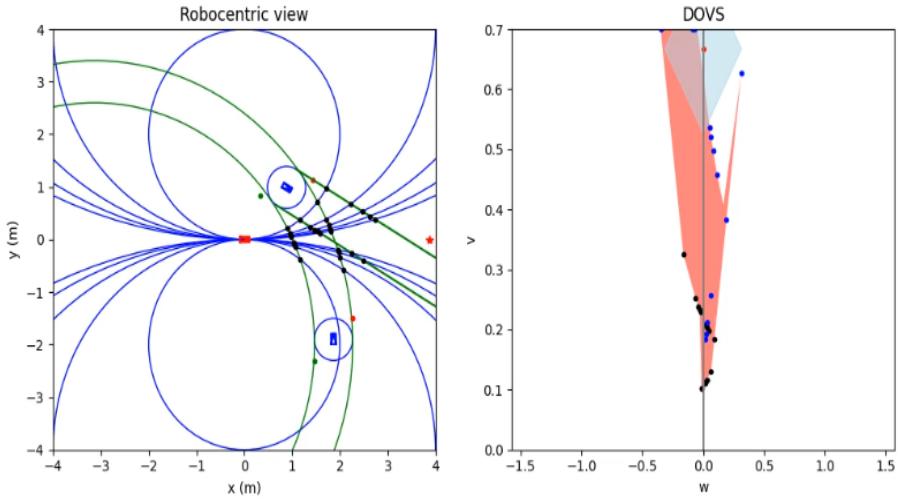


Figura 4.6: Experimento 2, DOVS del momento inicial

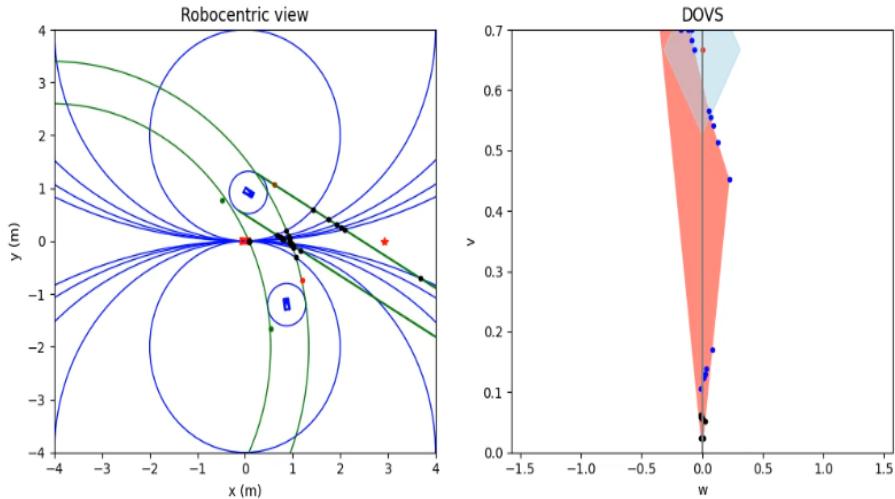


Figura 4.7: Experimento 2, situación más avanzada

Puede observarse en la figura 4.11 una situación más avanzada en la que ya es imposible pasar por delante del obstáculo y se observa un DOV más grande que el anterior.

La última figura 4.12 corresponde a momentos antes de la colisión y se puede observar cómo la situación apenas ha cambiado salvo que ahora el polígono de las velocidades prohibidas es ligeramente más grande.

#### 4.4. Experimento 4. Colisión con obstáculo, trayectoria rectilínea

La situación también es la misma que en el experimento 4.2, en este caso se va a seleccionar una velocidad dentro del DOV del obstáculo con trayectoria rectilínea de

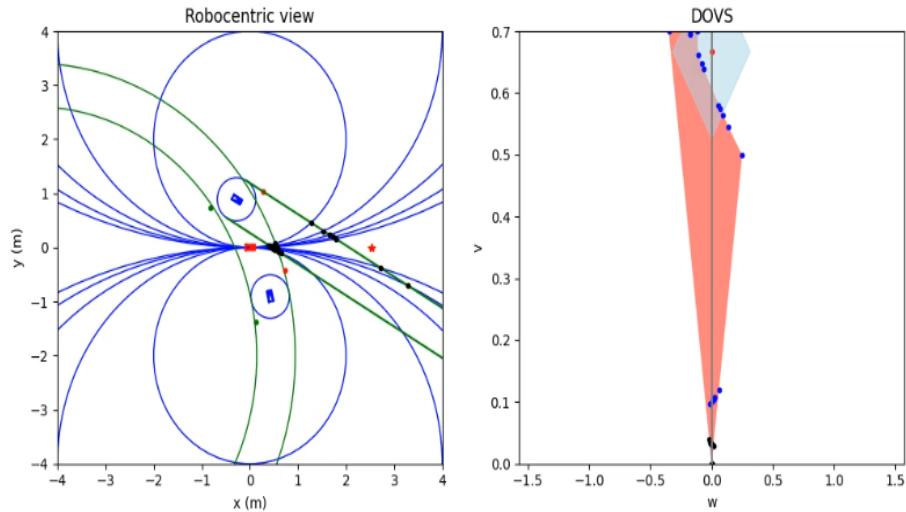


Figura 4.8: Experimento 2, robot dentro de la banda de colisión

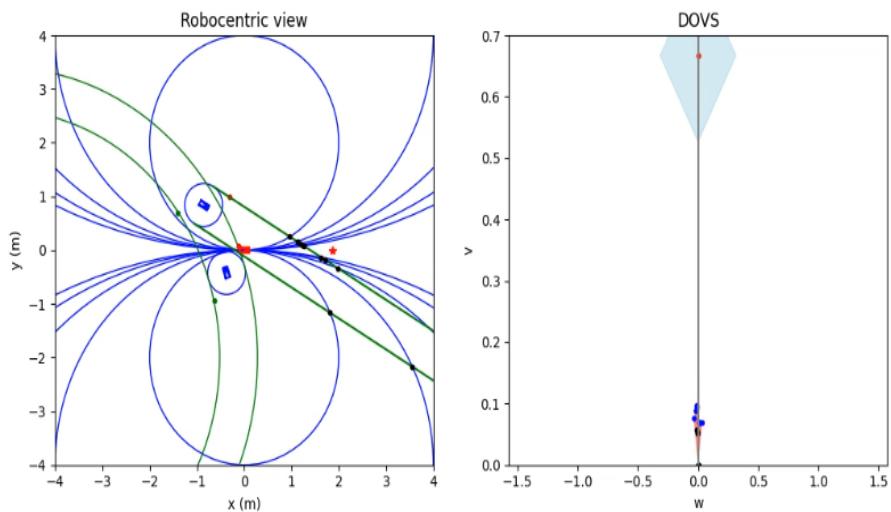


Figura 4.9: Experimento 2, obstáculo ya ha pasado

manera que se producirá la colisión.

En la figura 4.13 puede observarse la situación inicial del entorno y la velocidad que se va a seleccionar, que lleva a la colisión.

En las imágenes 4.14 y 4.15 se puede observar como va evolucionando el entorno. En la figura 4.15 el obstáculo que va más rápido ya no se tiene en cuenta porque se considera que los puntos de colisión que generan sus trayectorias están muy lejos de la situación actual del robot.

La última figura 4.16 es momentos antes de la colisión.

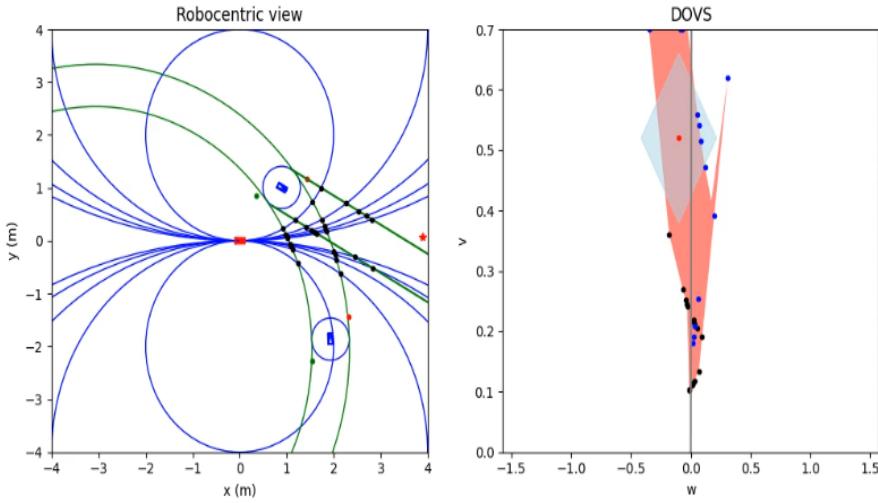


Figura 4.10: Experimento 3, DOVS del momento inicial

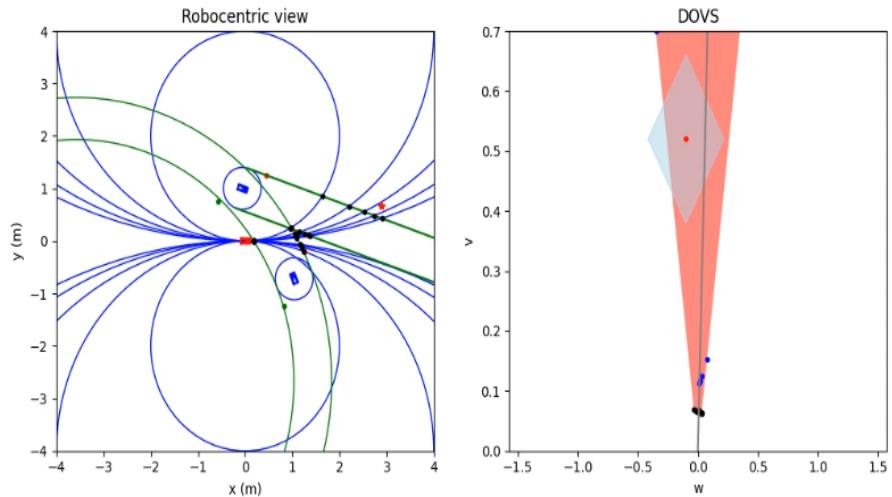


Figura 4.11: Experimento 3, situación avanzada, va a llevar a colisión

## 4.5. Experimento 5. Pasar por delante de 2 obstáculos y detrás de 1

En este experimento la situación cambia, la posición del robot es distinta y ahora tiene que sortear 3 obstáculos para llegar a la meta.

En la figura 4.17 puede verse la situación y en la figura 4.18 el DOVS generado.

En este experimento se ha seleccionado unas velocidades que pasando por de los dos primeros obstáculos y por detrás del 3 llega a la meta.

En este caso por la velocidad que lleva el último obstáculo es imposible pasar por delante de él.

En la figura 4.19 se puede observar como ya se ha pasado por delante del primer obstáculo. También es interesante observar cómo a pesar de que se detectan múltiples

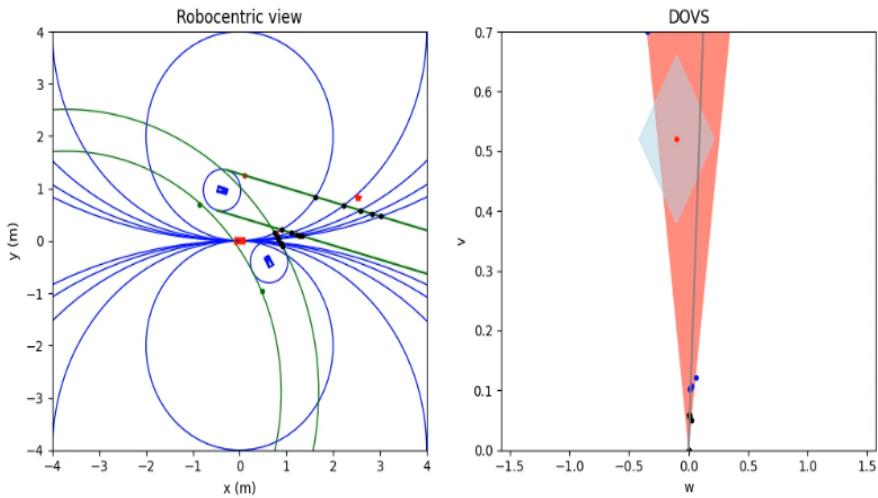


Figura 4.12: Experimento 3, momentos antes de la colisión

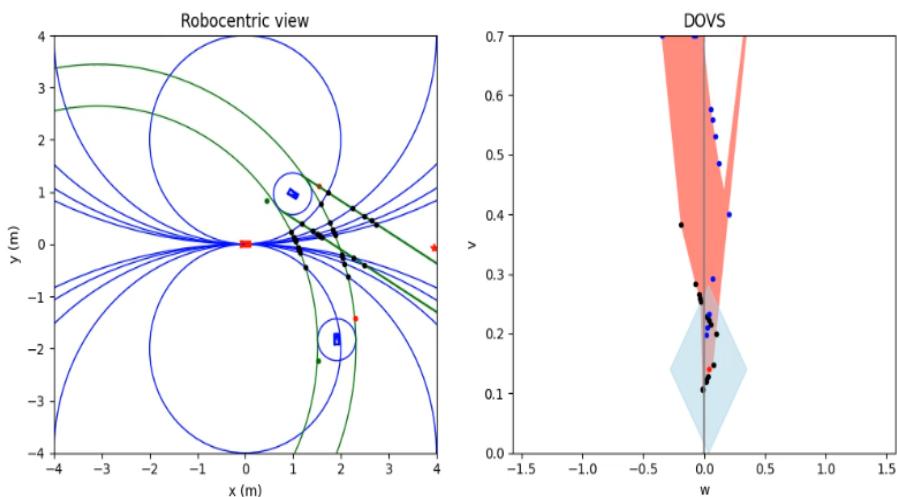


Figura 4.13: Experimento 4, DOVS del momento inicial

puntos de colisión para el último obstáculo, estas no se reflejan en el DOVS debido a que el obstáculo lleva tanta velocidad que para colisionar tendría que llevar unas velocidades superiores a las que el límite le permite. Es por esto que las velocidades prohibidas que se observan en el DOVS corresponden a las velocidades mínimas que el robot debe llevar para salir de la banda de colisión del segundo obstáculo.

La figura 4.20 muestra el instante anterior de llegar a la meta habiendo sobrepasado todos los obstáculos.

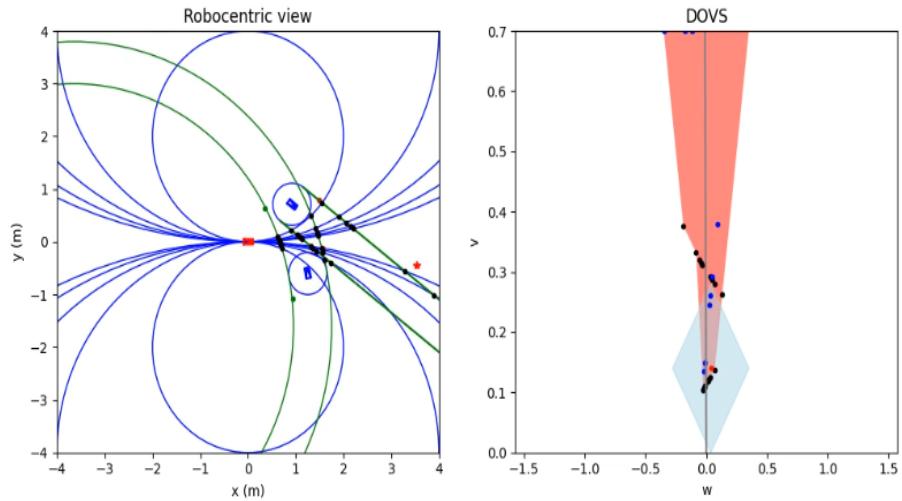


Figura 4.14: Experimento 4, situación más avanzada

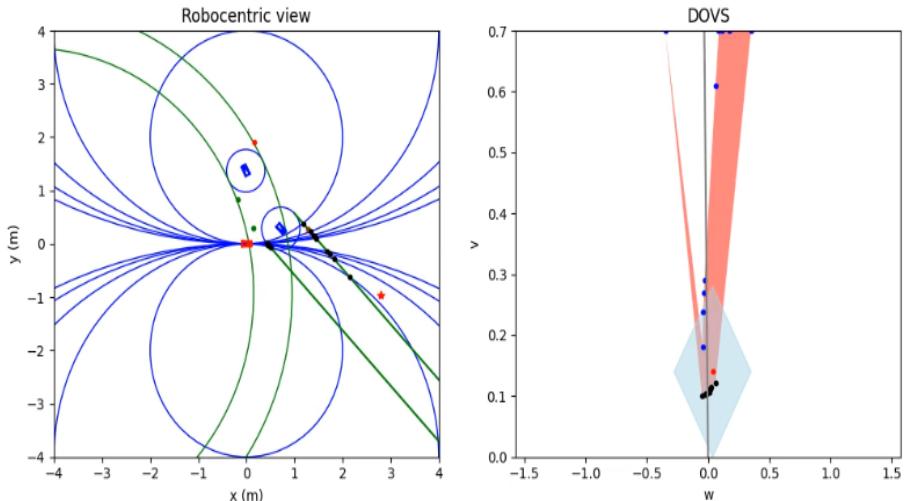


Figura 4.15: Experimento 4, un obstáculo ya ha pasado

## 4.6. Experimento 6. Pasar por detrás con 3 obstáculos

El entorno es el mismo que en el experimento 4.5, en este caso se va a seleccionar una velocidad de manera que se pasa por detrás de los 3 obstáculos.

La idea de este experimento es la misma que el de la sección 4.1, añadiendo un poco más de complejidad al ser 3 obstáculos en una situación distinta.

En la situación que representa la figura 4.22, podemos ver como es imposible que pasemos por delante de los obstáculos una vez que llegamos a esta situación.

En las imágenes 4.23 y 4.24 se puede observar como el DOVS se va haciendo más pequeño a medida que los obstáculos van pasando por delante del robot.

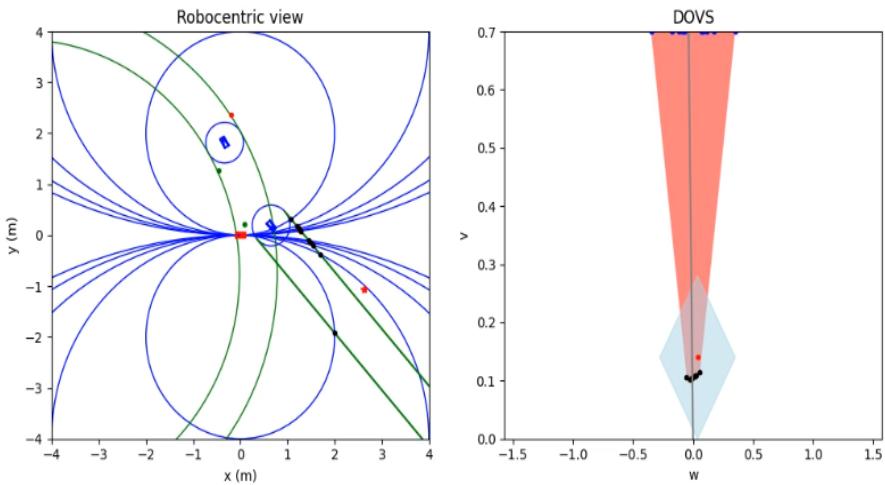


Figura 4.16: Experimento 4, momentos antes de la colisión

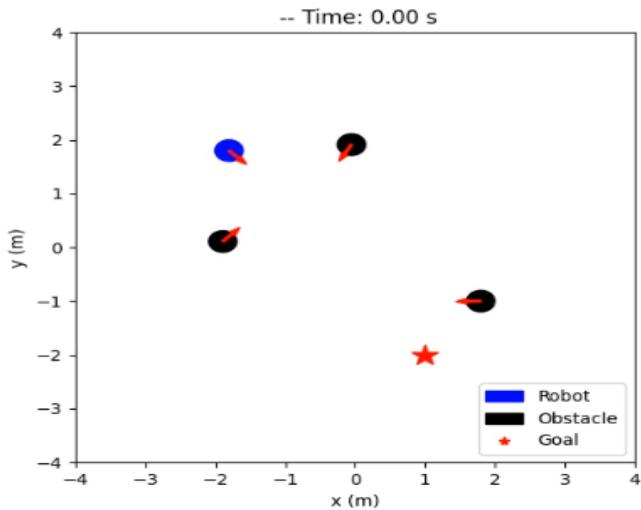


Figura 4.17: Experimento 5, entorno inicial

## 4.7. Experimento 7. Llega a la meta con trayectoria curvilínea pasando por delante de un obstáculo y por detrás de otro

En este experimento se intenta probar una distribución de los obstáculos muy distinta a las anteriores.

En este caso el robot sigue una trayectoria curva que le debería llevar a la meta.

Ambos obstáculos siguen trayectorias curvilíneas

La distribución de los objetos puede verse en la figura 4.25.

En la figura 4.28 vemos que el robot parece que se encuentra en la banda de colisión de uno de los obstáculos, pero este no es el caso, porque se ha implementado de manera que el robot sólo se encuentra en la banda de colisión si en ese instante el obstáculo

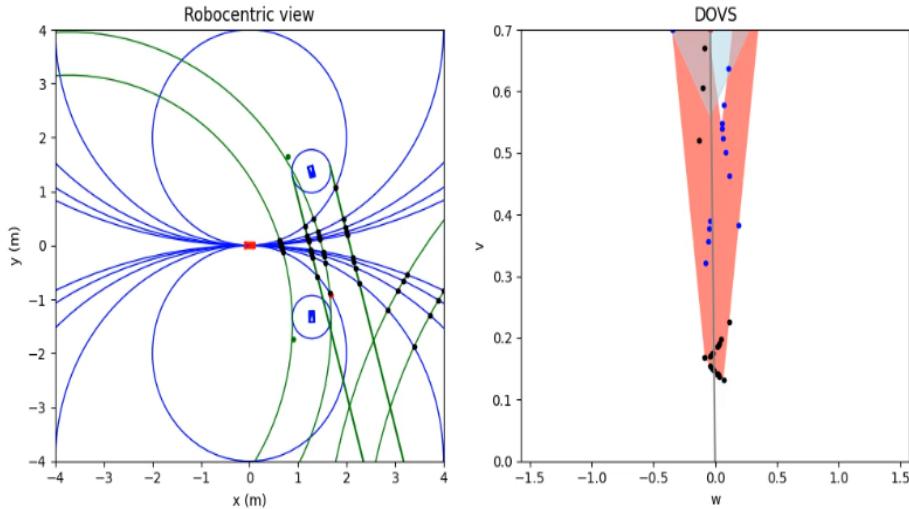


Figura 4.18: Experimento 5, momento inicial DOVS

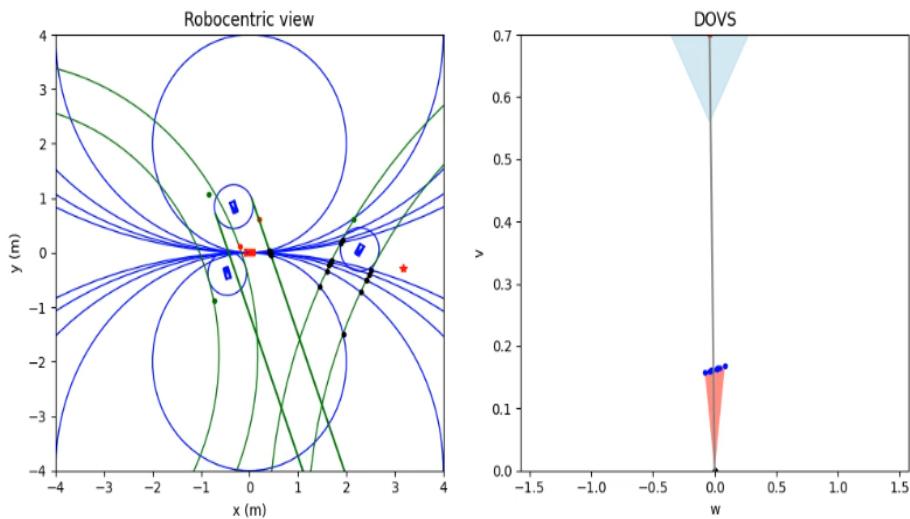


Figura 4.19: Experimento 5, situación intermedia, el robot ha pasado por delante de un obstáculo y se encuentra dentro de la banda de colisión de otro

está en la dirección del robot, esto se ha decidido así, porque aunque el obstáculo llegase a ese punto si hiciese toda su trayectoria curva este tardaría mucho tiempo en completarse por lo que solo tiene sentido considerarla si está dirigido para colisionar con el robot. Es por eso por lo que no se ha detectado ningún punto de colisión entre esas trayectorias.

En cambio el caso contrario puede observarse en la figura 4.30, en este caso el obstáculo si que está en la dirección en la que se encuentra el robot, por lo tanto los puntos de colisión que considera son los que le permitan al robot escapar de la banda de colisión, el DOVS reflejado en esa figura marca esas velocidades que nos llevarían a colisión.

En la figura final 4.31 se puede observar como ya ha sobrepasado las posibles

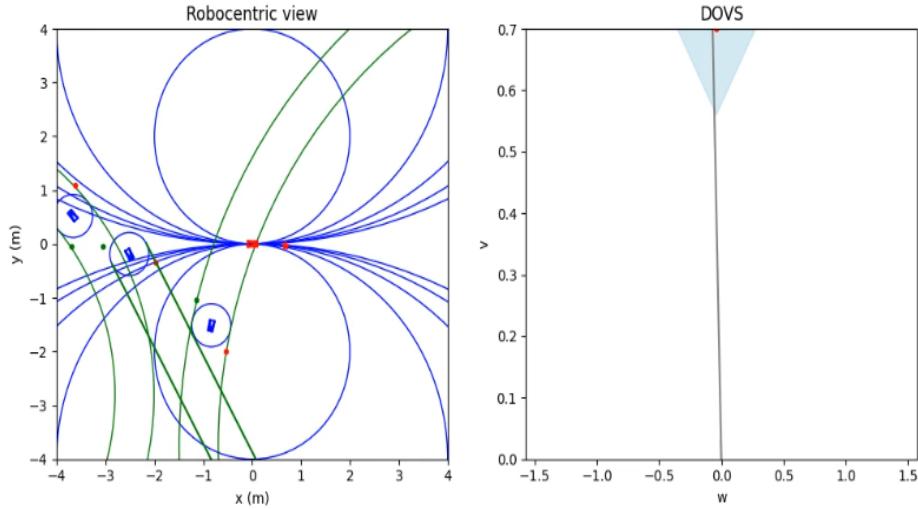


Figura 4.20: Experimento 5, instantes anteriores de llegar a la meta

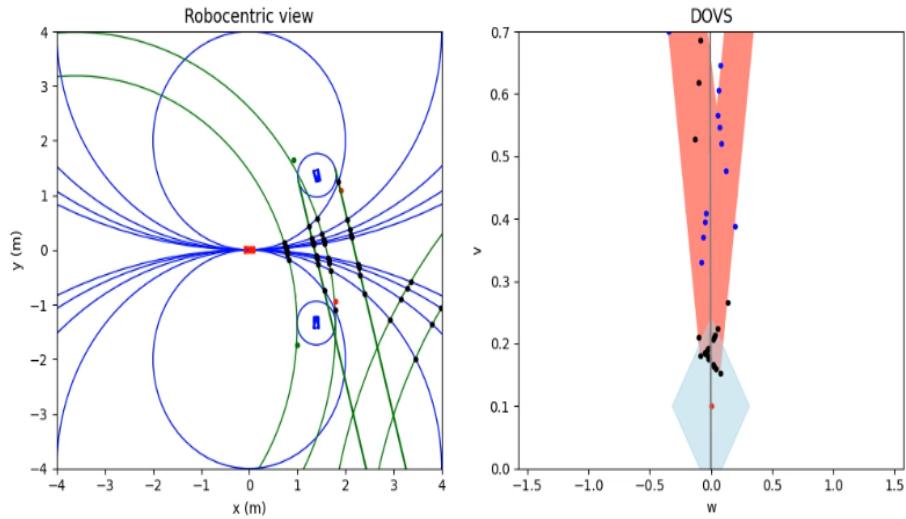


Figura 4.21: Experimento 6, DOVS del momento inicial

colisiones con los obstáculos y ya está al lado de la meta.

## 4.8. Experimento 8. Cinco obstáculos

Este experimento trata de probar el cálculo de modelo con un número elevado de obstáculos. Además se quiere probar que con un número elevado de obstáculos el robot es capaz de llegar a la meta sin colisionar si se selecciona una velocidad libre.

En todas las imágenes, de la 4.33 a la 4.37, se puede observar como la velocidad que lleva el robot no entra en ningún momento en el conjunto de velocidades prohibidas.

En la figura 4.37 se observa como acaba llegando a la meta sin colisionar con ningún obstáculo.

Además de todo esto se ha probado a añadir más trayectorias al robot, esto solo se

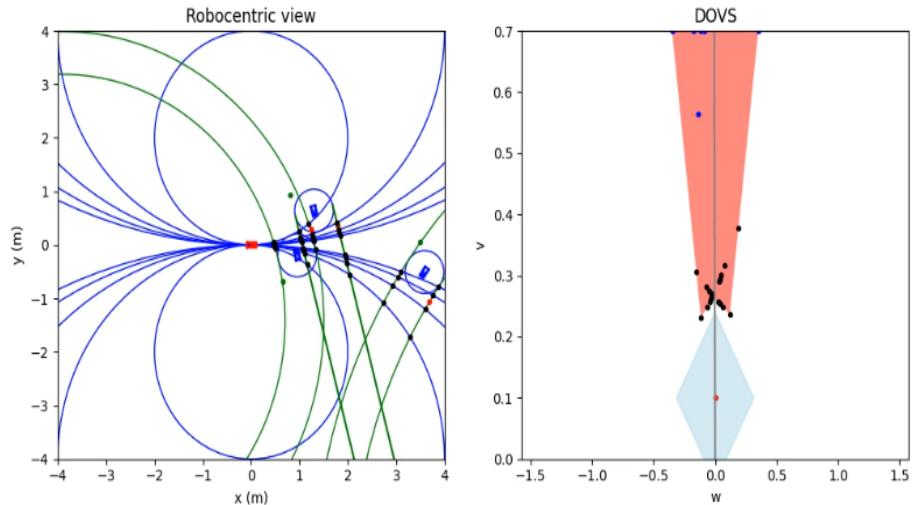


Figura 4.22: Experimento 6, momentos antes de que el primer obstáculo pase por delante del robot

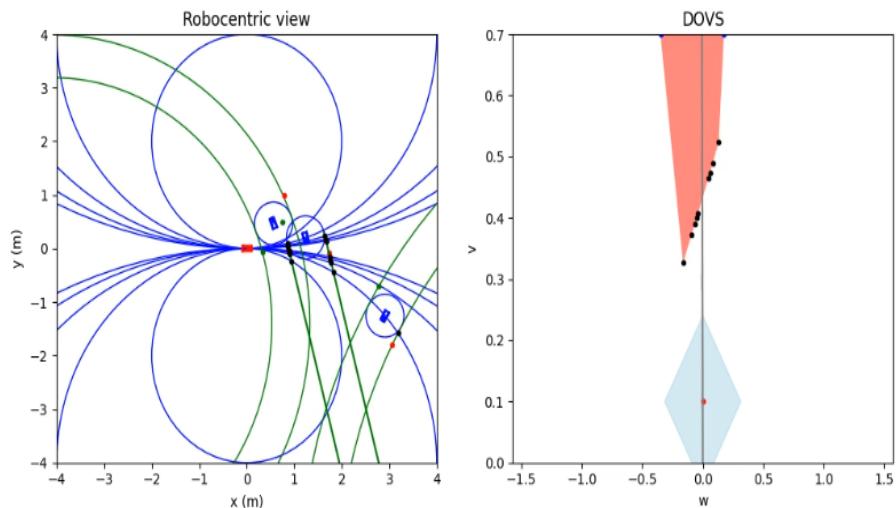


Figura 4.23: Experimento 6, el primer obstáculo ha pasado por delante del robot

ha hecho por aumentar la carga de trabajo en la generación del DOVS y observar los resultados.

Se ha observado que no cambia el resultado al añadir más trayectorias al robot, hay más muestras, por lo que se calculan más colisiones y velocidades, pero están todas muy cerca de unas de otras, por lo que no aportan gran información nueva.

## 4.9. Experimento 9. Colisión por detrás del robot

En este experimento el objetivo es probar el caso concreto de que un obstáculo se encuentra por detrás del robot.

Como puede verse en la figura 4.38 el robot no lleva la suficiente velocidad como

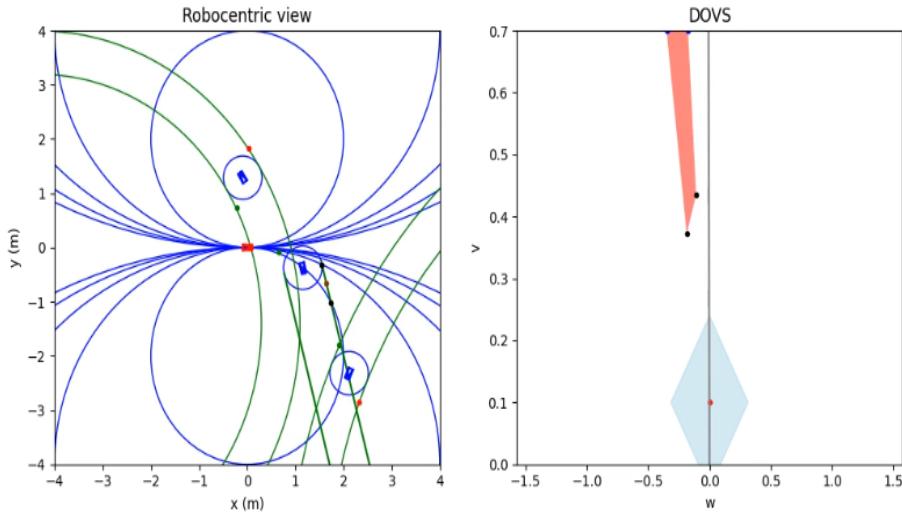


Figura 4.24: Experimento 6, básicamente ya han pasado los 3 obstáculos por delante del robot y ya no se detectan casi posibles puntos de colisión

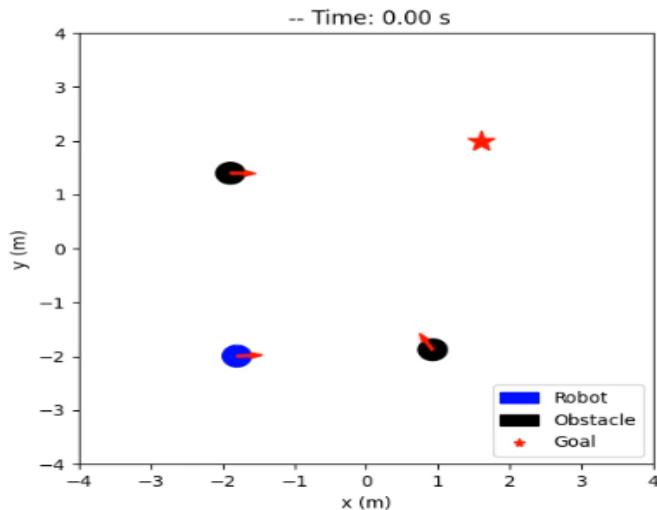


Figura 4.25: Experimento 7, entorno del momento inicial

para evadir el obstáculo.

En la figura 4.39 puede verse como en efecto el obstáculo está a punto de chocar con el robot

## 4.10. Experimento 10. Colisión frontal

En este experimento el objetivo es probar el correcto funcionamiento de la colisión frontal en el caso que no se detecten puntos de colisión debido a que el robot está muy cerca del obstáculo.

La situación de la figura 4.43 es la que queríamos comprobar, en este caso el robot está a punto de colisionar, y por la posición con el obstáculo no se detectan puntos de

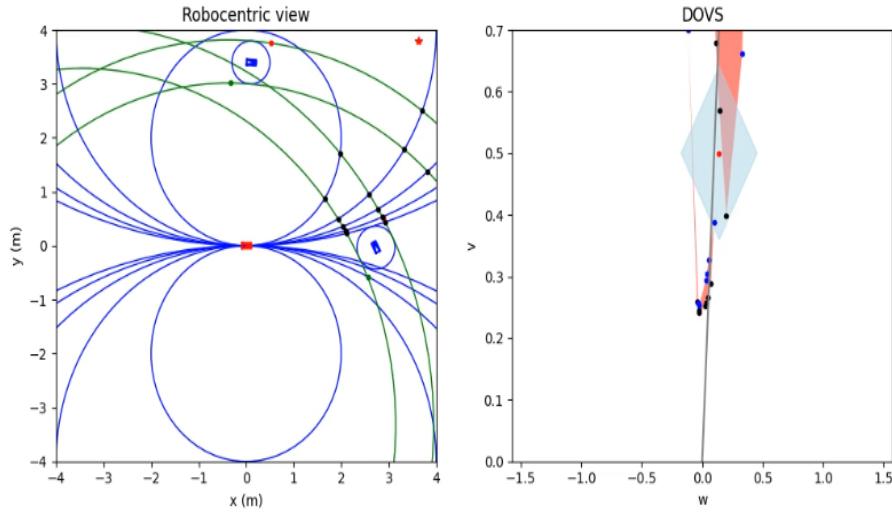


Figura 4.26: Experimento 7, DOVS del momento inicial

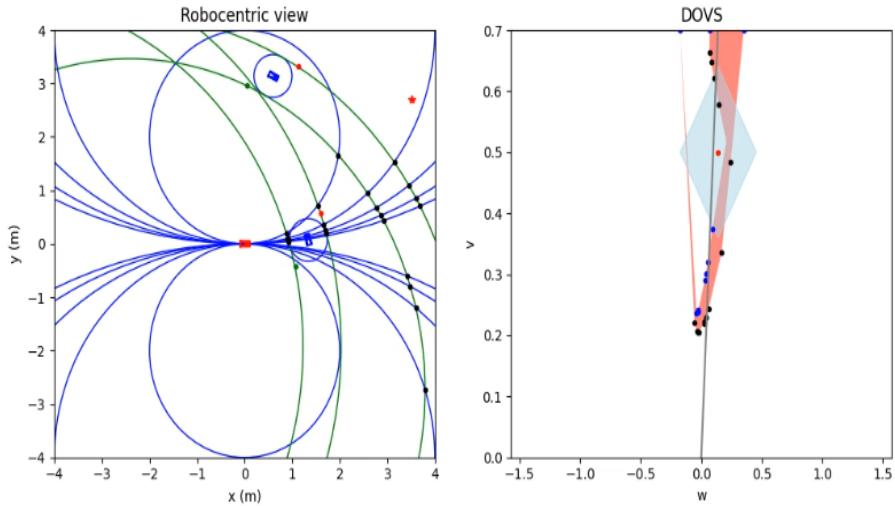


Figura 4.27: Experimento 7, justo antes de pasar por detrás del primer obstáculo

colisión, esto en otras circunstancias indicaría que el robot no va a colisionar con el obstáculo pero en este caso está claro que no.

Es por eso que en la implementación se ha añadido un caso específico en el caso de que no se detecten puntos de colisión, que es el caso de que el robot se encuentre en la banda de colisión del obstáculo, esto es una situación que solo ocurre en el caso que aquí mostramos, y lo que se hace es marcar todas las trayectorias que no tienen puntos de colisión, como trayectorias que llevan a colisión. De ahí el resultado que se observa.

## 4.11. Evaluación de resultados

La evaluación de resultados ha arrojado conclusiones positivas. Tras llevar a cabo exhaustivas pruebas en diferentes escenarios, se ha constatado de manera consistente

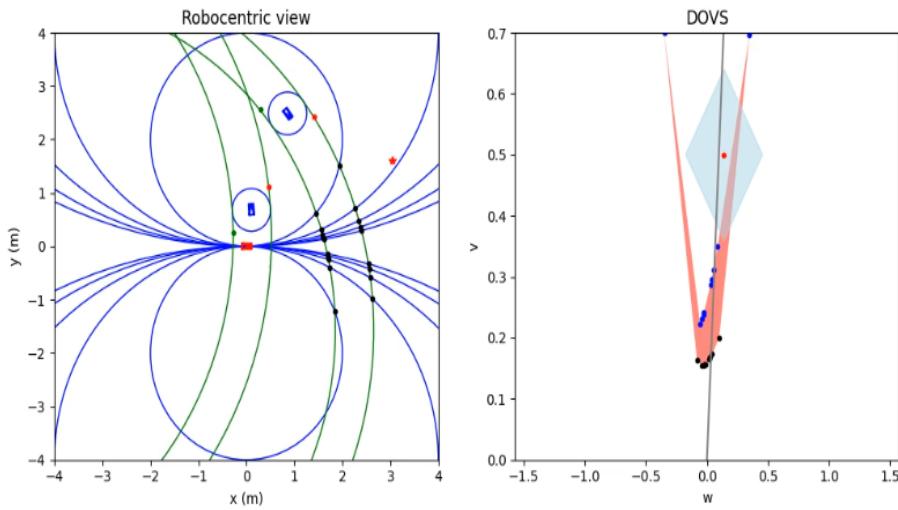


Figura 4.28: Experimento 7, el robot ha pasado por detrás del primer obstáculo

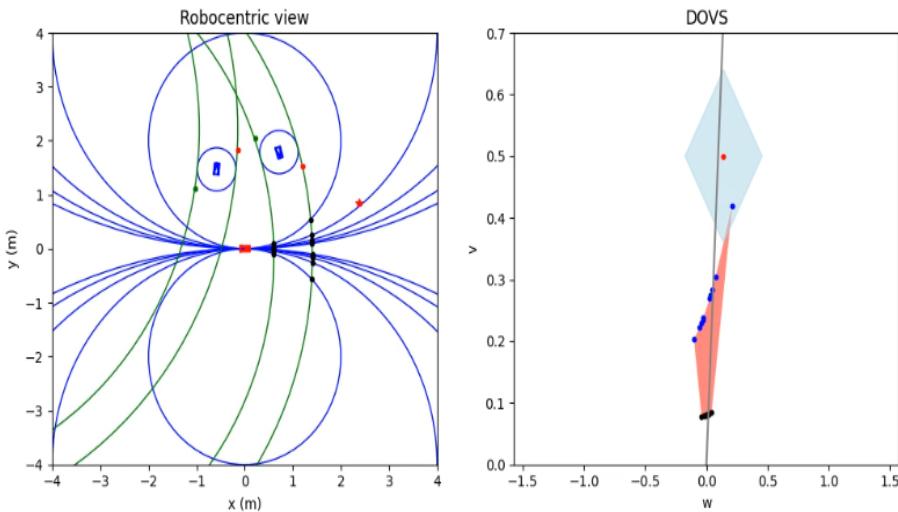


Figura 4.29: Experimento 7, el robot está cerca de pasar por delante del segundo obstáculo

que los resultados obtenidos se alinean con las expectativas previas, lo que da confianza en la efectividad del código implementado.

Al adentrarnos en casos específicos y analizarlos minuciosamente, se ha demostrado que el modelo implementado responde de manera precisa y acorde a las particularidades de cada uno de ellos. Este nivel de consistencia y fiabilidad refuerza aún más la confianza en la funcionalidad del código desarrollado.

Por tanto, podemos concluir de manera contundente que el modelado del entorno se encuentra en óptimas condiciones y desempeña su función correctamente.

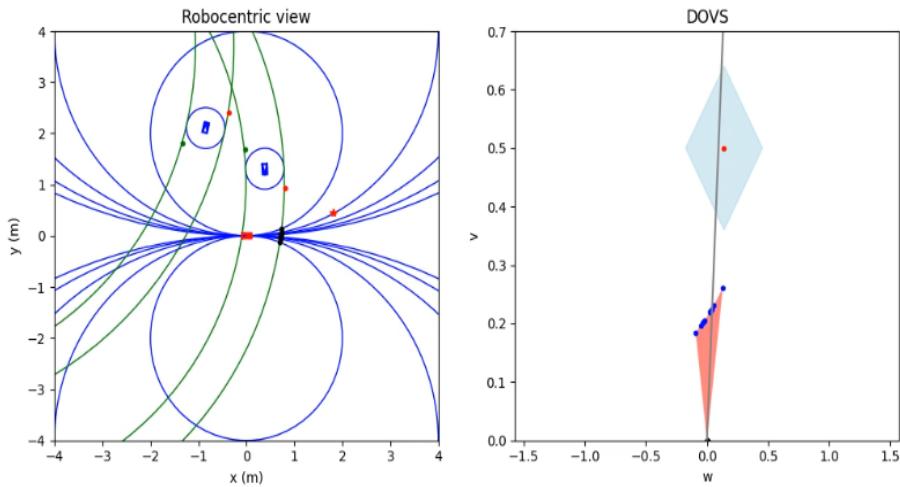


Figura 4.30: Experimento 7, el robot se encuentra dentro de la banda de colisión del segundo obstáculo, por lo que esas velocidades que se calculan corresponden a las velocidades necesarias para pasar por delante de él

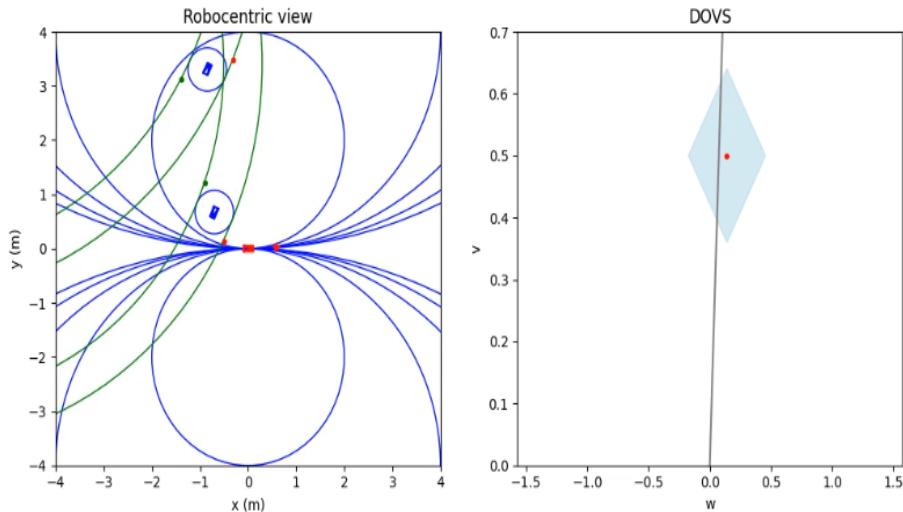


Figura 4.31: Experimento 7, el robot ya ha evitado la colisión con los obstáculos y está muy cerca de llegar a la meta

## 4.12. Problemas encontrados y soluciones

En esta sección se comentan los mayores problemas encontrados durante la resolución del trabajo, así como sus soluciones.

### Problemas matemáticos

La implementación del algoritmo del DOVS es un proceso complejo que requiere de una base matemática. Estos conceptos matemáticos y la variedad de situaciones que se presentan han supuesto bastante problema a la hora de implementar ciertas partes.

Sobre todo, han surgido problemas trigonométricos, y problemas de concepto al no saber cómo utilizar las matemáticas para resolver ciertas situaciones.

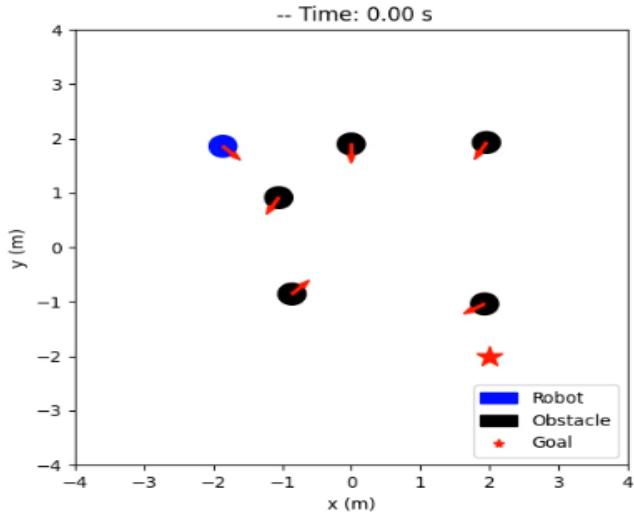


Figura 4.32: Experimento 8, entorno del momento inicial

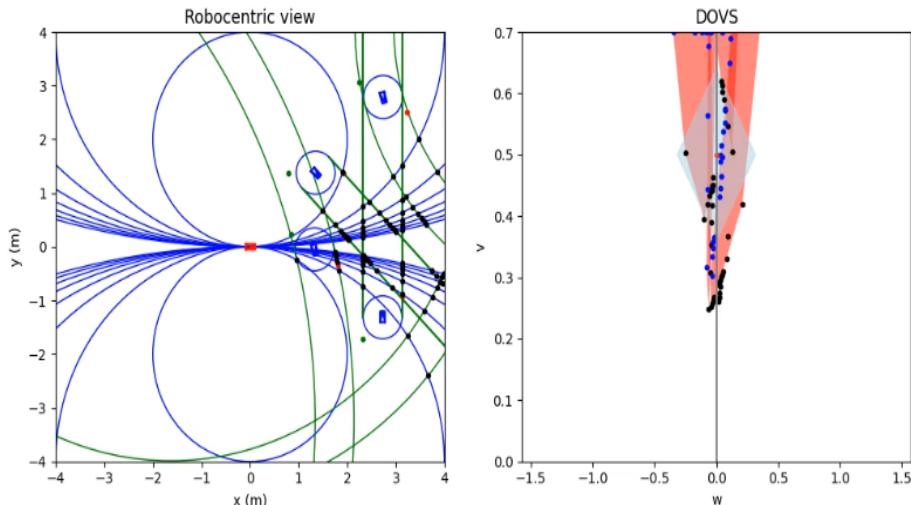


Figura 4.33: Experimento 8, DOVS del momento inicial

Estos problemas matemáticos se han resuelto poniendo mucho esfuerzo en estas partes para tratar de comprenderlas, y gracias a los profesores que me han ayudado.

### Puntos de colisión

La selección de los puntos de colisión correctos ha sido bastante problemática, ya que se tenían en cuenta muchos puntos que no se deberían tener en cuenta. Esto se debía a cómo se habían implementado las trayectorias y a que había que hacer un filtrado de los potenciales puntos de colisión que al principio no era evidente.

Para solucionar estas dificultades, se ha mejorado la implementación de las trayectorias y además conforme se iba probando el código con distintos entornos se iban perfilando los puntos de colisión correctos que tienen sentido considerar.

### Velocidades que llevan a colisión

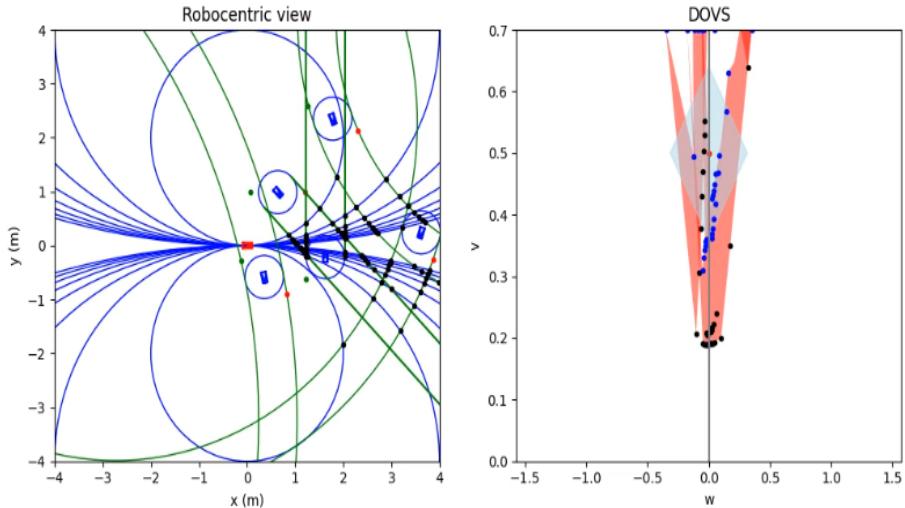


Figura 4.34: Experimento 8, momento 2

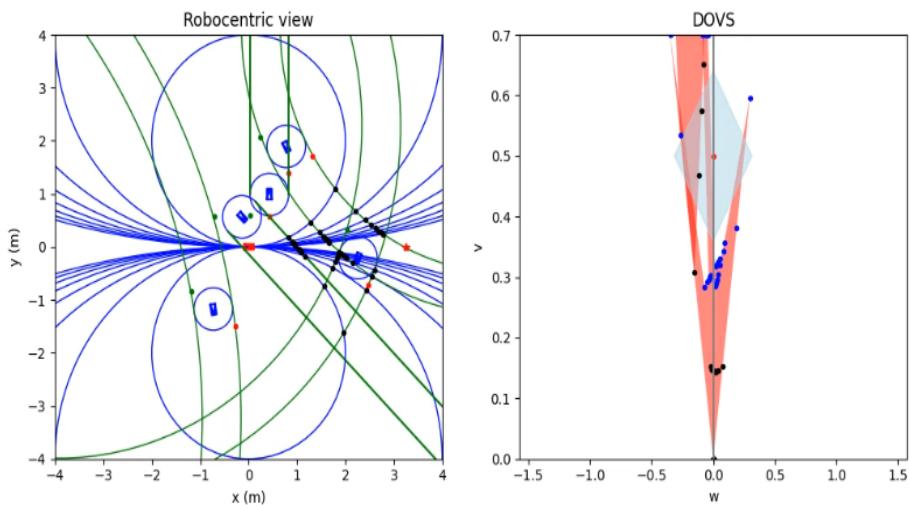


Figura 4.35: Experimento 8, momento 3

A la hora del cálculo de velocidades que llevan a colisión la mayor dificultad ha sido la de distinguir casos específicos en los que, dependiendo de los puntos de colisión calculados y la situación del obstáculo y del robot, la interpretación de los puntos variaba.

Estos son los casos específicos cuando el robot se encuentra dentro de la banda de colisión del obstáculo.

Para solucionar esto se ha tenido que probar exhaustivamente el código y al encontrar situaciones en las que las velocidades calculadas no tenían sentido se ha explorado junto con los profesores la razón de esto y cómo solucionarlo.

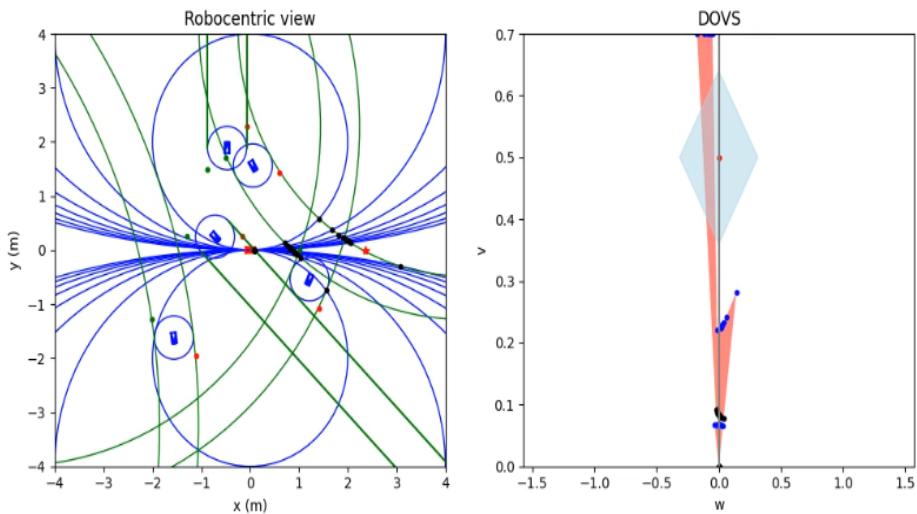


Figura 4.36: Experimento 8, momento 4

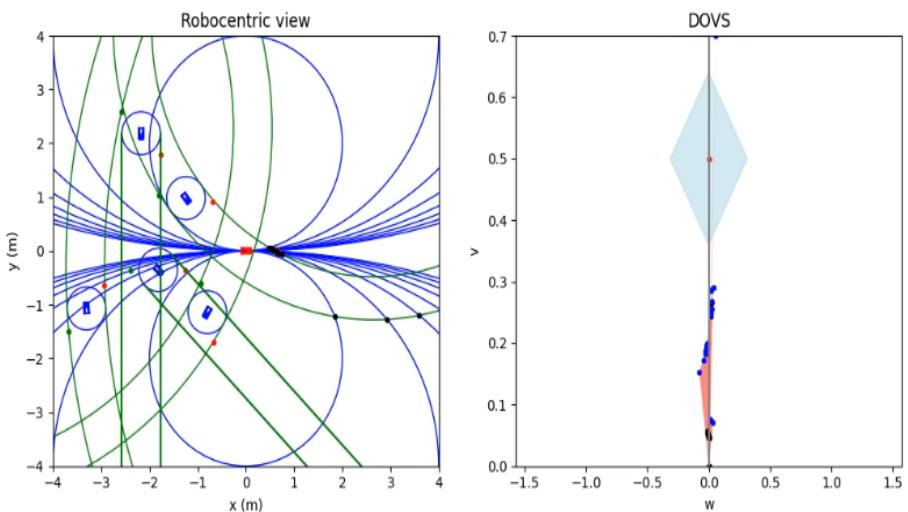


Figura 4.37: Experimento 8, llega a la meta sin colisionar

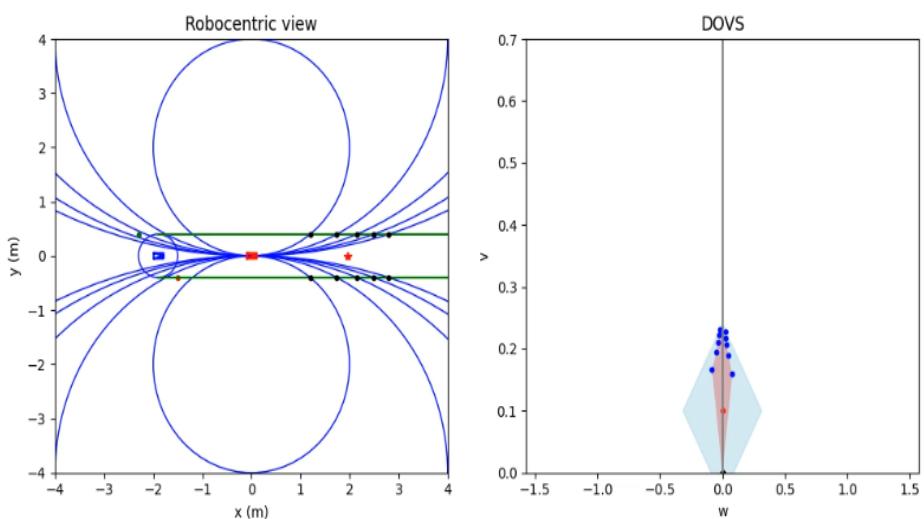


Figura 4.38: Experimento 9, momento inicial

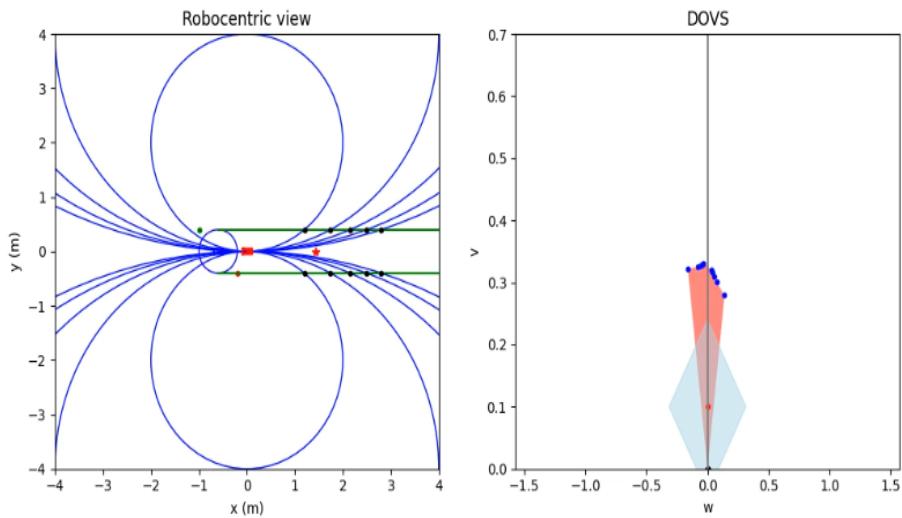


Figura 4.39: Experimento 9, a punto de colisionar

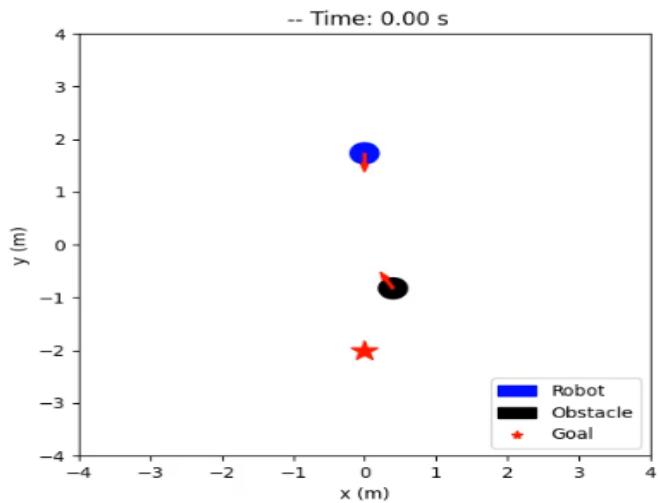


Figura 4.40: Experimento 10, entorno del momento inicial

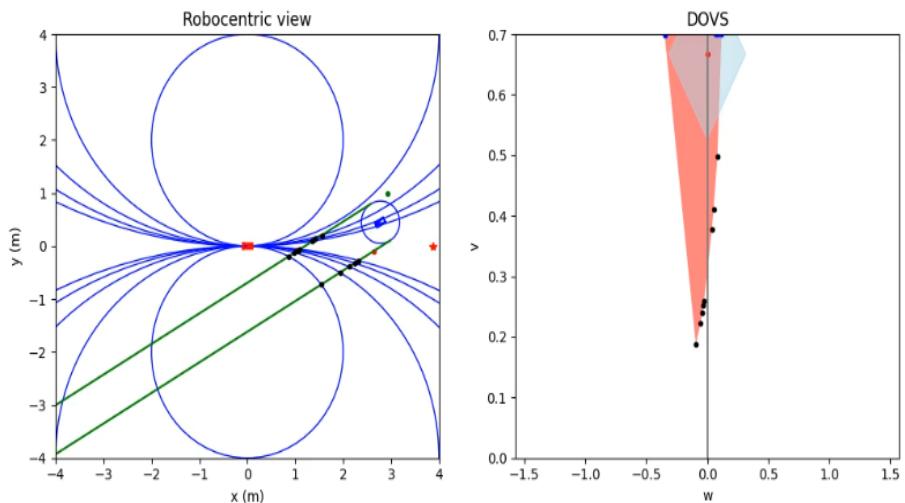


Figura 4.41: Experimento 10, DOVS del momento inicial

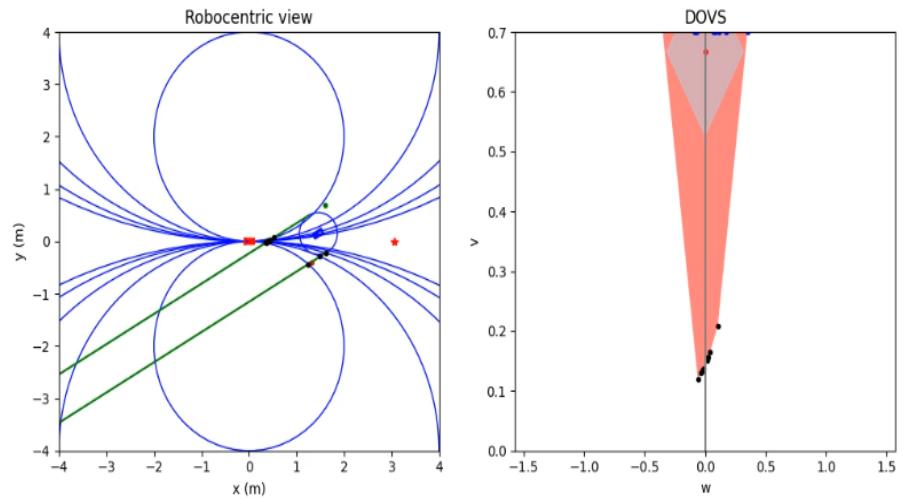


Figura 4.42: Experimento 10, el robot a avanzado con una velocidad que lleva a colisión, por lo que el DOV se ha hecho más grande

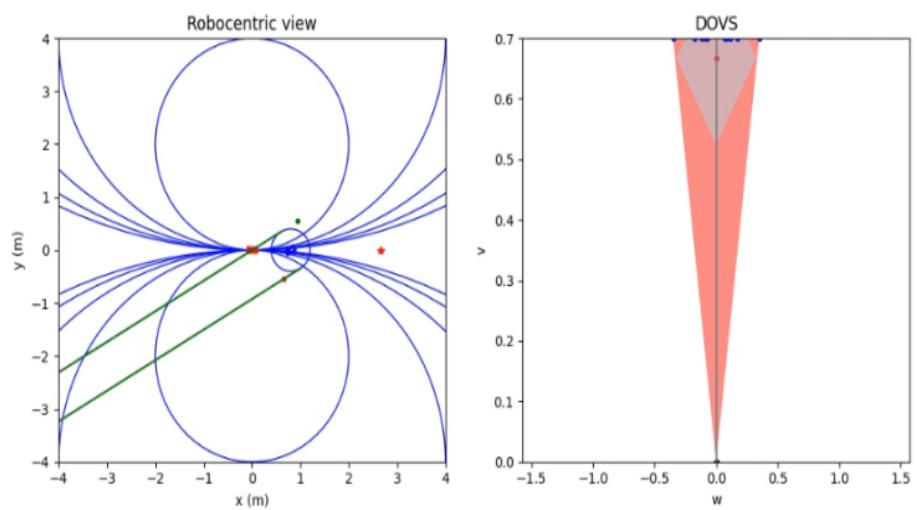


Figura 4.43: Experimento 10, momento justo antes de colisionar, el obstáculo está tan cerca que no se detectan colisiones, por ello se considera que todas las trayectorias llevan a colisión

# **Capítulo 5**

## **Conclusiones**

En esta sección se discuten las conclusiones extraídas de la librería desarrollada en este proyecto, así como de las futuras mejoras.

### **5.1. Conclusiones técnicas**

En resumen, se ha desarrollado una biblioteca en Python que cumple con éxito el objetivo de crear el Dynamic Object Velocity Space (DOVS) para la navegación en entornos dinámicos. Esta biblioteca ha demostrado su capacidad para modelar de manera efectiva y precisa el entorno dinámico, lo que la hace adecuada para ser utilizada por un planificador en la navegación de robots en entornos en constante cambio.

Un aspecto destacado de este proyecto es el diseño cuidadoso del código, que se ha enfocado en garantizar la comprensión y la extensibilidad. Se priorizó que el código fuera fácilmente entendible por aquellos que lo utilizarán, incluyendo a planificadores implementados por personas externas al proyecto. Esta característica es fundamental para adaptar el código a las necesidades específicas de cada planificador, y también para facilitar el mantenimiento y la futura expansión del sistema.

En conclusión, la biblioteca desarrollada ha logrado su objetivo de proporcionar un modelo DOVS eficiente y flexible para la navegación autónoma en entornos dinámicos. Su diseño pensado en la comprensión y la extensibilidad permite que sea utilizada de manera efectiva por planificadores externos, ofreciendo una solución versátil y adaptable para la navegación de robots en escenarios cambiantes.

### **5.2. Trabajo futuro**

El siguiente paso es la implementación de un planificador que mediante el modelo generado proponga una serie de comandos que lleven de manera segura el robot a su destino.

Centrándose en la modelación del entorno no se ha tenido en cuenta obstáculos estáticos. Esto es algo que se podría añadir en el futuro.

También podría centrarse en mejoras en el rendimiento, ya que cuando los entornos se complican con muchos obstáculos, el cálculo del DOVS no es muy rápido. Esto es algo a mejorar, revisando las partes críticas del código que más tiempo consumen e intentando mejorarlo. A pesar de eso esto es un problema menor ya que se eligió como lenguaje de programación Python, porque para el proyecto era muy importante la simpleza y legibilidad del código, algo que, en lenguajes compilados más rápidos, como C++ suele ser más difícil de llevar a cabo.

El haber implementado un código lo más simplificado y claro posible, permite el escalado del sistema incorporando una tercera dimensión, así como desarrollar un modelo avanzado con múltiples actores, robots y obstáculos que incorporen diferentes métodos de navegación, pudiendo reutilizarse la mayor parte del código, con las consiguientes ventajas derivadas de un modelo tridimensional, reflejo de la realidad.

# Capítulo 6

## Bibliografía

- [1] María Teresa Lorente, Eduardo Owen, and Luis Montano. Model-based robocentric planning and navigation for dynamic environments. *The International Journal of Robotics Research*, 37(8):867–889, 2018.
- [2] Diego Martinez, Luis Riazuelo, and Luis Montano. Deep reinforcement learning oriented for real world dynamic scenarios. *PNARUDE Workshop in 2022 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, 2022.
- [3] Diego Martínez, Luis Riazuelo, and Luis Montano. Full-stack s-dovs: Autonomous navigation in complete real-world dynamic scenarios. In *ROBOT2022: Fifth Iberian Robotics Conference: Advances in Robotics, Volume 2*, pages 14–25. Springer, 2022.
- [4] Andrew K Mackay, Luis Riazuelo, and Luis Montano. Rl-dovs: reinforcement learning for autonomous robot navigation in dynamic environments. *Sensors*, 22(10):3847, 2022.
- [5] Jorgemg117/dovs. GitHub repository. url: <https://github.com/JorgeMG117/DOVS>.
- [6] Matplotlib patches polygon. Matplotlib documentation, 2023. url: [https://matplotlib.org/stable/api/\\_as\\_gen/matplotlib.patches.Polygon.html](https://matplotlib.org/stable/api/_as_gen/matplotlib.patches.Polygon.html).
- [7] Sympy geometry. SymPy documentation, 2023. url: <https://docs.sympy.org/latest/modules/geometry/index.html>.
- [8] Shapely multipolygon. Shapely documentation, 2023. url: <https://shapely.readthedocs.io/en/latest/reference/shapely.MultiPolygon.html>.
- [9] Jorge Martinez. Ejemplos DOVS. YouTube playlist, 2023. url: <https://youtube.com/playlist?list=PLWgAuBjvj-dKgH7qs6X2vCW60ph4hPPak>.



# Listas de Figuras

2.1. Ejemplo DOVS. Las zonas rojas representan las velocidades que llevan a colisión y las blancas las libres/seguras. El punto rojo representa la velocidad actual y el rombo azul todas las velocidades alcanzables en un periodo de muestreo. . . . .	6
2.2. Ejemplo posibles trayectorias del robot en coordenadas robocéntricas . . . . .	8
2.3. Ejemplo ventana de velocidad del robot . . . . .	9
2.4. Ejemplo puntos colisión con obstáculo . . . . .	11
2.5. Robot con una sola trayectoria, banda de colisión del obstáculo, con los puntos representativos extremos de los obstáculos, para los que se calculan los puntos de colisión. El punto rojo del obstáculo representa el punto ${}^o x_{CPA}$ , calculado en la ecuación 2.11, el cual marca la posición en el obstáculo a partir de la cual se considera que el el robot pasa por delante del obstáculo. El punto verde del obstáculo representa el punto ${}^o x_{CPD}$ , calculado en la ecuación 2.12, el cual marca la posición en el obstáculo a partir del cual se considera que el obstáculo ha pasado antes de llegar el robot a él. . . . .	11
2.6. Ejemplo de trayectoria rectilínea de un obstáculo. Las rectas verdes delimitan la banda del plano barrida por el obstáculo, en la dirección de movimiento del mismo. . . . .	14
2.7. Ejemplo de trayectoria curvilínea de un obstáculo . . . . .	16
2.8. Ejemplo de DOVS. Las zonas rojas son las velocidades que llevan a colisión con los obstáculos. El resto de las velocidades son seguras. Los puntos azules corresponden a las velocidades mínimas necesarias para que el robot pase antes que llegue el obstáculo al punto de colisión, y las velocidades máximas que puede llevar el robot para que el obstáculo pase antes de que llegue el robot al punto de colisión. . . . .	17

2.9. Ejemplo de colisiones entre las trayectorias de un robot y la de un obstáculo, se ha limitado el número de trayectorias del robot por ser más claro en qué puntos de colisión se consideran. La linea izquierda de la trayectoria del obstáculo corta dos veces a la trayectoria del robot de menor radio, de entre los dos puntos se selecciona aquel que este más cerca del robot, ya que implica la situación más cercana en el tiempo de esa trayectoria del robot. . . . .	18
2.10. Ejemplo de situación con los puntos de colisión que se consideran. Puede observarse como los puntos de colisión cuyos ángulos en grados del arco sean mayor a 90º no son considerados. . . . .	21
2.11. Momento justo antes de colisionar, el obstáculo está tan cerca que no se detectan colisiones, por ello se considera que todas las trayectorias llevan a colisión . . . . .	23
2.12. Ejemplo de situación con los puntos de colisión que se consideran . . .	24
2.13. Ejemplo de situación con los puntos de colisión que se consideran . . .	24
 3.1. Diagrama de paquetes . . . . .	28
 4.1. Experimento 1, entorno del momento inicial . . . . .	34
4.2. Experimento 1, DOVS del momento inicial . . . . .	34
4.3. Experimento 1, cerca del robot . . . . .	35
4.4. Experimento 1, obstáculo ya ha pasado . . . . .	35
4.5. Experimento 2, entorno del momento inicial . . . . .	36
4.6. Experimento 2, DOVS del momento inicial . . . . .	37
4.7. Experimento 2, situación más avanzada . . . . .	37
4.8. Experimento 2, robot dentro de la banda de colisión . . . . .	38
4.9. Experimento 2, obstáculo ya ha pasado . . . . .	38
4.10. Experimento 3, DOVS del momento inicial . . . . .	39
4.11. Experimento 3, situación avanzada, va a llevar a colisión . . . . .	39
4.12. Experimento 3, momentos antes de la colisión . . . . .	40
4.13. Experimento 4, DOVS del momento inicial . . . . .	40
4.14. Experimento 4, situación más avanzada . . . . .	41
4.15. Experimento 4, un obstáculo ya ha pasado . . . . .	41
4.16. Experimento 4, momentos antes de la colisión . . . . .	42
4.17. Experimento 5, entorno inicial . . . . .	42
4.18. Experimento 5, momento inicial DOVS . . . . .	43
4.19. Experimento 5, situación intermedia, el robot ha pasado por delante de un obstáculo y se encuentra dentro de la banda de colisión de otro . . .	43

4.20. Experimento 5, instantes anteriores de llegar a la meta . . . . .	44
4.21. Experimento 6, DOVS del momento inicial . . . . .	44
4.22. Experimento 6, momentos antes de que el primer obstáculo pase por delante del robot . . . . .	45
4.23. Experimento 6, el primer obstáculo ha pasado por delante del robot . .	45
4.24. Experimento 6, básicamente ya han pasado los 3 obstáculos por delante del robot y ya no se detectan casi posibles puntos de colisión . . . . .	46
4.25. Experimento 7, entorno del momento inicial . . . . .	46
4.26. Experimento 7, DOVS del momento inicial . . . . .	47
4.27. Experimento 7, justo antes de pasar por detrás del primer obstáculo . .	47
4.28. Experimento 7, el robot ha pasado por detrás del primer obstáculo . .	48
4.29. Experimento 7, el robot está cerca de pasar por delante del segundo obstáculo . . . . .	48
4.30. Experimento 7, el robot se encuentra dentro de la banda de colisión del segundo obstáculo, por lo que esas velocidades que se calculan corresponden a las velocidades necesarias para pasar por delante de él .	49
4.31. Experimento 7, el robot ya ha evitado la colisión con los obstáculos y está muy cerca de llegar a la meta . . . . .	49
4.32. Experimento 8, entorno del momento inicial . . . . .	50
4.33. Experimento 8, DOVS del momento inicial . . . . .	50
4.34. Experimento 8, momento 2 . . . . .	51
4.35. Experimento 8, momento 3 . . . . .	51
4.36. Experimento 8, momento 4 . . . . .	52
4.37. Experimento 8, llega a la meta sin colisionar . . . . .	52
4.38. Experimento 9, momento inicial . . . . .	52
4.39. Experimento 9, a punto de colisionar . . . . .	53
4.40. Experimento 10, entorno del momento inicial . . . . .	53
4.41. Experimento 10, DOVS del momento inicial . . . . .	53
4.42. Experimento 10, el robot a avanzado con una velocidad que lleva a colisión, por lo que el DOV se ha hecho más grande . . . . .	54
4.43. Experimento 10, momento justo antes de colisionar, el obstáculo está tan cerca que no se detectan colisiones, por ello se considera que todas las trayectorias llevan a colisión . . . . .	54
A.1. Diagrama de Gantt . . . . .	68
C.1. Diagrama de clases . . . . .	76
C.2. Diagrama de secuencias, algoritmo DOVS . . . . .	78

C.3. Diagrama de secuencias, cálculo de los puntos de colisión . . . . .	79
C.4. Diagrama de flujo, cálculo de las velocidades prohibidas . . . . .	80

# **Lista de Tablas**

A.1. Horas invertidas . . . . .	67
---------------------------------	----



## **Anexos**



# Anexos A

## Gestion del trabajo

En cuanto a la distribución del trabajo realizado, este se comenzó a finales de Febrero y se acabo a principios de Junio. Se ha seguido un ritmo de trabajo estable, salvo en el último mes, que debido a que ya se habían acabado las clases, se le ha podido dedicar mas horas.

En el apartado A.1 se encuentran el total de horas dedicadas, y en el apartado A.2 el diagrama de Gantt con la distribución de los días.

### A.1. Horas invertidas

Tarea	Tiempos
Estudio del DOVS	38
Diseño del sistema	21
Implementacion funcion principal	5
Implementacion objetos del DOVS	19
Implementacion trayectorias de los objetos	22
Implementacion puntos de colision	31
Implementacion calculo de velocidades	39
Implementacion generacion del DOVS	18
Pruebas	64
Memoria	80
Reuniones	10
<b>Total</b>	<b>347</b>

Tabla A.1: Horas invertidas

### A.2. Diagrama de Gantt

## DOVS

Jorge Martinez Gil

Inicio de tfg

lu, 2/27/2023

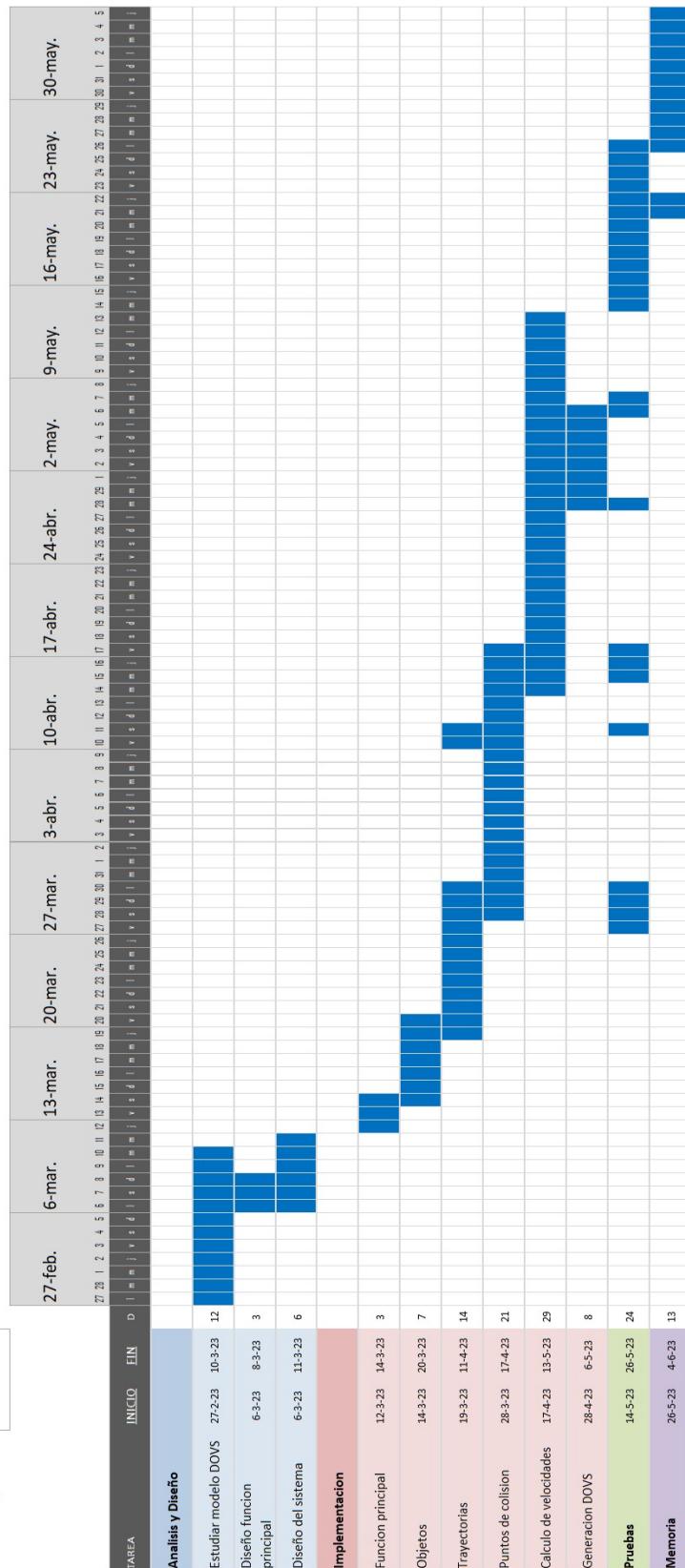


Figura A.1: Diagrama de Gantt

# Anexos B

## Código

### B.1. Calculo del DOVS

```
def compute_DOVS(self):
    """
    Compute the DOVS algorithm

    This function should be call every timestep
    """

    collision_points_list = []
    final_dovs = DOV()
    for obstacle in self.obstacles:

        # Check whether the robot is inside the collision band
        self.inside_col_band =
            → obstacle.inside_collision_band(self.robot.get_location())

        velocity_time_space = []

        for robot_trajectory in self.robot.trajectories:

            # Compute collision points of an obstacle for every robot
            → trajectory
            collision_points = self._compute_collision_points(robot_trajectory,
            → obstacle.trajectory, obstacle.get_location())
            collision_points_list.append(collision_points)

            velocity_time = self._collision_velocity_times(obstacle,
            → collision_points, robot_trajectory)
            if len(velocity_time) != 0:
                → velocity_time_space.append(velocity_time)

        dovs = DOV(velocity_time_space)
        final_dovs.combine_DOVS(dovs)

plotDOVS = PlotDOVS(self.robot, self.obstacles, self.fig_dovs, self.ax_dovs)
plotDOVS.plot_trajectories(collision_points_list)
plotDOVS.plot_DOVS(final_dovs)
```

```
    return self._choose_speed()
```

## B.2. Calculo de los puntos de colision

```
def _select_right_collision_point(self, collision_points, trajectory_radius,
→ obstacle_position):
    """
    Select the right collision point of the intersection of the robot trajectory
    and the obstacle trajectory

    Cojer de como mucho dos puntos de colision que estan por delante del
    → obstaculo el que esta mas cerca del robot
    """
    if collision_points:
        collision_point_1 = CollisionPoint(float(collision_points[0][0]),
→   float(collision_points[0][1]), trajectory_radius)

        collision1_from_obstacle =
→   ObjectDOVS.loc(np.dot(np.linalg.inv(ObjectDOVS.hom(obstacle_position)),ObjectDOVS.
→   collision_point_1.y, 0)))

        if len(collision_points) == 1:
            if collision1_from_obstacle[0] >= 0:
                return collision_point_1
            else:
                return None
        else:
            collision_point_2 = CollisionPoint(float(collision_points[1][0]),
→   float(collision_points[1][1]), trajectory_radius)

            collision2_from_obstacle =
→   ObjectDOVS.loc(np.dot(np.linalg.inv(ObjectDOVS.hom(obstacle_position)),ObjectDOVS.
→   collision_point_2.y, 0)))

            # Devolver el punto de colision mas cercano a la trayectoria del
            → robot
            if collision1_from_obstacle[0] >= 0 and collision2_from_obstacle[0]
            → >= 0:
                if collision_point_1.angle_arc <= collision_point_2.angle_arc:
                    return collision_point_1
                else:
                    return collision_point_2
            elif collision1_from_obstacle[0] >= 0:
                return collision_point_1
            elif collision2_from_obstacle[0] >= 0:
                return collision_point_2
            else:
                return None
        else:
            return None
    else:
        return None

def _valid_collision_points(self, collision_point_1, collision_point_2):

    if collision_point_1 != None and collision_point_2 != None and
    → collision_point_1.angle_arc <= 90 and collision_point_2.angle_arc <= 90:
        return collision_point_1, collision_point_2
    elif collision_point_1 != None and collision_point_1.angle_arc <= 90:
```

```

        return collision_point_1, None
    elif collision_point_2 != None and collision_point_2.angle_arc <= 90:
        return None, collision_point_2
    else:
        return None, None

def _compute_collision_points(self, trajectory, obstacle_trajectory,
→   obstacle_position):
    """
    Compute the collision points of the trajectory with the collision band

    :return: returns a tuple with two values, if there is an intersection the
→   value will be a Point2D, if not it will be None
    """

    intersection_1, intersection_2 = intersection(obstacle_trajectory,
→   trajectory)

    return
    →   self._valid_collision_points(self._select_right_collision_point(intersection_1,
    →   trajectory.radius, obstacle_position),
    →   self._select_right_collision_point(intersection_2, trajectory.radius,
    →   obstacle_position))

```

### B.3. Calculo de las velocidades

```

def _collision_velocity_times_aux(self, collision_point, obs_col, trajectory_radius,
→   v_obstacle, obs_trajectory):
    x_col = collision_point.x
    y_col = collision_point.y

    x_obs_col = obs_col[0]
    y_obs_col = obs_col[1]

    distance = obs_trajectory.distance_between_points(x_col, y_col, x_obs_col,
→   y_obs_col)

    t = distance/v_obstacle

    w = collision_point.angle/t
    v = trajectory_radius*w

    return (t, w, v)

def _collision_velocity_times(self, obstacle, collision_points, trajectory):
    """
    Devuelve tiempo y velocidades minimas y maximas que debe llevar le robot
    para no colisionar con el obstaculo
        Se llama a esta funcion con cada una de las trayectorias del robot y con
    cada uno de los obstaculos

    :param obstacle: obstaculo con el que se va a comprobar la colision
    :param collision_points: puntos de colision de la trayectoria del robot con
    el obstaculo, hay un maximo de dos colisiones
    """

```

```

:param trajectory: trayectoria circular del robot
:return ((t1, w_max, v_max), (t2, w_min, v_min)).
"""

obs_col_behind, obs_col_ahead = obstacle.get_colision_points()

v_object, _ = obstacle.get_speed()

w_min = v_min = w_max = v_max = t_min = t_max = 0

if collision_points[0] == None and collision_points[1] == None:
    if self.inside_col_band:
        # No puede escapar
        v_max = 0
        w_max = 0

        v_min = self.robot.max_v
        w_min = v_min/trajectory.radius
    else:
        return []
elif collision_points[0] == None or collision_points[1] == None:
    if collision_points[0] == None:
        collision_point = collision_points[1]
    else:
        collision_point = collision_points[0]

    if self.inside_col_band:
        # Calculamos la velocidad con la que podemos salir
        v_max = 0
        w_max = 0

        (t_min, w_min, v_min) =
            self._collision_velocity_times_aux(collision_point,
            obs_col_ahead, trajectory.radius, v_object, obstacle.trajectory)

    else:
        # Calculo la velocidad maxima que puedo llevar
        # Pongo como minima el limite superior. No hay velocidad de escape
        (t_max, w_max, v_max) =
            self._collision_velocity_times_aux(collision_point,
            obs_col_behind, trajectory.radius, v_object,
            obstacle.trajectory)

        v_min = self.robot.max_v
        w_min = v_min/trajectory.radius
else:
    idx_first = 0

    # Tenemos que ver cual de los dos puntos de colision es el que va a
    # definir la velocidad maxima

    if collision_points[0].arclength > collision_points[1].arclength:
        idx_first = idx_first + 1

    (t_max, w_max, v_max) =
        self._collision_velocity_times_aux(collision_points[idx_first],
        obs_col_behind, trajectory.radius, v_object, obstacle.trajectory)

```

```
(t_min, w_min, v_min) =  
    self._collision_velocity_times_aux(collision_points[(idx_first+1)%2],  
    obs_col_ahead, trajectory.radius, v_object, obstacle.trajectory)  
  
return [(t_max, w_max, v_max), (t_min, w_min, v_min)]
```



# Anexos C

## Diagramas

### C.1. Diagrama de clases

A continuación, se ha definido la estructura de las clases implementadas. Los objetivos que se han perseguido en este diseño son los siguientes:

- **Cohesión:** Cada clase debe tener una única responsabilidad y estar altamente enfocada en su propósito principal. Esto implica que los métodos y atributos de la clase deben estar relacionados entre sí y trabajar juntos para cumplir su función específica.
- **Encapsulación:** Las clases deben encapsular sus datos y comportamiento interno, exponiendo solo una interfaz bien definida para interactuar con ellos. Esto se logra utilizando atributos y métodos privados o protegidos, y proporcionando métodos públicos para acceder y modificar los datos de la clase de manera controlada.
- **Abstracción:** Las clases deben representar conceptos o entidades abstractas del dominio del problema, ocultando los detalles de implementación innecesarios. Esto facilita la comprensión y el uso de las clases, ya que los usuarios pueden centrarse en las características y comportamientos esenciales, sin tener que preocuparse por los detalles internos.
- **Herencia y reutilización:** Utilizar la herencia y la composición adecuadamente puede ayudar a crear jerarquías de clases coherentes y promover la reutilización de código. Las clases deben diseñarse de manera que permitan la extensión y la modificación sin alterar su funcionalidad básica, fomentando la reutilización de código y evitando la duplicación innecesaria.
- **Mantenibilidad y flexibilidad:** Un buen diseño de clases en Python debe ser fácil de mantener y permitir la flexibilidad para futuras mejoras o cambios en los

requisitos. Esto se puede lograr mediante la creación de clases bien estructuradas, con nombres descriptivos y coherentes, y utilizando principios como el principio de responsabilidad única y el principio de abierto/cerrado.

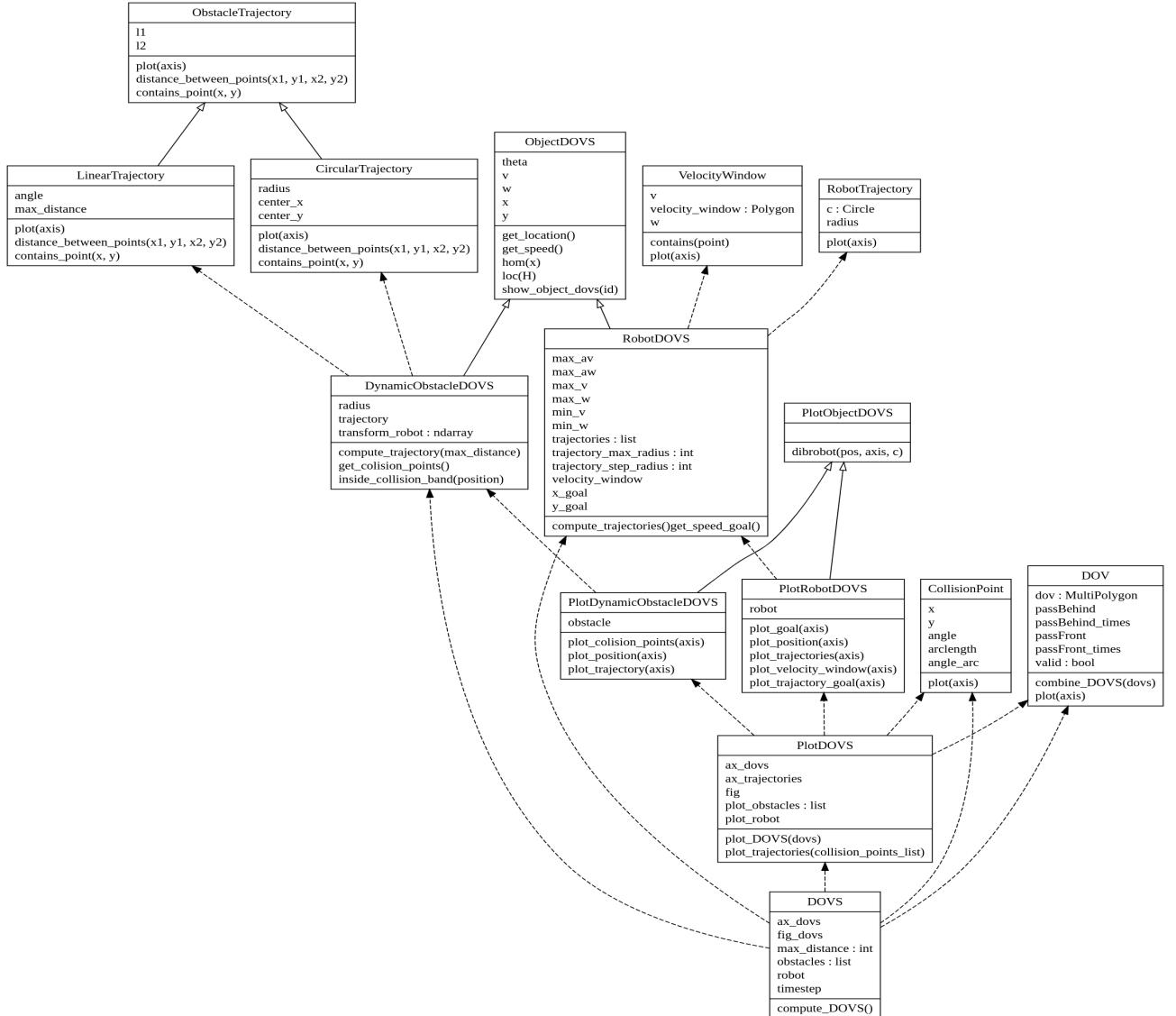


Figura C.1: Diagrama de clases

Siguiendo los objetivos comentados se ha construido el siguiente diagrama de clases C.1. A continuación, se describen las clases que posteriormente se han implementado.

- **DOVS**: Clase principal de la librería, contiene la función que crea y muestra el DOVS, creando las clases y llamando a los métodos correspondientes.
- **ObjectDOVS**: Contiene las funciones y atributos comunes a todos los objetos del DOVS

- **DynamicObstacleDOVS**: Contiene toda la lógica de los obstáculos que forman parte del entorno.
- **RobotDOVS**: Contiene toda la información necesaria para modelar el robot del DOVS.
- **PlotDOVS**: Clase encargada de mostrar los gráficos del DOVS
- **PlotObjectDOVS**: Clase que se encarga de mostrar toda la información relativa a todos los obstáculos del modelo.
- **PlotDynamicObstacleDOVS**: Muestra las gráficas relacionadas con los obstáculos del DOVS.
- **PlotRobotDOVS**: Clase que se encarga de mostrar toda la información relacionada con el robot.
- **ObstacleTrajectory**: Tiene la información común a todas las posibles trayectorias de un obstáculo.
- **LinearTrajectory**: Define la trayectoria lineal de un obstáculo.
- **CircularTrajectory**: Define la trayectoria circular de un obstáculo.
- **RobotTrajectory**: Contiene la información de la trayectoria del robot.
- **VelocityWindow**: Define la ventana de velocidad del robot
- **CollisionPoint**: Clase que contiene la definición de los puntos de colisión mediante los cuales se calculan posteriormente las velocidades.
- **DOV**: Clase que implementa el espacio de velocidades que llevan a la colisión del robot.

Se ha mantenido una abstracción y encapsulación de los objetos geométricos de manera que si se quiere cambiar la librería que utilizan por detrás para representar los objetos geométricos del modelo bastará con cambiar la clase que implementa el objeto.

Se ha hecho uso de la herencia para implementar aquellas clases de las cuales luego se podrían añadir más tipos, como es el caso de las posibles trayectorias de los obstáculos y los objetos que forman parte del DOVS. De manera que además se pueda reutilizar el código en común.

Se ha intentado hacer un uso de nombres para clases, métodos y atributos que sean entendibles de manera que el código sea más entendible por personas ajenas a la implementación.

En Python no existe una manera de crear métodos privados, pero existe la convención de añadir el carácter `_` antes de los métodos y atributos privados.

En resumen, todas las clases tienen una función específica, con una API que permite que las clases interactúen entre sí para conseguir el propósito que se busca, sin conocer la implementación interna de cada clase. Además, se ha usado la herencia para reducir código y sea más fácil expandir la funcionalidad del código.

## C.2. Diagrama de secuencias del cálculo del DOVS

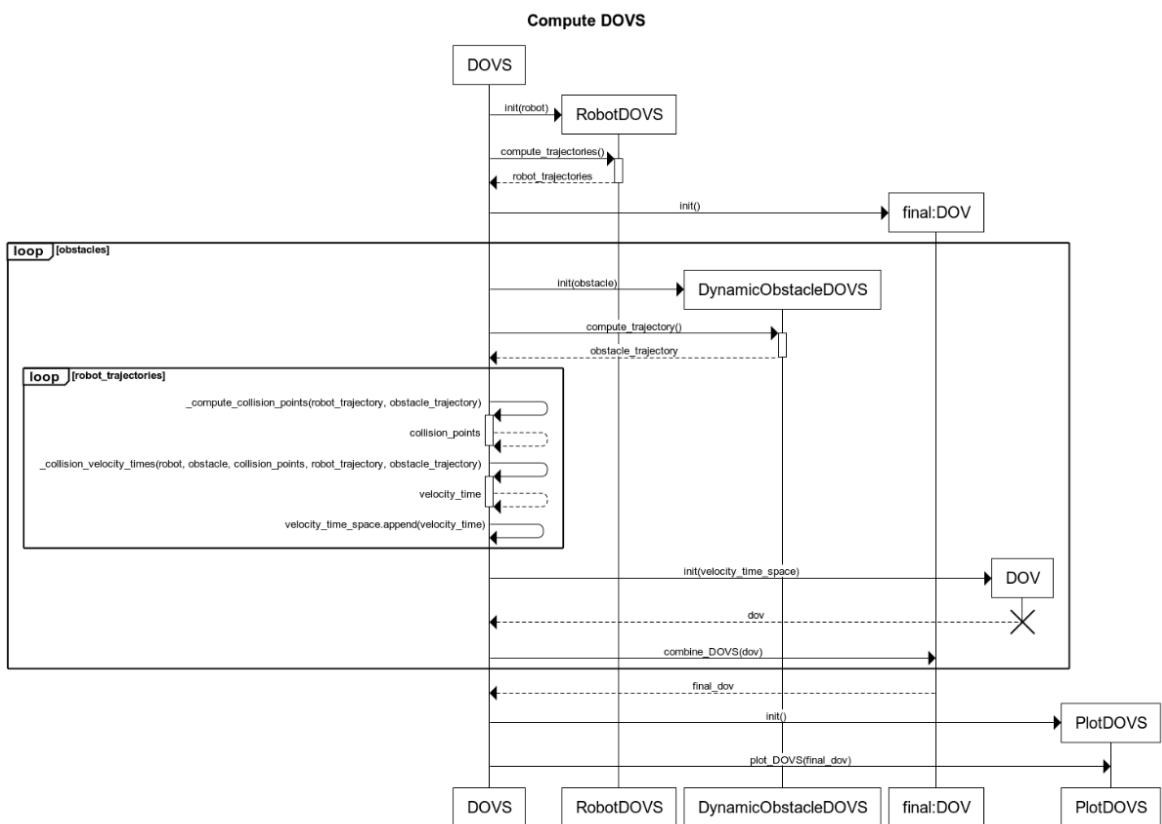


Figura C.2: Diagrama de secuencias, algoritmo DOVS

### C.3. Diagrama de secuencias del cálculo de los puntos de colisión

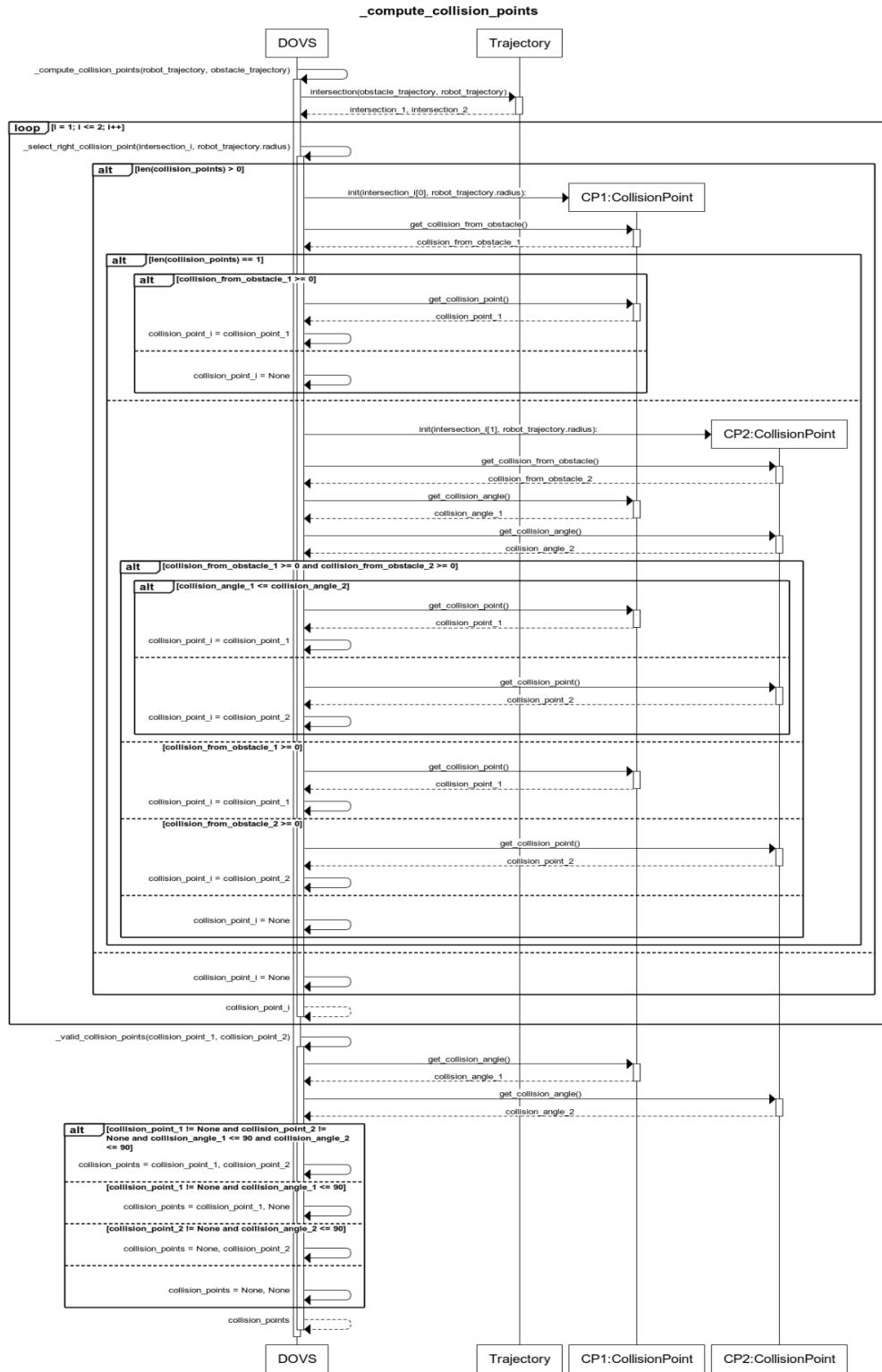


Figura C.3: Diagrama de secuencias, cálculo de los puntos de colisión

## C.4. Diagrama de flujo, cálculo de las velocidades de colisión

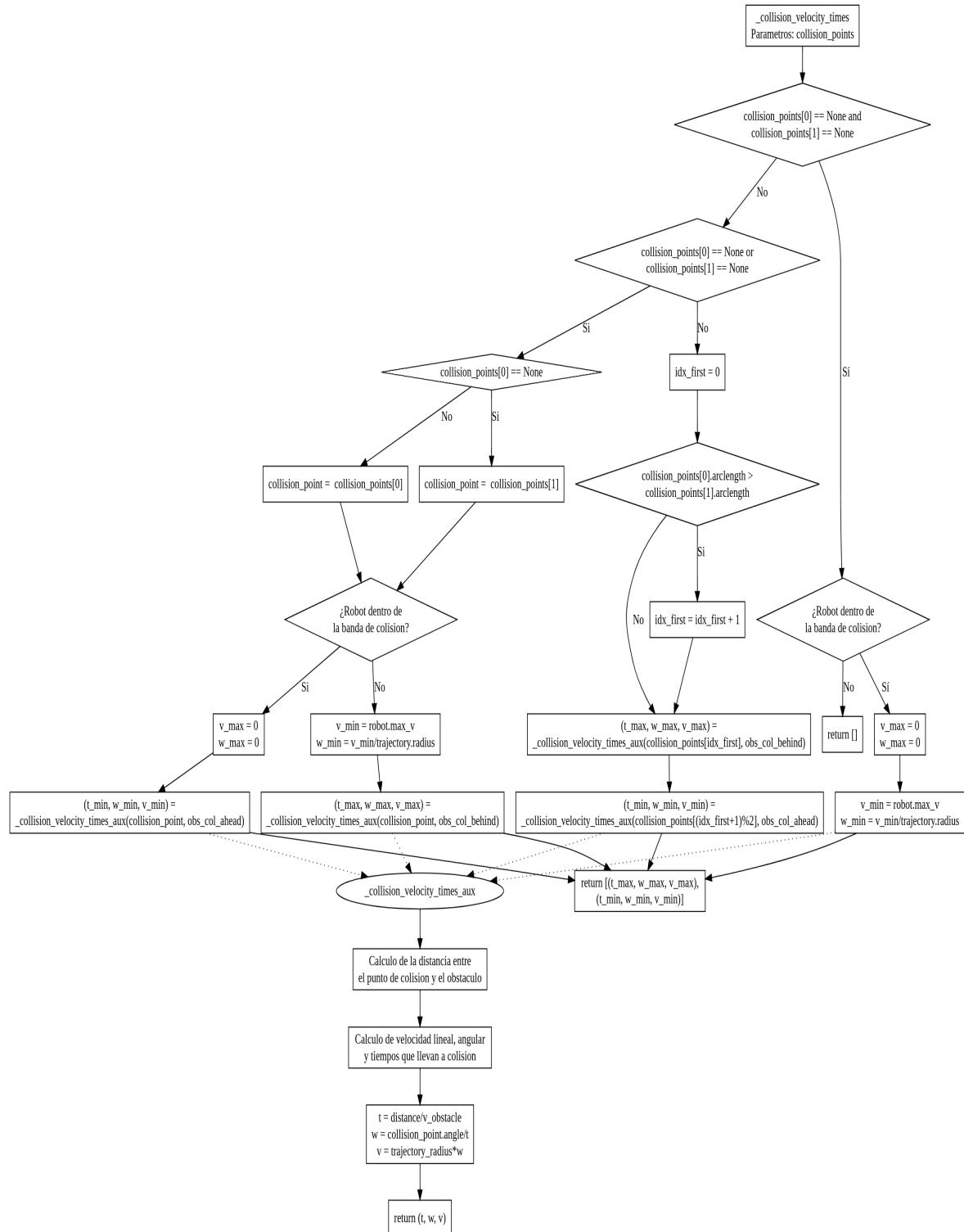


Figura C.4: Diagrama de flujo, cálculo de las velocidades prohibidas