Universidad Zaragoza
1542

Trabajo Fin de Máster

Implementación de un control probabilístico de sistemas multirobots para cumplir objetivos de alto nivel en la plataforma Robotarium

Implementation of probabilistic control of multi-robot systems to achieve high level objectives on the Robotarium platform

Autor/es

Pablo Fernández García

Director/es

Eduardo Montijano Muñoz
Cristian Florentín Mahulea

ESCUELA DE INGENIERÍA Y ARQUITECTURA

2024

Implementación de un control probabilístico de sistemas multirobots para cumplir objetivos de alto nivel en la plataforma Robotarium

RESUMEN

A lo largo del presente trabajo se ha buscado estudiar el control y planificación sobre sistemas multi-robots. El objetivo final ha consistido en el desarrollo de un algoritmo que controle múltiples equipos para la consecución de misiones complejas en entornos reales.

Para ello, se ha implementado un algoritmo de planificación de trayectorias capaz de trabajar en entornos dinámicos y cambiantes, que busca la consecución de unas tareas definidas por medio de ordenes a alto nivel, incluyendo definiciones temporales. Se ha partido de un algoritmo de planificación existente y se ha adaptado su funcionamiento y ampliado su características. Además, se ha introducido un control de colisiones para abordar el manejo de múltiples robots.

Para comprobar el funcionamiento correcto del sistema se ha implementado el algoritmo en una plataforma de robots reales permitiendo la simulación completa de los escenarios. Se han analizado las métricas correspondiente al movimiento de los robots para optimizar el comportamiento del sistema.

Se concluye que el algoritmo desarrollado puede de ser de gran utilidad al contar con una adaptación plena a entornos dinámicos y desconocidos, y al haber sido comprobado en un sistema real. Trabajos como este pueden ayudar al avance de la industria robótica y a las tecnologías que nos rodean.

Implementation of probabilistic control of multi-robot systems to achieve high level objectives on the Robotarium platform

ABSTRACT

Throughout this work, the control and trayectory planning of multi-robot systems has been studied. The final objective has been to develop an algorithm that controls multiple devices to perform complex missions in real environments.

To this end, a path planning algorithm capable of operating in dynamic and changing environments has been implemented. This algorithm seeks the completion of defined tasks through high-level commands, including temporal definitions.An existing planning algorithm has been taken as a starting point, extending its features. In addition, collision control has been introduced to deal with the handling of multiple robots.

To verify the correct operation of the system, the algorithm has been implemented on a real robot platform that allows the complete simulation of the scenarios. The movements of the robots have been analyzed to optimize the system's behavior.

It is concluded that the algorithm developed can be of great utility due to its full adaptation to dynamic and unknown environments, as well as its validation in a real system. Works like this can contribute to the advancement of the robotics industry and surrounding technologies.

Índice general

Índice de figuras

Capítulo 1

Introducción

1.1. Motivación

En los últimos años se está presenciando un auge en el desarrollo de sistemas robóticos en diferentes actividades a nivel industrial, como por ejemplo el mundo de la logística. El uso de estos sistemas puede ser una herramienta muy útil para abordar problemas de la organización de la mercancía y su traslado de material, y permitir a los usuarios automatizar completamente la tarea consiguiendo una eficiencia máxima que marque la diferencia.

Dentro de los sistemas de automatización destacan los equipos multi-robot, que otorgan una gran versatilidad a la hora de resolver varias tareas de manera simultánea. El campo de aplicaciones comprende diversas industrias y entornos, que van desde la agricultura o la ganadería de precisión, la conducción autónoma, hasta el uso en la industria logística. Las tareas que abordan este tipo de tecnologías suelen ser difíciles de explicar y, sobre todo, de resolver, ya que al coexistir múltiples robots es necesario calcular muchas variables y solucionar problemas de escalabilidad y de comunicación. Esto exige el uso de métodos que optimicen el rendimiento de los sistemas. En la figura 1.1 se pueden observar diferentes sistemas robóticos aplicados en casos reales. Para que estos equipos funcionen de la manera esperada es necesario implementar algoritmos que sean capaces de tomar decisiones de manera robusta y escalable, teniendo en cuenta tanto las especificaciones de alto nivel como las particularidades físicas de cada robot.

Uno de los problemas principales con los que deben lidiar estos equipos es la necesidad de adaptarse a entornos dinámicos, donde existe un continuo flujo de información debido a actualizaciones del escenario. El constante cambio de los entornos de trabajo se ha convertido en un desafío técnico significativo, ya que provoca que los algoritmos deban ser eficientes y ágiles para sobrellevar la variabilidad de situaciones.

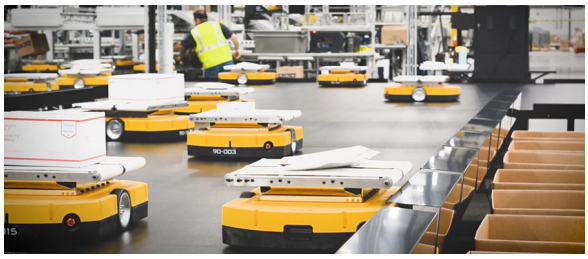


Figura 1.1: Ejemplos de aplicaciones reales de sistemas robóticos en la agricultura [1] y en la organización logística [2]

Otros de los desafíos con los que cuenta este tipo de tecnologías es la capacidad de codificar órdenes complejas de alto nivel donde no solo se tenga en cuenta la meta o el destino, sino que también entre en juego variables como el tiempo y el orden de ejecución. Además, si las tareas aumentan en complejidad, será necesario asegurar la capacidad de coordinación de los robots para abordar los objetivos de la manera más eficiente posible.

Por último, en sistemas que cuentan con más de un robot, se puede presentar situaciones donde se provoquen bloqueos o colisiones entre ellos. Es necesario que este tipo de instalaciones cuente con sistemas que sean capaces de supervisar posibles colisiones, con el objeto de predecir estas situaciones y realizar trayectorias alternativas que las eviten.

Todos estos puntos anteriormente citados motivan la realización del presente trabajo, donde se busca desarrollar una herramienta para el control de escenarios multi-robot comandados por especificaciones de alto nivel, manejando las diferentes situaciones y problemáticas que se pueden dar. Además, el presente estudio esta enmarcado dentro de un proyecto de investigación, denominado "Mejoras en comprensión automática de escenas mediante modalidades múltiples de sensores y percepción activa"(iSUMA), financiado por la Agencia Estatal de Investigación (código PID2021-125514NB-I00).

1.2. Objetivos

El objetivo principal del presente trabajo ha consistido en el desarrollo de un sistema multi-robot capaz de solucionar problemas complejos descritos mediante especificaciones de alto nivel. Para lograr satisfacer el objetivo principal surgen dos trabajos diferentes.

Por un lado, se ha centrado el foco en el desarrollo del algoritmo de planificación bajo especificaciones de alto nivel. En esta primera sección, se busca la obtención de una trayectoria para cada robot que permita cumplir la misión impuesta. El objetivo final radica en el desarrollo de un algoritmo de planificación que sea capaz de trabajar en entornos más realistas teniendo en cuenta todas las restricciones que se pueden dar.

Por otro lado, para poder verificar el comportamiento del sistema se va a hacer uso de la plataforma Robotarium, altamente utilizada para la experimentación con robots reales. El objetivo principal consiste en el desarrollo de un control de bajo nivel que sea capaz de transformar las órdenes de alto nivel en comandos para que los robots realicen las tareas específicas. Esta plataforma cuenta con una metodología propia que permite adaptar las simulaciones al área de pruebas.

1.3. Alcance

El alcance del trabajo realizado comprende dos grandes puntos de trabajo y una serie de tareas internas a realizar para lograr el funcionamiento completo del sistema.

En primer lugar, es necesario trabajar en la planificación a alto nivel de la plataforma. Se ha tomado como punto de partida una implementación ya realizada en el pasado [3] que cuenta con una serie de limitaciones a la cual se le han introducido una serie de bloques:

- Extensión del problema para manejar escenarios dinámicos.
- Gestión de colisiones para evitar que se produzcan situaciones de bloqueo del sistema.
- Definición de misiones más complejas teniendo en cuenta el tiempo como una variable más para ser capaces de definir sucesiones de acciones.

En segundo lugar, con el fin de experimentar en la plataforma real, se requiere el desarrollo de la herramienta de control de bajo nivel, la cual sea capaz de transformar las órdenes resultantes del algoritmo de planificación en consignas para los elementos del equipo. Para la consecución de este objetivo, se han definido dos tareas diferentes:

- Generación de trayectorias a partir de la planificación de alto nivel e implementación de las mismas en la plataforma real.
- Implementación de un sistema de visualización para comprensión del estado de la plataforma por parte del usuario.

Capítulo 2

Estado inicial

Como punto de partida para el desarrollo del presente trabajo, se ha realizado un análisis previo de los dos dominios principales que componen el sistema propuesto. En la fase inicial se ha estudiado un algoritmo de planificación existente que se ha tomado como base para su posterior desarrollo, mientras que para la rama que abarca el bajo nivel se ha estudiado la plataforma escogida para realizar simulaciones reales.

2.1. Planificación de alto nivel

En el ámbito robótico que se está trabajando, es fundamental y necesario la existencia de planificadores que permitan comandar los equipos. Estos planificadores de alto nivel presentan la función principal de proporcionar un conjunto de trayectorias diseñadas para cumplir con diferentes requisitos y restricciones impuestas. Estos planificadores actúan como guías estratégicos, que son capaces de captar la información del ambiente, procesarla de una manera eficiente y devolver unos resultados que satisfagan las necesidades.

Como punto de inicio para el posterior desarrollo se ha usado una implementación de un planificador para multi-robots que opera bajo especificaciones de alto nivel [3]. Estas especificaciones consisten en fórmulas booleanas en las cuales se estipula unas condiciones a cumplir basado en unas zonas objetivos finales. Uno de los aspectos innovadores con los que cuenta este artículo radica en la implementación de escenarios con incertidumbre, donde los robots no son plenamente conocedores de todo su entorno, si no que se basan en probabilidades. Toda esta planificación acaba devolviendo unas trayectorias para la consecución de las especificaciones impuestas, y es modelado por medio de las Redes de Petri.

2.1.1. Redes de Petri

Las Redes de Petri son una herramienta matemática proveniente de la teoría de grafos que permite la representación de sistemas concurrentes por medio de nodos y arcos de conexión [4]. El rango de aplicación de esta metodología es muy amplia gracias a la gran versatilidad que ofrece para representar diferentes sistemas.

La red esta conformado por nodos y arcos de conexión. Los nodos están interconectados por medio de arcos, que pueden llevar asignados diferentes pesos. Estos arcos únicamente pueden interconectar nodos de distinto tipo, nunca podrán tener en sus extremos nodos de la misma clase. Las dos tipologías de nodos son:

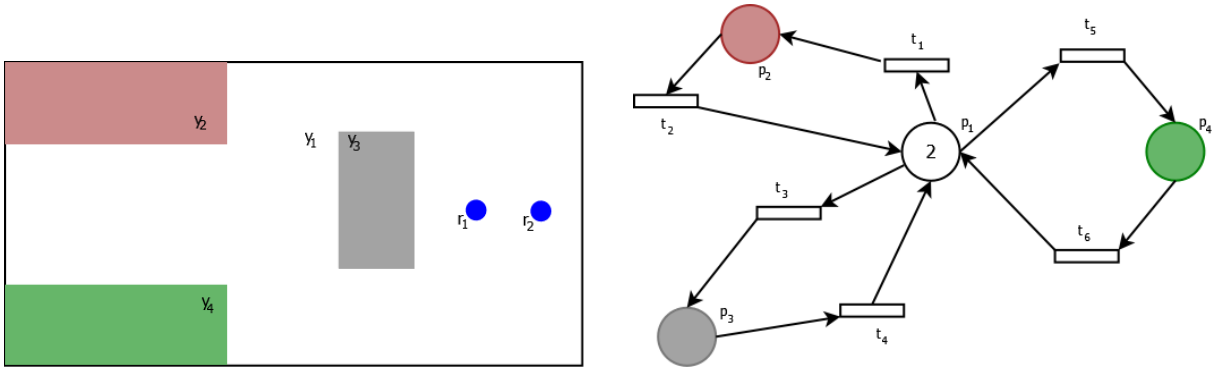
- ***Lugares:*** estos nodos suelen ser representados por medio de círculos y se corresponden con variables de estado. Los lugares pueden tener un número de tokens (marcas) que representan el valor de la variable de estado, i.e., un estado local. Estos tokens aumentan o disminuyen en función de las transiciones disparadas.
- ***Transiciones:*** estos nodos suelen ser representados por medio de un rectángulo y modelan eventos. Para poder disparar una transición, se necesita que todos los lugares de entrada (lugares que se conectan con una transición por medio de un arco) tengan el marcado necesario acorde al peso de los arcos. Tras la ejecución de la transición los lugares entrantes perderán el marcado correspondiente, y los lugares salientes aumentarán su número de tokens. El disparo de una transición es una política de consumo - producción sin que haya ninguna ley de conservación de marcas, i.e., el número de marcas consumidas no tiene que ser igual al número de marcas producidas.

Para la aplicación de este sistema a escenarios robóticos se realiza una extensión de las redes de Petri denominada *Red de Petri de movimiento de robots* (*Robot Motion Petri Net, RMPN*) (definición 1).

Definición 1. *[5] [3] Una* Red de Petri de movimiento de robots *es una tupla* $\mathcal{N} = \langle P, T, \boldsymbol{Pre}, \boldsymbol{Post}, Y, h, \boldsymbol{M_0} \rangle$*, donde:*

- $P = \{p_1, p_2, \ldots, p_m\}$ *es un conjunto finito de lugares, cada lugar* $p_i \in P$ *modela una región del entorno;*
- $T = \{t_1, t_2, \ldots, t_n\}$ *es un conjunto finito de transiciones. Si dos regiones* p_i *y* p_j *son adyacentes, existirán dos transiciones* $t_{i,j}$ *y* $t_{j,i}$ *modelando el desplazamiento en ambos sentidos.*
- $\boldsymbol{Pre} \in \mathbb{N}^{|P|\times|T|}$ *es la matriz de preincidencia. Contiene los pesos de las arcos que conectan lugares con transiciones. Si existe un arco de* p_i *a* t_j *con peso unitario, se corresponderá con* $\boldsymbol{Pre}[p_i, t_j] = 1$

- $\boldsymbol{Post} \in \mathbb{N}^{|P|\times|T|}$ *es la matriz de postincidencia. Contiene los pesos de las arcos que conectan transiciones con lugares. Si existe un arco de* t_j *a* p_i *con peso unitario, se corresponderá con* $\boldsymbol{Post}[p_i, t_j] = 1$
- $\mathcal{Y} = \{y_1, y_2, \ldots, y_{|M|}\}$ *es un conjunto finito de etiquetas. Cada etiqueta almacena información sobre las regiones. A cada región (lugar) se le puede asignar una o varias etiquetas.*
- $h : P \to \mathcal{Y}$ *es la función de etiquetación donde* $h(p_i)$ *es el conjunto de etiquetas asignadas al lugar* $p_i \in P$*.*
- $\boldsymbol{M}_0 \in \mathbb{N}^{|P|}$ *es el marcado inicial y se corresponde con la ubicación inicial de cada robot en las regiones analizadas.*



(a) Representación escenario ejemplo (b) Red de Petri asociada al escenario anterior

Figura 2.1: Representación de un escenario ejemplo por medio de Redes de Petri

En la figura 2.1 se puede apreciar el modelado de un escenario con una Red de Petri. La Red de Petri generada consiste en un en un conjunto de lugares $P = \{p_1, p_2, p_3, p_4\}$ y otro conjunto de transiciones $T = \{t_1, t_2, t_3, t_4, t_5, t_6\}$. El marcado inicial es $\mathbf{M}_0 = \begin{bmatrix}2 & 0 & 0 & 0\end{bmatrix}^T$ y la función de salida (de asignación de etiquetas a cada lugar) es $h(p_i) = y_i, \forall p_i \in P$ con el conjunto dfe etiquetas definido como $\mathcal{Y} = \{y_1, y_2, y_3, y_4\}$. Por otro lado, las matrices **Pre** y **Post** son,

$$\boldsymbol{Pre} = \begin{pmatrix} 1 & 0 & 1 & 0 & 1 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{pmatrix} \tag{2.1}$$

$$\boldsymbol{Post} = \begin{pmatrix} 0 & 1 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \end{pmatrix} \tag{2.2}$$

La matriz de incidencia se define como $\boldsymbol{C} = \boldsymbol{Post} - \boldsymbol{Pre}$. El estado en una red de Petri es distribuido y numérico, y se puede definir una ecuación de estado llamada ecuación

fundamental de la red de Petri (ecuación 2.3), donde $\boldsymbol{\sigma}$ se corresponde con el vector que almacena en cuántas ocasiones se ha disparado cada transición y $\boldsymbol{M}$ es el marcado que se obtiene al disparar una secuencia de transiciones correspondiente a $\boldsymbol{\sigma}$.

$$\boldsymbol{M} = \boldsymbol{M}_0 + \boldsymbol{C} \cdot \boldsymbol{\sigma} \tag{2.3}$$

2.1.2. Algoritmo de planificación

A lo largo de la presente sección se define el algoritmo de planificación del artículo de referencia [3]. Este algoritmo se basa en un entorno probabilístico, donde la ubicación de los diferentes componentes del entorno tiene una probabilidad de serlo, lo que convierte el estado de las celdas a no ser un estado binario (es o no es), si no que cada celda tendrá una probabilidad de corresponderse con una zona. Este hecho presenta ventajas ya que permite realizar una asociación mucho más sencilla con herramientas de visión computacional, donde la detección de distintos objetos utilizando inteligencia artificial siempre va asociada una probabilidad de acierto. De esta manera se podría realizar una asociación directa de la visión con la planificación en los robots.

El algoritmo de partida consiste en la resolución de un problema de optimización basado en modelos de redes de Petri. Usar este tipo de descripción de los escenarios permite la generación de modelos más escalables ya que no necesitan generar todos los estados posibles del sistema, y utilizan técnicas de diseño que exploran todo el ámbito posible de una manera eficiente.

Se busca minimizar la función de coste

$$\min_{\boldsymbol{M},\tilde{\boldsymbol{\sigma}},b} \quad C(\tilde{\boldsymbol{x}}) = J_\sigma + J_{\mathbb{P}} + J_{obs} + J_{col} \tag{2.4}$$

teniendo en cuenta las restricciones

$$\begin{cases} \boldsymbol{A}_{eq} \cdot \tilde{\boldsymbol{x}} = \boldsymbol{b}_{eq} \\ \boldsymbol{A} \cdot \tilde{\boldsymbol{x}} \leq \boldsymbol{b} \end{cases}, \quad \tilde{\boldsymbol{x}} = \begin{bmatrix} \tilde{\boldsymbol{\sigma}} \\ \boldsymbol{m} \\ \boldsymbol{x} \\ b \end{bmatrix}. \tag{2.5}$$

Las matrices $\boldsymbol{A}_{eq}$ y $\boldsymbol{A}$ y los vectores $\boldsymbol{b}_{eq}$ y $\boldsymbol{b}$ corresponden a las 6 restricciones para asegurarse que el sistema simulado se corresponde con el escenario diseñado. La restricción

$$\boldsymbol{M} = \boldsymbol{M}_k + \boldsymbol{C} \cdot \boldsymbol{\sigma} \tag{2.6}$$

es la ecuacion de estado y emula la evolución del marcado de la red de Petri; la ecuación

$$\boldsymbol{\eta}^T \cdot \tilde{\boldsymbol{\sigma}} = \mathbf{0} \tag{2.7}$$

evita que se alcancen lugares con probabilidades altas de ser obstáculos; la restricción

$$\boldsymbol{A}_{task} \cdot \boldsymbol{x} = \boldsymbol{b}_{task} \tag{2.8}$$

asegura que se cumpla la condición booleana impuesta; las ecuaciones

$$\begin{cases} -\boldsymbol{W} \cdot \boldsymbol{M} + \tau_{\mathbb{P}} \cdot \mathbf{1}_M \leq N \cdot \boldsymbol{x} & \text{(2.9a)} \\ \boldsymbol{W} \cdot \boldsymbol{M} - \leq N \cdot (\mathbf{1}_M - \boldsymbol{x}) & \text{(2.9b)} \end{cases}$$

busca colocar robots en aquellas zonas donde se maximice la probabilidad de cumplir la condición booleana; y por último, la condición

$$\boldsymbol{Post} \cdot \tilde{\boldsymbol{\sigma}} \leq b \cdot \mathbf{1}_P \tag{2.10}$$

junto con el termino J_{col} de la función de coste minimiza la probabilidad de que los robots pasen por una celda más de una vez.

Las incógnitas del problema de optimización se listan a continuación:

- $\boldsymbol{M}$ - marcado final del sistema.
- $\tilde{\boldsymbol{\sigma}}$ - vector conteniendo número de veces que se dispara cada transición.
- $\boldsymbol{x}$ - estado final de cada una de las zonas definidas (etiquetas).
- b - número máximo de veces que los robots pasan por una misma celda.

En lo que refiere a la función del coste, se introducen una serie de pesos que buscan optimizar la solución encontrada, para siempre decantarse por la mejor solución posible. Para ello, se definen 4 pesos distintos:

- J_σ: peso que que minimiza el número de movimientos realizados por la plataforma
- $J_{\mathbb{P}}$: elemento que busca maximizar la probabilidad de éxito del escenario resuelto, en base a las probabilidades del sistema
- J_{obs}: peso que busca minimizar la probabilidad de moverse por lugares con probabilidad de ser obstáculos
- J_{col}: elemento que busca reducir el número de robots que pasa por una celda para minimizar el riesgo de colisión

Con esta definición quedaría establecido el punto de partida del algoritmo de planificación para sus posteriores modificaciones.

2.2. Plataforma robotarium

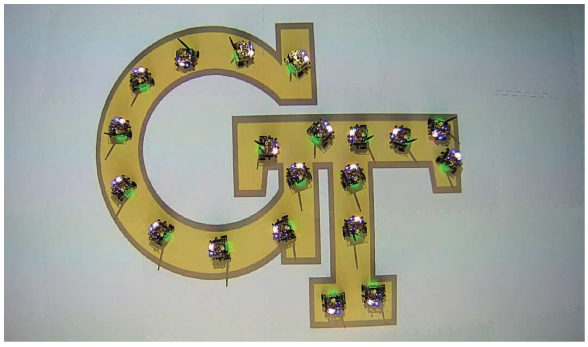
El algoritmo de planificación que se quiere desarrollar es independiente del escenario real que se esta calculando, lo cual le otorga la capacidad de poder aplicarse en multitud de ámbitos distintos. En el caso que nos atañe, se ha escogido la plataforma denominada *Robotarium*, desarrollada por el Instituto de Tecnología de Georgia (*GeogiaTech*) [6]. Esta plataforma permite subir algoritmos de ejecución a su servidor y realizar una simulación real en remoto con los robots, remitiendo todos los archivos de salida junto a un vídeo resultante de la ejecución de la simulación.

La plataforma consiste en una superficie de 3,66x4,26m [6] donde los robots se pueden desplazar acorde a las instrucciones dadas. El número máximo de robots que se puede utilizar es de 20 unidades. Cada chip cuenta con un placa principal ESP8266 que lleva incorporado un transmisor WIFI. Para poder asegurar la autonomía durante la simulación cada robot cuenta con una batería LiPo de 400 mAh, que le otorga una duración de carga de hasta 40 minutos. Se pueden apreciar imágenes de la plataforma en la figura 2.2.

El control implementado por defecto en los robots es un control de velocidad. La librería de la plataforma proporciona un control básico para calcular automáticamente las consignas de velocidad a partir de la posición actual del robot y el punto de destino.

Además, la plataforma cuenta con un sistema de geolocalización global por medio de la Webcam, que se utiliza para informar al usuario constantemente de la posición de los robots. Por último, para cargar los robots entre simulaciones, los robots llevan incorporado un sistema de carga inalámbrico para poder cargarlos automáticamente sin necesidad de actuación humana.

(a) Plataforma de simulación Robotarium.[7]



(b) Ejemplo de simulación con proyección.[7]

Figura 2.2: Imágenes reales de la Plataforma Robotarium

Respecto al software, la plataforma se divide en tres programas diferentes que aseguran el éxito de las pruebas:

- **Simulación:** La plataforma otorga una librería que simula la plataforma para que

se pueda testear el código antes de subirla. Además, antes de iniciar cualquier simulación se realiza un testeo del código en el servidor y una comprobación de no colisiones.

- **Interacción:** la interacción con los robot se realiza por medio de ordenes transmitidas a partir de las librerías específicas de la plataforma. Estas están disponibles para los lenguajes de *Matlab* y *Python*.
- **Coordinación:** el servidor es la aplicación central de la plataforma que se encarga de ejecutar el código del usuario, mandar órdenes y recibir datos de los robots.

Por último, el sistema cuenta con un proyector en la parte superior que permite el renderizado de imágenes o dibujos sobre la plataforma. Esta herramienta dota a la plataforma de una capacidad de visualización y representación del entorno de simulación para aumentar el realismo y la calidad de el experimento.

La interacción entre el usuario y la plataforma se realiza por medio de la web `https://www.robotarium.gatech.edu/`.

Capítulo 3

Algoritmia

En cualquier solución moderna centrada en sistemas multi-robots la parte más importante del sistema es el algoritmo de planificación. Realizar una correcta organización de las tareas controlando el estado del entorno en todo momento es imprescindible para lograr un desarrollo de los objetivos con eficiencia. Para lograr este objetivo, a lo largo del presente capítulo se desarrollan diferentes especificaciones para conseguir un algoritmo optimizado y fiable capaz de trabajar en gran diversidad de entornos.

3.1. Problema dinámico

Uno de los retos a los cuales se someten los sistemas robóticos de la actualidad es la necesidad de enfrentarse ante escenarios en constante cambio. Este hecho hace que sea necesario que cualquier sistema robótico tenga que contar con capacidad de manejar sistemas variables. Los retos que presentan esta implementación son la introducción de toma de decisión en tiempo real y la detección de todas las zonas de interés para posteriormente realizar la replanificación.

El algoritmo de partida [3] está preparado para trabajar con sistemas estáticos, donde únicamente se recibe un entorno inicial con unas condiciones y se devuelve una solución. Este esquema de funcionamiento hace que el cálculo iterativo de sistemas sea lento e ineficiente, convirtiendo en inviable su planteamiento para escenarios cambiantes.

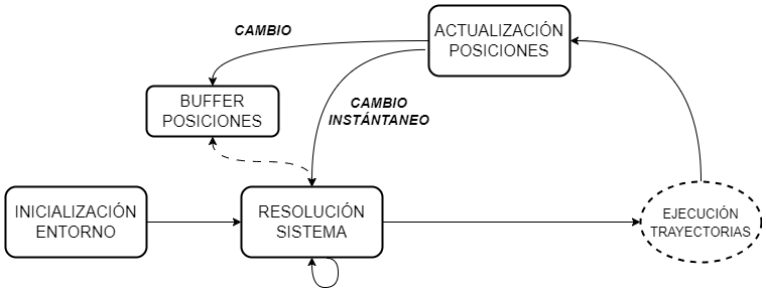


Figura 3.1: Sistema dinámico - Actualización entorno

Ante esto, se ha rediseñado el funcionamiento del algoritmo permitiendo la actualización directa del escenario de una manera robusta y eficiente. El esquema de funcionamiento (figura 3.1) comienza con una inicialización del entorno, consistente en leer todos los archivos de configuración donde se definen las variables básicas del escenario. Los entornos trabajados serán siempre de definición rectangular divididas en celdas de forma rectangular. Estas celdas serán la unidad más pequeña de trabajo del algoritmo de planificación. Las variables básicas de cualquier entorno son:

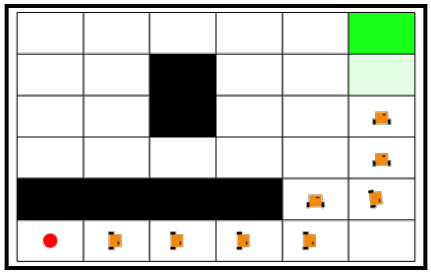
- W: número de celdas de ancho del sistema
- H: número de celdas de alto del sistema
- N_r: número de robots del sistema
- $\boldsymbol{Initial\_positions} \in \mathbb{N}^{N_r \times 2}$: matriz que almacena la posición inicial de robots por filas. La primera columna se corresponde con la ordenada X que toma valores enteros entre 0 y $W - 1$, y la segunda con la ordenada Y, que toma valores enteros entre 0 y $H - 1$.
- N_z: número de zonas de interés, sin contar los obstáculos
- $\boldsymbol{Destination\_probability} \in \mathbb{R}^{C_z \times 4}$: matriz que almacena cada una de las celdas que tienen una probabilidad de tener asignada una zona. Consta de 4 columnas y tantas filas como celdas se quieran definir (C_z). Las columnas, de la primera a la cuarta, se corresponden con la coordenada X de la celda, coordenada Y de la celda, número identificativo de la zona (1, 2, ..., N_z) y en último lugar una probabilidad entre 0 y 1.
- $\boldsymbol{Obstacles\_probability} \in \mathbb{R}^{H \times W}$: matriz que almacena la probabilidad de ser un obstáculo de cada celda. Cada elemento toma valores entre 0 y 1.

Los escenarios planteados se corresponden con entornos probabilísticos, donde la clasificación de la celda no tiene un carácter binario. Es por eso que es necesario la inicialización de las variables de probabilidad de zonas y obstáculos.

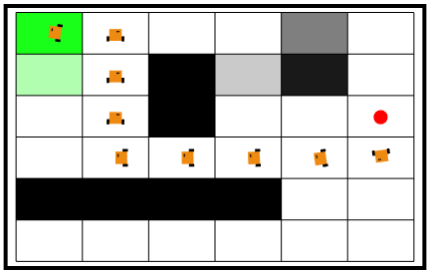
Una vez el escenario ha sido cargado, se procede a la resolución del sistema, obteniendo una serie de trayectorias para que los robots comiencen el desplazamiento. Posteriormente, en tiempo real se reciben actualizaciones constantes de la ubicación de los robots y de las diferentes zonas de interés, y cuando se produce un cambio frente al estado anterior se emite un evento de actualización. Este evento puede tener dos tipos, *CAMBIO INSTANTÁNEO* y *CAMBIO*. El primero de ellos introduce los cambios almacenados en el entorno de cálculo y produce que el sistema recalcule las trayectorias de manera inmediata. Por otro lado, el segundo tipo de evento únicamente almacena en un buffer del escenario los cambios detectados para que el sistema, una vez sea necesario recalcular, actualice las posiciones.

Ambos eventos tienen cabida en un sistema real, ya que para tener un sistema eficiente no es necesario recalcular constantemente todo el escenario, ya que esto produciría que el sistema fuese lento. Por otro lado, se podría tener un algoritmo que evaluase la magnitud de los cambios producidos en el entorno, para diferenciar cuando sería necesario recalcular de manera instantánea y cuando no.

En esta arquitectura implementada es necesario la definición del periodo de recálculo del sistema, pudiendo establecer una condición temporal (cada X segundos se realiza el recalculo) o espacial (cuando los robots avancen dos unidades espaciales del sistema de referencia del algoritmo). En la imagen 3.2 se puede apreciar iteración a iteración como se produce la modificación de trayectoria del robot cuando el sistema es consciente del cambio en el entorno. Como se puede apreciar en el ejemplo, el cambio de probabilidades se puede producir tanto en las zonas objetivo como en los obstáculos del sistema.



(a) Trayectoria previa al cambio entorno



(b) Trayectoria después cambio entorno

Figura 3.2: Replanificación de trayectoria por modificación entorno

3.2. Bloqueos y colisiones en las trayectorias

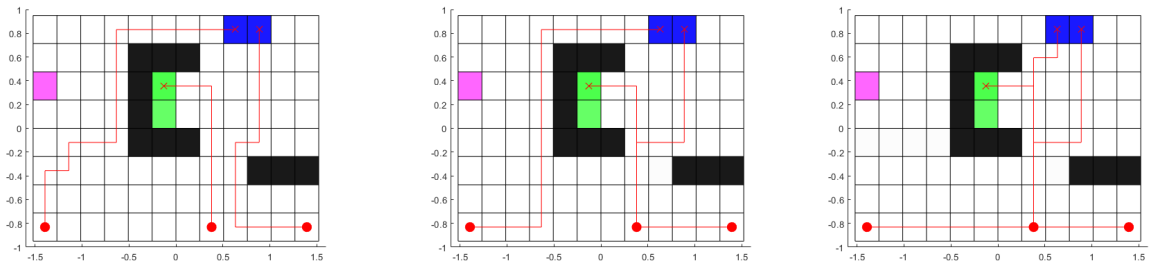
Cuando se trabaja con sistemas multi-robots uno de los aspectos más cruciales es el manejo de colisiones entre las entidades que conforman la red. Es muy importante tener mecanismos para evitar la producción de choques o incluso de bloqueos, que produjesen

situaciones que hiciesen inalcanzables la consecución de los objetivos de alto nivel impuestos. Para abordar esta situación se han implementado dos sistemas diferentes, uno de ellos en el problema de optimización a resolver, y otro independiente basado en la supervisión de las trayectorias.

3.2.1. Problema de optimización

Dentro de la definición teórica del problema de optimización (ecuación 2.5) se incluía la incógnita b, que se corresponde con el parámetro que define el número máximo de veces que cualquier celda es atravesada en las trayectorias resultantes. Este parámetro permite conocer si es necesario estudiar con detalle colisiones o no. Si $b = 1$, ningún robot que siga las trayectorias podrá sufrir colisiones ni bloqueos, ya que no se produce ningún tipo de cruzamiento. Sin embargo, si $b > 1$, las trayectorias en algún momento sufren cruzamientos. Aunque no tendrían por que producirse de manera simultánea, sería necesaria introducir un supervisor de colisiones para hacer frente a esta posibilidad. En la imagen 3.3 se muestran diferentes trayectorias para un mismo mapa en función del parámetro b.

El algoritmo de optimización original no incluía el parámetro b como una incógnita a optimizar, si no como únicamente una variable a definir ya que su implementación solamente se realizó de manera teórica. En este caso, se ha modificado el algoritmo para que sea una variable a optimizar, definiéndola por medio de la restricción 2.10. Además, se ha definido un peso asociado J_{col} que produce que se busque minimizar siempre el parámetro b, intentando reducir las probabilidades de colisiones siempre que sea posible.



(a) b=1 (b) b=3 (c) b=2

Figura 3.3: Muestra de diferentes casos optimizados con distintos parámetros b

3.2.2. Algoritmo del banquero

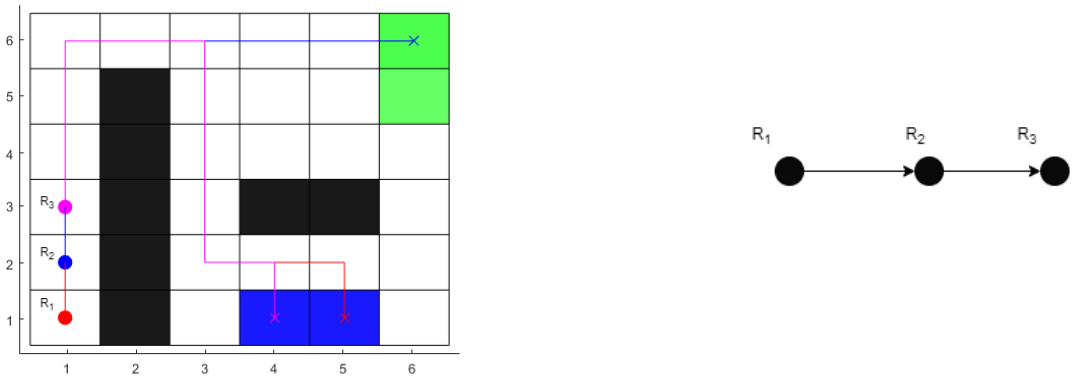
Una vez realizada la resolución de la red, es necesario verificar la seguridad de las trayectorias planteadas para evitar bloqueos. Con ese fin se introduce un sistema de supervisión de colisiones y bloqueos basado en el Algoritmos del Banquero [8] [9]. Este

algoritmo busca definir el orden de ejecución de las trayectorias para asegurar que a lo largo de ellas no se produce ninguna situación de bloqueo o colisión.

El Algoritmo del Banquero se basa en la generación de un grafo $H = (N, A)$ que representa las trayectorias que los robots deben realizar. Los nodos (N) se corresponden con los centros de las celdas por los cuales el robot debe desplazarse a lo largo de su trayectoria, y los arcos (A) representan los tramos entre cada nodo de la ruta.

El objetivo principal del algoritmo es evitar que se produzcan estados de bloqueo, el cual puede ser total, cuando ningún robot puede avanzar en su trayectoria, o un bloqueo parcial, cuando un conjunto de robots no pueden avanzar ni terminar sus misiones. La aplicación del algoritmo permite determinar si un estado concreto de los robots es un estado seguro o por el contrario se produce una situación de bloqueo.

Para comprobar la validez de una disposición se construye un grafo dirigido "wait-for"$(G = (N_g, A_g))$. En este grafo los nodos se corresponden con cada robot que tiene que desplazarse por el escenario, y únicamente existirá un arco de un nodo i a un nodo j $(robot_i, robot_j)$ si el robot j ocupa actualmente un arco necesario en la trayectoria del robot i, y por tanto un arco que el robot i necesita para terminar su misión. En la imagen 3.4b se recoge el grafo wait-for correspondiente a escenario mostrado en la imagen 3.4a.



(a) Trayectorias de ejemplo

(b) Grafo "wait-for"

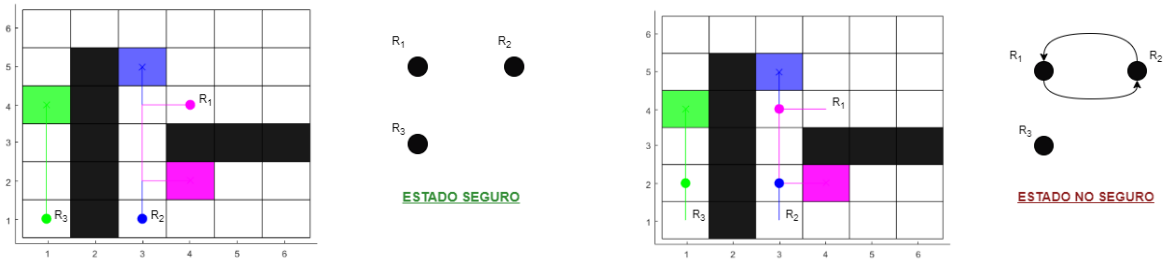
Figura 3.4: Ejemplo aplicación Algortimo Banquero

Para comprobar si un estado es seguro se realiza un proceso iterativo donde se comprueba si existe algún nodo que no tenga arcos de salida, lo cual implicaría que tiene todos los recursos libres para completar su misión, y en ese caso se eliminaría ese nodo del grafo y se repetiría la comprobación. El proceso consiste en comprobar que el grafo es acíclico. Si se encuentra una parte cíclica el estado no será seguro. El proceso devuelve un orden de ejecución de las trayectorias que asegura que se puede completar la misión, aunque no tiene por que ser la única opción plausible.

Para realizar el proceso de manera segura habría que desplazar cada robot de manera independiente paso a paso verificando que el siguiente estado es uno seguro. Para conseguir que el movimiento de los robots sea más fluido y las misiones se realicen de manera más eficiente, se realiza un proceso alternativo.

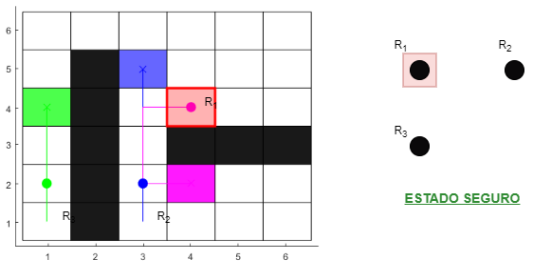
Partiendo de las posiciones iniciales, se comprueba la seguridad de que todos los robots avancen a la siguiente posición en su trayectoria al mismo tiempo. Si el resultado es que el estado no es seguro se procede a realizar una comprobación parcial, donde se prueba si desplazando solo una parte de los robots el estado es seguro. El objetivo principal es que en cada iteración estén el mínimo número de robots parados. Debido a eso, las pruebas parciales consisten en comprobar con el máximo número de robots, por tanto, se estudia primero todas las combinaciones moviendo $n-1$ robots, posteriormente $n-2$, y así sucesivamente hasta que se obtiene un estado seguro que es el que se abordará. Una vez se consigue un estado seguro se repite el proceso iterativo con todos los pasos restantes. En el presente trabajo no se analizan todas las posiciones hasta el final de la trayectoria, si no que se van calculando poco a poco según los robots van avanzando, ya que como se trabaja con sistemas dinámicos, con el fin de ahorrar coste computacional, se van calculando los movimientos de manera progresiva según los robots se mueven a lo largo de la trayectoria.

En la figura 3.5 se puede apreciar un ejemplo del proceso de cálculo que se sigue: partiendo de una posición inicial segura (3.5a) se intenta avanzar todos los robots a la vez (3.5b), pero se produce una situación no segura que produciría un bloqueo. Por tanto, se decide desplazar el máximo número de robots sin que se produzca un estado de bloqueo, por lo que se deja parado el robot 1 y se desplaza los robots número 2 y 3 3.5c. El robot 1 permanece parado hasta que desplazarlo no bloquee el sistema 3.5d y es cuando R_2 alcanza su destino final.

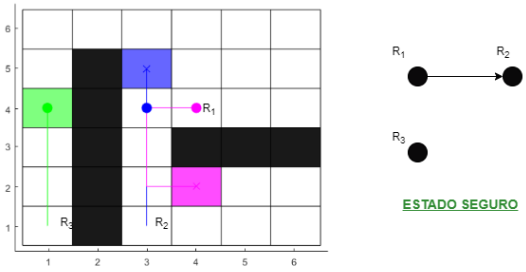


(a) Estado inicial

(b) Intento avance con todos los robots



(c) Movimiento con R_1 bloqueado



(d) Bloqueo solventado

Figura 3.5: Ejemplo aplicación Algoritmo Banquero

Se puede concluir que con este sistema se evitaría que se produjesen colisiones y bloqueos, y se resuelve el escenario de la manera más eficiente posible.

Algorithm 1 Ejecución algoritmo banquero

Resultado: Orden ejecución trayectorias de robots
  Cálculo de trayectorias para cada robot
  while *objetivo no alcanzado* **do**
    Avance posición robots
    if Nuevo estado es seguro **then**
      Añadir avance a trayectorias definitivas
    else
      Recalcular con otro robot avanzando
    end if
  end while

3.3. Misiones más complejas

El algoritmo de partida era comandado a través de condiciones booleanas (ejemplo: $A\&B$) que producían que las trayectorias finalizasen en una posición que satisficiera la condición. Sin embargo, en sistemas más complejos este método puede llegar a quedarse corto y requerirse la introducción de una variable más, la línea temporal.

Esta característica ha sido introducida por medio de las fórmulas de *Lógica Temporal Lineal* (LTL). Estas fórmulas permiten la codificación futura de condicionantes, lo cual otorga una mayor posibilidad de especificación. Este tipo de fórmulas combinan los operadores Booleanos (negación, y lógico, o lógico, implicación e equivalencia) con algunos operadores temporales que pueden ser unarios (variables individuales) o binarios (entran en juego dos variables):

Operadores unarios

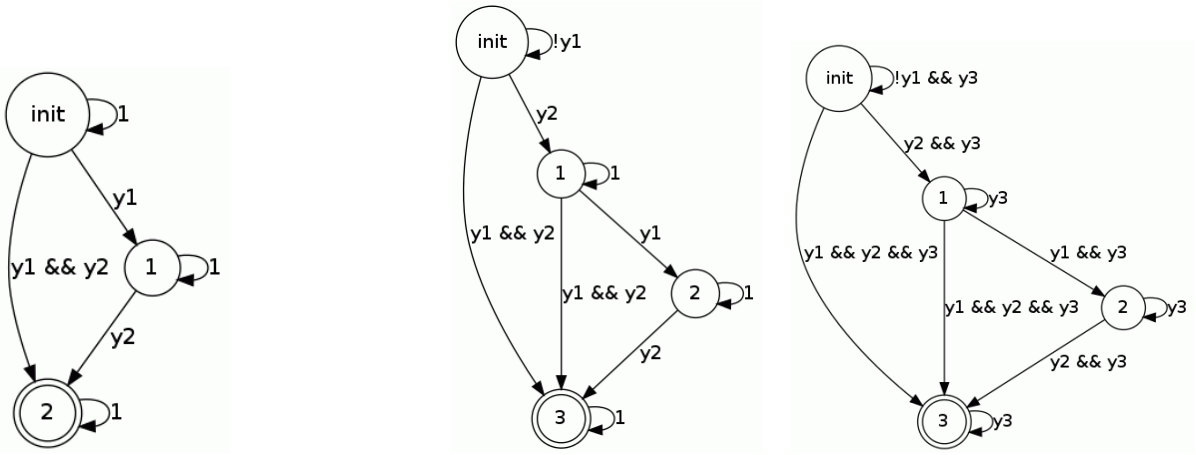
- $\mathbf{X}\varphi$: donde φ es una fórmula booleana. La fórmula LTL $\mathbf{X}\varphi$ es válida si φ tiene valor verdadero en el siguiente estado. Por ejemplo, la especificación $\mathbf{X}y_j$ impone que uno de los robots se debería mover a una región adyacente p_i tal que $h(p_i) = y_j$. Si no existe ninguna región adyacente a la actual con la salida y_j, la misión especificada con la fórmula LTL $\mathbf{X}\varphi$ no se puede cumplir.
- $\mathbf{F}\varphi$: φ tiene que tomar valor verdadero en algún momento futuro. Por ejemplo, si se quieren en el futuro alcanzar un estado con un robot en una región y_1 y otro en una región y_2, la fórmula LTL es $\mathbf{F}y_1 \wedge y_2$.
- $\mathbf{G}\varphi$: φ siempre debe ser verdadera. Por ejem'plo, si se quiere que en la celda y_1 siempre haya un robot, la fórmula LTL es $\mathbf{G}y_1$.

Operadores binarios

- $\alpha\mathbf{U}\varphi$: Este operador especifica que α debe ser verdadero continuamente hasta que φ se vuelva verdadero. Se utiliza para establecer una secuencia de eventos o estados. Por ejemplo, la expresión $y_1\mathbf{U}y_2$ indica que debe haber un robot la celda y_1 y que este debe moverse a la celda y_2 en un estado sucesor. Este operador garantiza que la condición inicial (y_1) se mantiene hasta que se cumple la condición final (y_2).
- $\alpha\mathbf{R}\varphi$: Este operador especifica que φ debe ser verdadero continuamente hasta que α se vuelva verdadero, incluyendo el momento en que esto sucede. En otras palabras, φ mantiene su verdad hasta la validación de α.

Para poder calcular las trayectorias de los robots y cumplir una misión expresada por una fórmula LTL es necesario realizar la conversión en una sucesión de condiciones booleanas. Para obtener el listado de condiciones es necesario transformar la fórmula LTL en un autómata de Büchi. Un autómata de Büchi consiste en un conjunto de estados y transiciones, un conjunto de estados iniciales y otro de estados finales. Las transiciones están etiquetadas por funciones Booleanas y el autómata cambia de estado cuando la función lógica asignada a la transición es verdadera. Un camino desde un estado inicial a un estado final y luego del estado final alcanzado volviendo al mismo estado final se genera por una secuencia de funciones Booleanas que permiten la ejecución de estas transiciones. Si este camino se ejecuta validando las funciones lógicas correspondientes a las transiciones, entonces la fórmula LTL se cumple. La fórmula entrada del problema se transforma a un autómata de Büchi usando el algoritmo presentado en [10]. En la figura 3.6 se pueden apreciar la transformación de diferentes fórmulas LTL en los autómatas. Importante de notar que en este trabajo estamos considerando una subclase de formulas LTL, llamada *safe-LTL*. En este caso, la secuencia de estados en el autómata de Büchi para satisfacer la fórmula es de longitud finita y basta con encontrar un camino desde un estado inicial

a un estado final y seguir las formulas Booleanas correspondiente a las transiciones entre los estados correspondientes.



(a) F(y1 & Fy2) (b) **F**(y1 & **F**y2) & ! y1**U**y2 (c) **F**(y1 & **F**y2) & ! y1**U**y2 & **G**y3

Figura 3.6: Diferentes ejemplos de transformación LTL en autómata de Buchi

Una vez generado el autómata, se ha implementado el algoritmo de Dijkstra para calcular un camino de coste mínimo entre un nodo inicial y uno final. A cada transición entre dos estados en el autómata de Büchi se le asigna un peso unitario, y el coste de llegar a cada nodo se obtendrá por medio de un bucle iterativo, en el cual se irán visitando todos los nodos y asignando el coste mínimo para llegar a cada uno de ellos. Posteriormente, desde el nodo de interés se puede recorrer de manera inversa obteniéndose el camino más corto.

Un aspecto a tener en cuenta es el hecho de que no todas las transiciones dentro del autómata de Büchi pueden ser posibles. Un ejemplo se puede apreciar en la figura 3.6c. En ella, si se tuviese que cumplir esta fórmula por medio del camino más corto, nunca se podría dar con dos o menos robots, ya que se necesitarían mínimo 3 robots. La única excepción sería que las superficies objetivo estuviesen superpuestas. Todos estos aspectos se tienen en cuenta y en la aplicación de Dijkstra se descartan aquellos caminos que no sean posible cumplir con las condiciones del escenario.

Tras la aplicación de este método se obtiene una secuencia de fórmulas booleanas que, si se evalúan verdadero en secuencia, la fórmula LTL se cumple. Únicamente habría que aplicar el algoritmo de planificación descrito en la Sección 2.1.2 de manera iterativa mientras se van cumpliendo cada una de ellas. Esta forma de definición de objetivos es una alternativa a la opción original y permite poder definir especificaciones de alto nivel de mayor complejidad, incluyendo aspectos temporales.

Capítulo 4

Bajo nivel

Una vez han sido definidas las trayectorias que deben acometer los robots se procede a trasladar estas al sistema real. Para realizar esta aplicación se requiere la transformación del sistema ficticio en el que se calcula la ruta a las dimensiones del sistema real. Para ello, se requiere tener en cuenta todas las restricciones con las que cuentan los equipos, tanto físicos como tecnológicos, e intentar replicar con la mayor exactitud las trayectorias definidas. Para testear el algoritmo de planificación se ha implementado el sistema en la plataforma Robotarium.

4.1. Seguimiento de trayectoria

El objetivo principal abordado en este control a bajo nivel consiste en implementar un sistema que sea capaz de controlar varios robots siguiendo una sucesión de puntos. El resultado obtenido del algoritmo de planificación es una sucesión de celdas discretas, mientras que los robots presentan un movimiento continuo.

Se realiza una discretización de la plataforma en un número de celdas igual al del algoritmo de planificación, teniendo en cuenta los límites físicos de la misma. Se ha definido un valor constante *SECURE AREA* que se corresponde con el porcentaje de longitud (respecto a cada una de las dimensiones) que se van a dejar con los límites de la plataforma. Esta variable es configurable y permite establecer una distancia de seguridad para prevenir el choque de los robots con las paredes.

Para comandar los robots se ha utilizado la librería base de la plataforma, que a través de una serie de funciones permite obtener un control completo sobre los dispositivos. Una de las ventajas que presenta la plataforma es que por medio del mismo código se puede ejecutar el escenario tanto el simulador como en el sistema real. En primer lugar, se realiza una inicialización de la librería, introduciéndole el número de robots y la posición inicial

de los mismos. La plataforma trabaja con tres coordenadas para posicionar los robots dentro del espacio 2D, siendo estas la posición del mismo (X e Y) y la orientación (θ_i). En este trabajo la orientación no es objeto de interés ya que no se ha tenido en cuenta como una variable a controlar.

Una vez inicializada la plataforma, se procede a instanciar una serie de controladores necesarios para su funcionamiento. Los robots realizan su movimiento a través de consignas de velocidad, por lo que para conseguir que los robots alcancen los objetivos se requerirá implementar un controlador de posición.

La plataforma cuenta con un control de posición propio que ha sido el utilizado en este trabajo. Este control dada una posición objetivo y una posición actual devuelve la consigna de velocidad necesaria a aplicar. Esta función acepta también una serie de limites funcionales como pueden ser velocidades máximas que permiten asegurarse de que en ningún momento se intenten introducir velocidades inalcanzables.

4.1.1. Discretización

La información recibida por el controlador de la plataforma consiste en un listado de celdas que el robot debe seguir para cumplir el objetivo impuesto. Esta sucesión de celdas no puede ser seguida directamente por el robot ya que el sistema de referencia de las celdas, en el cual cada celda tiene un número identificativo, es diferente al sistema de referencia utilizado por la plataforma. Debido a esto, es necesario realizar una transformación entre ambos sistemas.

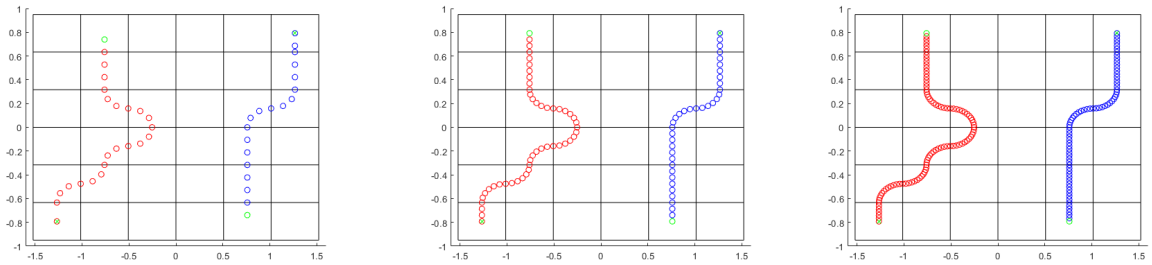
El módulo de discretización se encarga de, a partir de los datos extraídos del problema de optimización, transformarlo en una serie de puntos con coordenadas reales a seguir para cada uno de los robots por medio de la definición de una curva que realiza movimientos suaves para seguir las celdas objetivo en el orden impuesto. A la hora de implementar dicho algoritmo se vio que existían diversos métodos para obtener esta curva. Entre ellas se valoraron dos opciones. Por un lado, la implementación de un sistema sencillo conformado por líneas rectas y arcos de circunferencia, y por otro lado, se estudió la posibilidad de introducir *splines*. Los splines son un tipo de funciones definidas por medio de polinomios de bajo grado en un rango concreto. Gracias a su versatilidad, posibilitan aproximar curvas complejas con un polinomio sencillo. Sin embargo, posteriormente a la realización del estudio se descartó esta opción debido a la complejidad adicional que introducía sin aportar valor añadido. Por tanto, se optó por utilizar las líneas rectas y arcos circulares como opción para conformar las trayectorias (ecuaciones 4.1, 4.2 y 4.3).

$$y_{hor} = y_j,\ \forall\ x \in [x_{low}, x_{high}] \tag{4.1}$$

$$x_{ver} = x_k,\ \forall\ y \in [y_{low}, y_{high}] \tag{4.2}$$

$$\begin{cases} x_i = x_C + R \cdot cos(\theta_i) \\ y_i = y_C + R \cdot sin(\theta_i) \end{cases} \tag{4.3}$$

El proceso de generación de trayectorias se divide en dos etapas. En primer lugar, se realiza el estudio celda a celda de la forma que la trayectoria debe tener en ella, si debe ser una línea recta, o por otro lado una curva, teniendo en cuenta la celda anterior y la posterior. En segundo lugar, una vez la curva continua ha sido definida, se procede a discretizar cada tramo. Para controlar la precisión del proceso de discretizado se ha introducido un parámetro que el usuario puede modificar, consistente en el número de puntos por celda que la trayectoria debe contener. Este número permite adaptar el algoritmo a problemas que requieran de mucha exactitud, o a problemas sencillos que no necesitan tanta precisión. En base a este parámetro se define la trayectoria dividiendo el tramo en tantos puntos como el parámetro incluya. En la imagen 4.1 se puede observar los trazados discretizados de unas trayectorias de ejemplo para diferentes valores del parámetro.



(a) Línea con 3 nodos por celda (b) Línea con 6 nodos por celda (c) Línea con 12 nodos por celda

Figura 4.1: Discretización dela misma trayectoria con distintos parámetros de puntos por celda

4.1.2. Control sistema

Una vez las trayectorias han sido discretizadas, se aplica un control de posición sobre los robots de manera que se calculen las consignas adecuadas para alcanzar los puntos objetivos. Para verificar que los robots llegan a destino, se ha definido una función de error, como se puede apreciar en la fórmula 4.4. Cuando este error sea inferior a un límite impuesto se marcará la posición como alcanzada y se procederá a avanzar a la siguiente.

$$error_{pos}(x, y) = abs(\sqrt{(x - X_{target})^2 + (y - Y_{target})^2}) \tag{4.4}$$

$$error_{pos}(x, y) < E_{limite} \tag{4.5}$$

Cuando un robot ha discurrido por todos los puntos de la trayectoria, adquiere un estado de reposo, donde esperará a que todos los robots finalicen la trayectoria parcial. Posteriormente se volverá a calcular el algoritmo de optimización y la discretización parcial. Este bucle iterativo se repetirá hasta que la condición booleana inicial se satisfaga. Una vez cumplida, se procederá a repetir todo el cálculo iterativo para la siguiente condición si se están utilizando fórmulas LTL. Cuando todas las especificaciones impuestas sean cumplidas, el algoritmo de control finalizará. El sistema almacena todas las posiciones de cada robot, y al finalizar la simulación guarda los datos en un archivo para permitir realizar un análisis posterior.

4.2. Visualización

Una de las facetas mas importantes de todo sistema robótico moderno consiste en dotar al usuario de una interfaz que le permita ver fácilmente que esta ocurriendo en tiempo real. Para satisfacer esta necesidad, desde un primer momento se ha desarrollado un sistema que permitiese representar las distintas decisiones que los robots van tomando. Para agrupar estas funciones, se ha definido un módulo encargado de todas las funciones de dibujado.

Como los sistemas simulados son sistemas dinámicos, la representación del escenario tiene que ser capaz de adaptarse a lo que está sucediendo en tiempo real. Con el fin de simplificar y optimizar al máximo el sistema de representación se realizan subdivisiones modulares que realizan ejecuciones parciales de código para de esta manera solo actualizar lo que sea necesario y ahorrar coste computacional.

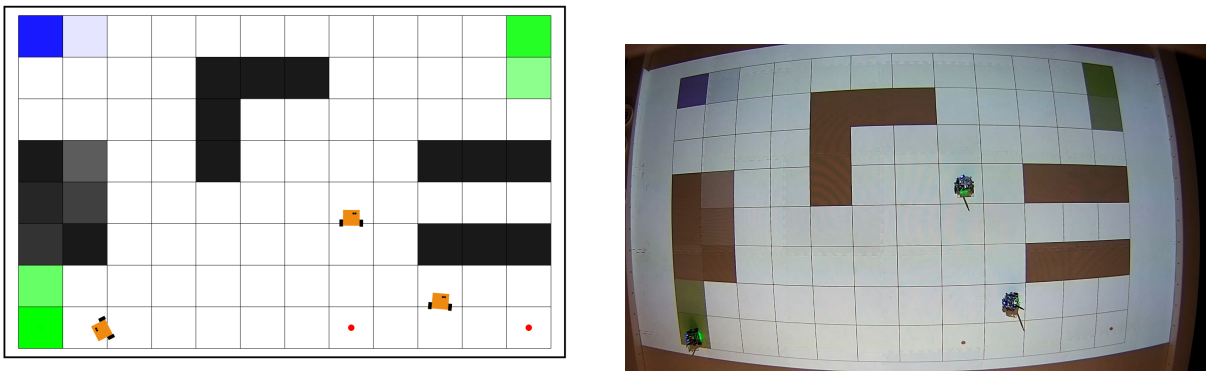
Para lograr representar el entorno dentro del área de proyección hay tener en cuenta los límites físicos del sistema y la superficie de seguridad, en la cual el sistema no operará. Existen 3 subsistemas, los cuales comparten un sistema de coordenadas común.

El primero de los módulos que es ejecutado en cualquiera de las simulaciones es el dibujador del mallado. Este módulo es el único que no se actualiza de manera dinámica ya que el sistema parte de la premisa que se encuentra siempre ante un entorno de tamaño fijo. Debido a la sencillez de su lógica, únicamente necesita como parámetro las dimensiones de entorno, el área de seguridad y el tamaño del mallado (ancho y alto). Para su representación se dibujan líneas verticales y horizontales que simulan el mallado.

Una vez finalizada esta tarea, tras la inicialización completa del sistema, se ejecuta un segundo módulo de dibujado encargado de la representación del entorno de trabajo. Este segundo módulo permite la actualización de la visualización para poder representar estados cambiantes de las diferentes zonas del sistema en tiempo real. Su función principal consiste en mostrar tanto las celdas objetivo como los obstáculos.

Para facilitar el entendimiento del sistema, se asigna un color negro a los obstáculos,

y un color distinto para cada una de las zonas objetivo. Como se está trabajando en un entorno probabilístico, se asigna a la componente transparente el valor de la probabilidad, haciendo que cuánto más visible sea una celda, mayor probabilidad de serlo tiene. Para realizar la representación de las diferentes celdas se utiliza el comando *fill* de Matlab, que permite un dibujado bidimensional de un polígono relleno y que a la vez facilita el control de la visualización. En la figura 4.2 se puede observar la comparativa de un mismo escenario en el entorno simulado en Matlab y la proyección en la plataforma real.

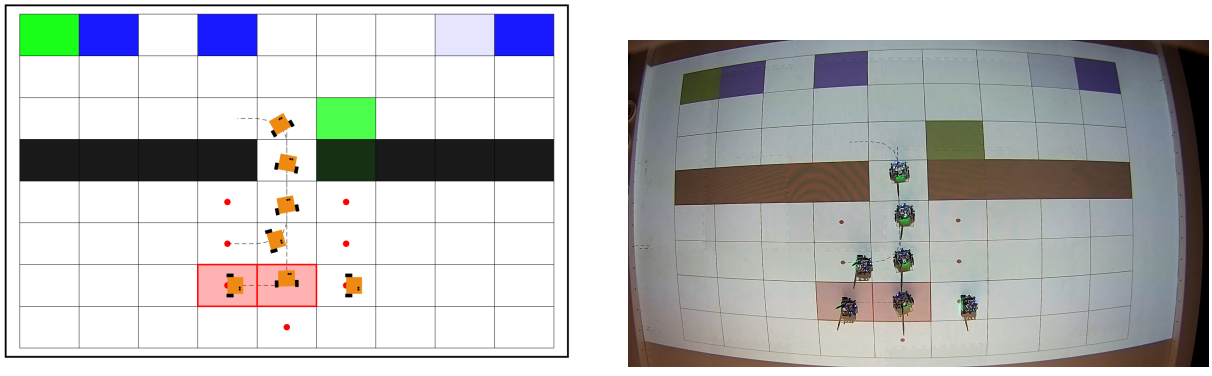


(a) Simulación (b) Plataforma real

Figura 4.2: Comparación entorno representado en plataforma real y simulada

En último lugar, se ha implementado un módulo cuya función principal es la representación de todos los componentes relacionados con la trayectoria. Esto abarca desde el dibujado de la trayectoria que el robot está recorriendo, hasta la posición inicial o el marcado de un robot que está parado para evitar una colisión.

La representación de la línea a seguir por el robot es dibujada por medio de líneas discontinuas, y es actualizada en cada recalculo del problema. Por otro lado, cuando el control de colisiones obliga a un robot a entrar en estado de espera, se actualiza el dibujado de la celda por medio de un marcado rojo, como se puede apreciar en la imagen 4.3.



(a) Simulación (b) Plataforma real

Figura 4.3: Comparación representación trayectorias y robots en espera

Este módulo aumenta la calidad del sistema desarrollado al conseguir una visualización en tiempo real, la cual dota al usuario de la capacidad de análisis y compresión del sistema para analizar el comportamiento de los robots.

Capítulo 5

Experimentos

Una vez desarrollada la herramienta por completo, se han definido una serie de escenarios para probar todos los complementos incluidos para verificar que funcionen correctamente. Estos escenarios han sido diseñados de tal manera que ponga a prueba el funcionamiento del algoritmo y puede llegar a ser representativo como una prueba de funcionamiento.

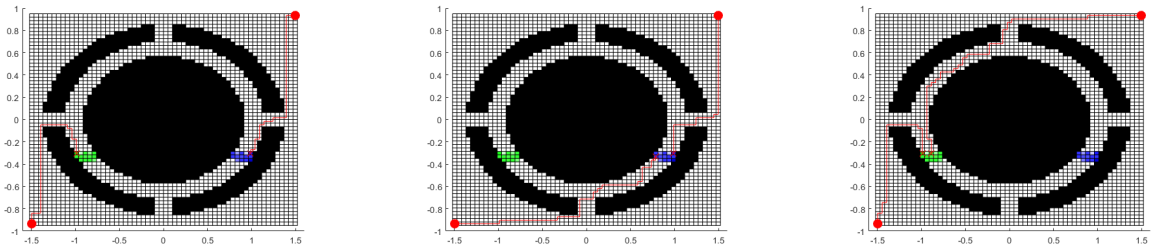
En algunos de ellos el criterio de funcionamiento es binario (funciona o no) y se han incluido una serie de frames extraídos del video resultante de la plataforma. En otros de ellos se han extraído métricas que permiten comprobar el rendimiento del módulo. Por último, se incluye un escenario en el cual se ponen a prueba todos los módulos al mismo tiempo, y se puede observar el comportamiento de la plataforma final.

5.1. Verificación funcionamiento

5.1.1. Planificación alto nivel

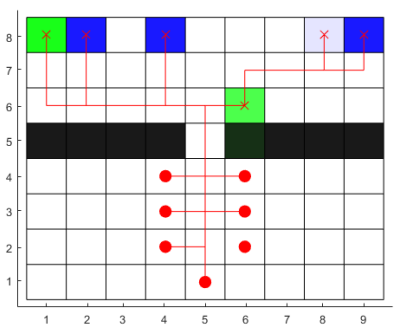
Algoritmo planificación

Uno de los componentes fundamentales del sistema desarrollado radica en el algoritmo de planificación. Este algoritmo devuelve trayectorias para cumplir un objetivo de alto nivel en un escenario concreto. En la figura 5.1 se puede apreciar las trayectorias obtenidas al someter un mismo entorno ante diferentes condiciones booleanas.

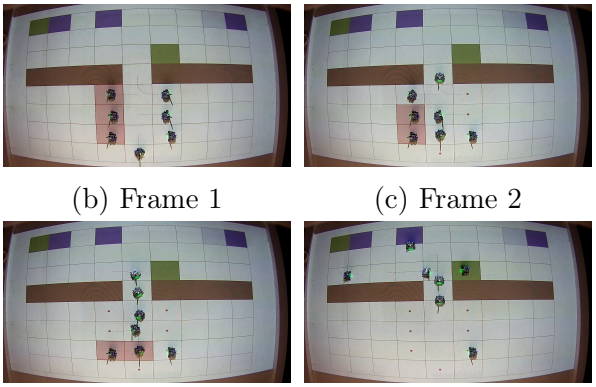


(a) A & B (b) !A & B (c) A & !B

Figura 5.1: Resultado de la optimización para un mismo entorno antes distintas formulas booleanas de entrada, en un sistema con dos robots y dos zonas de interés (A zona verde, B zona azul)



(a) Trayectorias resultantes de la planificación



(b) Frame 1 (c) Frame 2

(d) Frame 3 (e) Frame 4

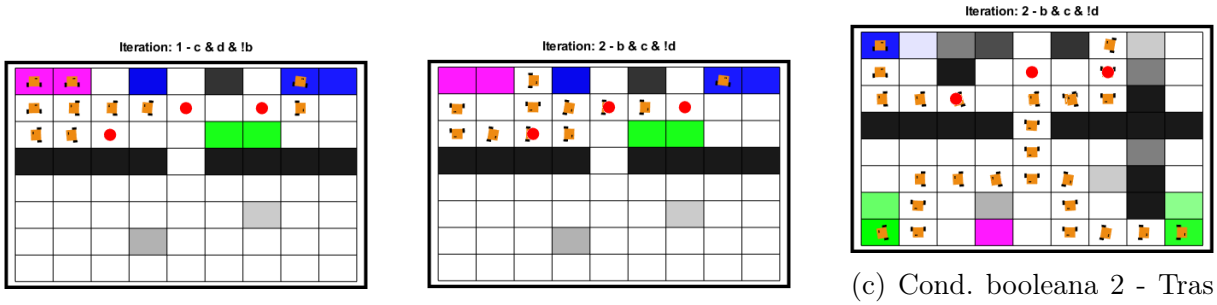
Figura 5.2: Ejemplo de la aplicación en la plataforma del algoritmo del banquero

Colisiones y bloqueos

El sistema desarrollado es capaz de manejar diferentes situaciones de bloqueo a través del algoritmo de optimización y el del banquero. En sistemas dinámicos con varios robots es un componente fundamental para asegurar la fiabilidad, robustez y eficiencia del sistema. Se han realizado diferentes ejemplos para cerciorarse del funcionamiento correcto. En las figuras 5.2 se puede ver la plataforma real ejecutando una simulación donde es necesario que los robots esperen para no colisionar entre ellos (celda roja).

Sistemas dinámico y condiciones complejas

En último lugar, se procede a ensayar el funcionamiento de la herramienta ante sistemas dinámicos y especificaciones de alto nivel más complejas. Para ello, se somete ante la siguiente condición LTL: "$F(d \ \& \ F(c)) \ \& \ (NOTdUc) \ \& \ NOT(b \ \& \ c \ \& \ d)$", donde los robots no podrán estar en la zona d hasta que no hayan estado en la c, y deberán



(a) Cond. booleana 1 (b) Cond. booleana 2 (c) Cond. booleana 2 - Tras cambio escenario

Figura 5.3: Evolución trayectorias de escenario con fórmula LTL y cambio dinámico de escenario (b ->zona verde; c ->zona azul; d ->zona morada)

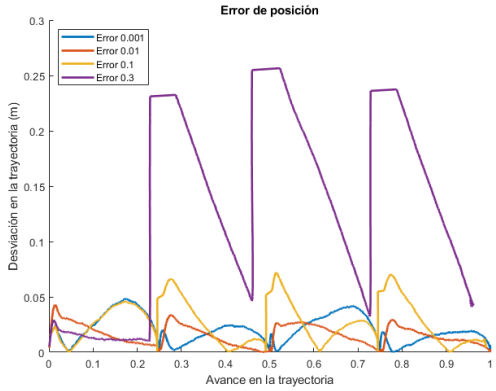
en la siguiente interacción de visitar c ir a la zona d. Además, no podrán estar en las tres zonas analizadas a la vez. El modulo LTL automáticamente descompondrá en dos iteraciones distintas la ejecución de la orden. Sin embargo, a mitad de la segunda iteración se actualizará por completo el escenario. Se puede ver el desarrollo de las trayectorias en la imagen 5.3

5.1.2. Control plataforma

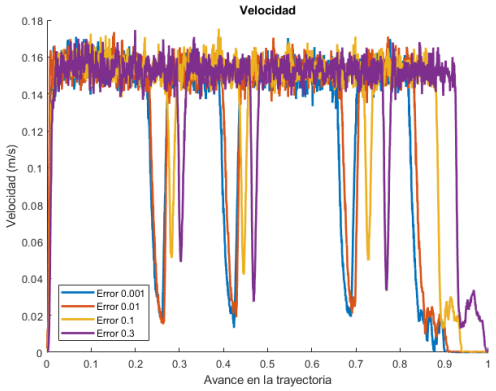
A lo largo de todo el trabajo, se ha podido ver en todas las simulaciones el entorno gráfico implementado que permite visualizar el escenario al usuario. En lo que respecta al control de bajo nivel de los robots, se ha definido una serie de métricas que permiten valorar la calidad del seguimiento de las trayectorias discretizadas por los robots.

Las métricas implementadas se basan en el error de distancia de los robots a la trayectoria planificada. Estás métricas permiten ajustar dos parámetros del control de movimiento de los robots. Por un lado, el número de puntos discretizados por celda, y por otro lado, la distancia de proximidad, consistente en la distancia mínima que debe tener un robot a un punto para avanzar al siguiente de la trayectoria.

En la gráfica 5.4 se puede observar los valores resultantes de someter un robot a una trayectoria rectangular con 4 puntos de control, uno en cada esquina. Se puede observar como el error de posición aumenta en los puntos de control cuanto mayor es el parámetro de error admitido. En la gráfica 5.4b se puede observar que la velocidad se mantiene constante entre puntos discreteados. Sin embargo, cuando se llega a los puntos de control, cuánto menor es el error admitido menor es la velocidad del robot para lograr satisfacer la distancia mínima.



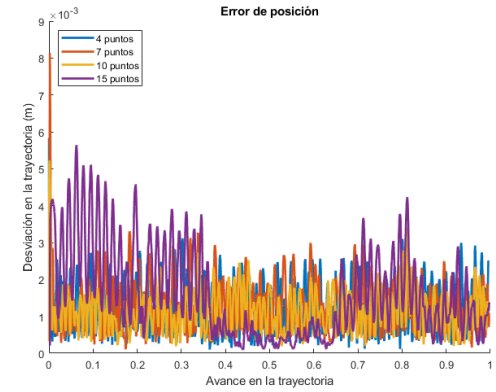
(a) Error de posición



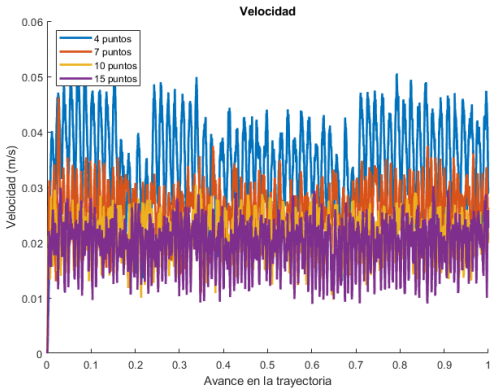
(b) Velocidad instantánea

Figura 5.4: Error de posición y velocidad de un robot ante una trayectoria rectangular con diferentes error admitidos

En último lugar, se realiza el mismo estudio variando el parámetro de puntos discretizados por celda. En la imagen 5.5 se puede apreciar los valores del sistema respondiendo ante diferentes configuraciones. El error de posición no se ve afectado prácticamente por el tipo de discretizado, ya que varía en un rango despreciable. Por otro lado, la velocidad sí que se ve afectada, ya que al aumentar el número de puntos la velocidad global del robot se ve reducida, ya que necesita ir corrigiendo la velocidad y posición de manera más precisa para alcanzar cada uno de los puntos.



(a) Error de posición



(b) Velocidad instantánea

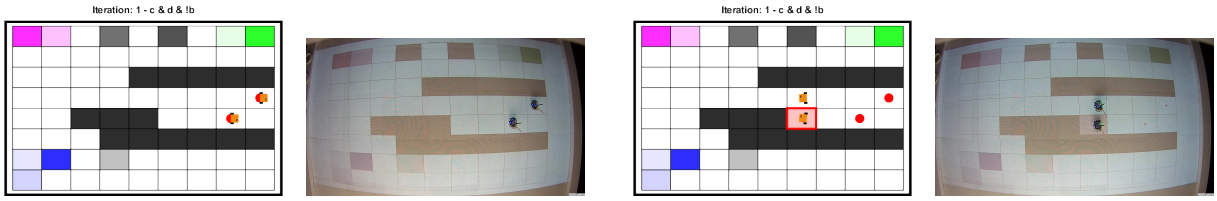
Figura 5.5: Error de posición y velocidad de un robot ante una diferentes tipos de discretizados

5.2. Implementación completa

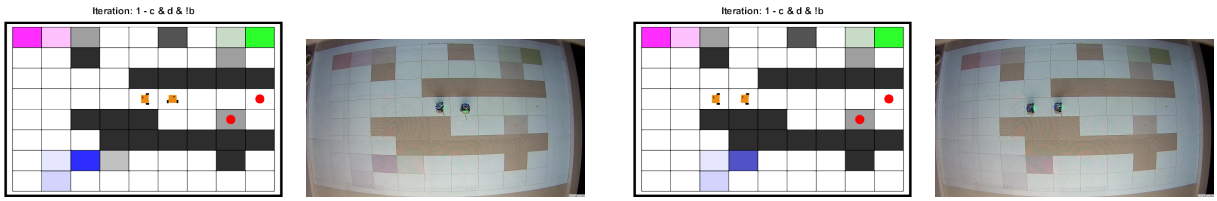
En último lugar, se ha planteado una simulación completa donde se pueden apreciar todos los módulos del sistema actuando al mismo tiempo. A continuación se dispone en las imágenes 5.6 algunos de los frames extraídos del vídeo del simulador. El escenario esta compuesto por dos robots los cuales deben cumplir la siguiente fórmula LTL:

$$G(c)\ \&\ d\ \&\ X(b) \tag{5.1}$$

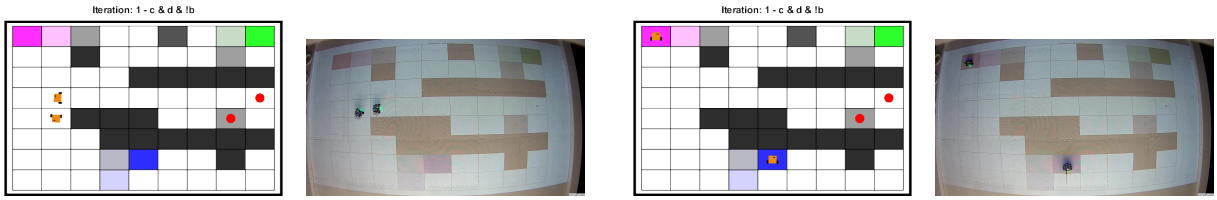
Esta fórmula busca posicionar uno de los robots siempre en una zona C, y otro inicialmente en la zona D, y para posteriormente desplazarse a una zona B. La ejecución se realiza en dos iteraciones y se actualiza progresivamente el estado del escenario, tanto los obstáculos como las zonas objetivo.



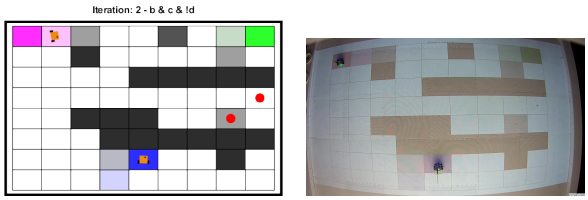
(a) Estado inicial (b) Algoritmo banquero



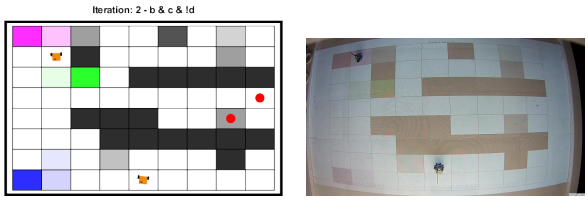
(c) Actualización 1 (d) Actualización 2



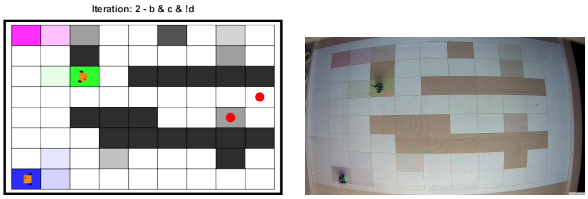
(e) Actualización 3 (f) Final iteración 1



(g) Inicio Iteración 2



(h) Actualización 4 1



(i) Final Iteración 2

Figura 5.6: Ejemplo completo del sistema de control

Como se puede apreciar, el sistema responde correctamente con trayectorias adecuadas y cumpliendo siempre las especificaciones de alto nivel. Con todos estos resultados se da por válido el sistema y se concluye que la implementación realizada es correcta.

Capítulo 6

Conclusiones

En un mundo cada vez más automatizado es necesario la mejora continua de los procesos para conseguir actividades cada vez más eficientes. Ese propósito es el que ha motivado la realización del presente trabajo, donde se ha desarrollado un sistema de multi-robots que permita operar en escenarios complejos con el fin de cumplir misiones impuestas.

En primer lugar, se ha implementado sobre el algoritmo original la capacidad de adaptación ante entornos dinámicos. Los escenarios donde trabajan los sistemas robóticos hoy en día son cada vez más complejos y variables, lo que implica un mayor número de señales a procesar. Hacer que los robots sean capaces de manejarse en este tipo de entornos garantiza la existencia de procesos más eficientes y seguros.

En segundo lugar, cada vez es más común la implementación de sistemas multi-robots, donde existe una coexistencia de equipos realizando diferentes actividades al mismo tiempo. En este tipo de escenarios, es fundamental una correcta supervisión de colisiones y bloqueos para evitar cualquier tipo de fallo. En el presente trabajo se ha abordado este problema desde varios puntos de vista.

En tercer lugar, hoy en día los sistemas cada vez ejecutan tareas de mayor complejidad, por lo que es requerido ser capaces de definir objetivos y tareas cada vez más completas. Con ese fin, se ha implementado un sistema que permite definir especificaciones con mayor precisión.

Una vez desarrollado el algoritmo de planificación se ha procedido a su prueba en una plataforma real. Para ello, se ha optado por la plataforma Robotarium, que permite la simulación real de escenarios con multi-robots. Se ha podido probar el funcionamiento de cada uno de los módulos de manera individual y conjunta.

La automatización es una demanda creciente en el mundo actual debido a la búsqueda de la máxima eficiencia y productividad. Este trabajo aborda la necesidad de avance de los algoritmos de control, extrayendo resultados muy significativos y optimistas de cara a

la implementación de este tipo de sistemas en instalaciones reales.

6.1. Trabajo futuro

Este trabajo realizado puede servir a modo de punto de partida para otros estudios, ya que el recorrido dentro del sector robótico es muy grande y tienes muchas ramas todavía en las que profundizar. Uno de los rumbos de investigación viables relacionados con el algoritmo de planificación tiene que ver con los sistemas distribuidos. El sistema tratado en este trabajo se corresponde con un sistema centralizado, con una única ejecución del algoritmo. Sin embargo, el rumbo actual de la industria tiende a tener equipos individuales con su propia lógica, es decir, con una planificación distribuida. Esto permite una aceleración del proceso de cálculo y redundar el número de equipos con lógica para evitar fallos, aunque esto requeriría una mejora en la comunicación entre los robots para tener una correcta estimación del escenario.

Por otro lado, respecto al control de bajo nivel de los robots, se podría estudiar la implementación de una generación de trayectorias más avanzadas que aumentasen la eficiencia del sistema, como por ejemplo la introducción de splines para conseguir trazados más suaves, o una discretización a trozos para reducir coste computacional.

Apéndice A

Anexos

A.1. Algoritmo de planificación

El algoritmo de planificación a alto nivel utilizado definido en la sección 2.1.2 tiene su origen en un artículo de de referencia [3]. En dicha sección se realiza una introducción al algoritmo explicando superficial del sistema, definiendo sus componentes más importantes para el entendimiento de la planificación. Sin embargo, lograr una profundización conceptual sobre el algoritmo puede permitir el desarrollo de una mayor personalización en casas futuros. El objetivo de este capítulo es la explicación completa del algoritmo, defiendo cada uno de los condicionantes del mismo.

El algoritmo de planificación se basa en la resolución de un problema de optimización que permite la obtención de las trayectorias de los robots. Estas rutas a seguir buscan conseguir satisfacer una serie de condiciones de destino, colocando los robots en unas determinadas aéreas concretas. El procedimiento de resolución se realiza dividendo en primer lugar el escenario en unidades más pequeñas. En el caso tratado en el trabajo se utilizan celdas rectangulares.

El entorno de trabajo se corresponde con un escenario probabilísticos, donde las celdas (unidades elementales) se pueden corresponden con un obstáculo o con una zona objetivo con una cierta probabilidad, no únicamente una asignación binaria de ser o no ser. Tal y como se han comentado a lo largo del trabajo, trabajar con entornos probabilísticos permite una asociación mucho más sencilla con herramientas de visión computacional, donde la detección va correlativa siempre a una probabilidad.

Con lo expuesto anteriormente, en un entorno planteado, una misma celda puede llegar a tener probabilidades de ser diferentes etiquetas y obstáculos, pero gracias al trabajar con un problema de optimización siempre se buscará maximizar la probabilidad de éxito del objetivo. El objetivo del problema se definirá por medio de una especificación de alto nivel

con una serie de operadores binarios. Cada zona (etiqueta) del escenario tendrá asociado una letra (en función del orden de definición), y por medio de una formula booleana se definirá el objetivo final del sistema.

Ejemplo: En un mallado con obstáculos y dos zonas etiquetadas, si se quiere que los robots únicamente vayan a la primera de ellas se puede definir la condición booleana de "B & !C". La primera letra queda reservada para los elementos obstáculos, que nunca serán incluidos en la condición booleana.

La definición del problema de optimización se expresa en la ecuación A.1. Se corresponde con un problema lineal donde se busca minimizar una función coste $c(x)$ cumpliendo una serie de restricciones, las cuales expresarse mediante desigualdades o igualdades.

$$\begin{aligned} \min_{X} \quad & c(x) = \sum_{i=1}^{3} J_i \cdot X \\ & con \begin{cases} A_e q \cdot X = b_e q \\ A \cdot X \leq b \end{cases} \end{aligned} \tag{A.1}$$

Las variables de entradas del problema serán:

1. ***N:*** Red de Petri de movimiento de Robots *(RMPN)* con la definición del entorno
2. $\boldsymbol{\varphi}$: condición booleana a cumplir
3. $\mathbb{P}_{\boldsymbol{ik}}$: matriz con la distribución probabilística para todos los lugares y todas las etiquetas (obstáculos + zonas de interés).

La optimización consistirá en, partiendo de un marcado inicial $\boldsymbol{m_0}$ definido en la *RMPN*, alcanzar un marcado $\boldsymbol{m^*}$ que asegure cumplir la fórmula de alto nivel y minimizar el coste asociado. A continuación se definen por partes las diferentes restricciones y costes del problema para poder modelar el comportamiento y calcular las trayectorias.

A.1.1. Evolución del marcado

Para modelar el comportamiento de movimiento de los robots se hacen uso de los elementos definidos en la *RMPN*. Teniendo una situación inicial con un marcado m_k, un movimiento es simulado por medio del disparo de una transición t_j activada siempre y cuando $m_k \geq \boldsymbol{Pre}[\boldsymbol{\cdot}, \boldsymbol{t_j}]$ (columna correspondiente a la transición j).

Si t_j es disparado, el nuevo marcado será:

$$\boldsymbol{m}_{k+1} = \boldsymbol{m}_k + \boldsymbol{C}[\cdot, t_j] \tag{A.2}$$

donde

$$\boldsymbol{C} = \boldsymbol{Post} - \boldsymbol{Pre} \tag{A.3}$$

Si en vez de dispararse únicamente una transición, se disparan un conjunto de ellas el marcado final $\boldsymbol{m}$ será:

$$\boldsymbol{m} = \boldsymbol{m}_k + \boldsymbol{C} \cdot \tilde{\sigma} \tag{A.4}$$

definiendo el vector $\tilde{\sigma}$ como $\tilde{\sigma} \in \mathbb{N}^T$, donde cada elemento recoge el número de disparos de cada transición.

Con la igualdad A.4 quedaría definido la restricción asociada a la evolución de los marcados para asegurarnos siempre que los destinos propuestos son alcanzables. Por último, con el fin de reducir el número de movimientos realizados dentro de la red se define un coste a optimizar que penaliza el número de movimientos realizados, descrito con la ecuación A.5.

$$J_\sigma = \mathbf{1}_T^T \cdot \tilde{\sigma} \tag{A.5}$$

A.1.2. Satisfacción probabilística de la ecuación booleana

Para cada una de las etiquetas ($y_k \in Y$), se define una variable booleana x_k tal que

$$x_k = \begin{cases} 1 & \text{si } y_k \text{ está activada} \\ 0 & \text{si } y_k \text{ no está activada} \end{cases} \tag{A.6}$$

En el artículo [11] se desarrolla un algoritmo que transforma la condición booleana en una serie de desigualdades a cumplir para que se verifique la condición dada. Introduciendo estas restricciones en el problema de optimización se conseguiría forzar el cumplimiento de la condición siempre.

$$\boldsymbol{A}_task \cdot \boldsymbol{x} \leq \boldsymbol{b}_task \tag{A.7}$$

Sin embargo, este trabajo tiene un planteamiento probabilístico, porque será necesario sustituir el anterior método por otro planteamiento, en concreto el definido en el artículo [3] Asumiendo que la probabilidad de que una celda pertenezca a una etiqueta es

independiente, para un marcado $\boldsymbol{m}$ la probabilidad de que la etiqueta y_k este activada es igual a:

$$\mathbb{P}(y_k) = 1 - \prod_{p_i \in P, \boldsymbol{m}[p_i]>0} (1 - \mathbb{P}_{ik}) \tag{A.8}$$

Para poder implementar la ecuación A.8 en el problema lineal, se requiere realizar su transformación. Para ello se define el logaritmo de la probabilidad complementaria de que la celda p_i tenga activa la etiqueta y_k:

$$\omega = log(1 - \mathbb{P}_{ik}) \tag{A.9}$$

A partir de esa definición, se obtienen las siguientes matrices de probabilidad asociadas a cada celda y etiqueta:

$$\boldsymbol{w} = [w_{1k}, ..., w_{Pk}] \in \mathbb{R}^P \tag{A.10}$$

$$\boldsymbol{W} = [w_1, ..., w_M]^T \in \mathbb{R}^{MxP} \tag{A.11}$$

Posteriormente, se establece un variable τ que se corresponde con el umbral probabilístico. Se buscará satisfacer todas las etiquetas presentes en φ que cumplan $\mathbb{P}_{ik} > \tau$. Este límite τ tomará valores entre 0 y 1.

El sistema resultante es:

$$\begin{cases} -\mathbf{W} \cdot \mathbf{m} + \tau_{\mathbb{P}} \cdot \mathbf{1}_M & \leq N \cdot \mathbf{x} \\ -\mathbf{W} \cdot \mathbf{m} + \tau_{\mathbb{P}} \cdot \mathbf{1}_M & \leq N \cdot \mathbf{x} \end{cases} \tag{A.12}$$

donde:

1. $\tau_{\mathbb{P}} = log(1 - \tau)$
2. $\mathbf{1}_M$ es un vector unitario
3. N es un numero grande mayor que el número de robots del sistema

Como se puede apreciar en la ecuación A.12, si la variable booleana x_k toma valor de cero, se fuerza que $\boldsymbol{w}_k^T \cdot \boldsymbol{m} \geq \tau_{\mathbb{P}}$, obligando a que exista un robot en esa posición.

Aunque el problema situará un robot en aquellas celdas que cumplan las restricciones anteriormente planteadas, se busca maximizar la probabilidad de éxito de la misión. Con

este fin se añade un coste para maximizar las probabilidades de éxito de cumplimiento de la condición teniendo en cuenta las celda con un probabilidad inferior al umbral:

$$J_{\mathbb{P}} = N \cdot \mathbf{1}_M^T \cdot \boldsymbol{W} \cdot \boldsymbol{m} \tag{A.13}$$

A.1.3. Disminución probabilidad obstáculos

De la misma manera que en el caso anterior, se busca realizar un planteamiento probabilístico para que los robots eviten los obstáculos. Para ello, se hace uso del vector resultante de la ecuación $\boldsymbol{Post} \cdot \tilde{\sigma}$, que se corresponde con un vector que almacena el número de ocasiones que cada lugar es visitado. Se establece un límite σ_0 para el cual, si la probabilidad de que una celda sea un obstáculo es mayor, esta sea evitada ($\sigma_0 \in [0, 1]$). Para poder realizar la traducción matemática se define la función $\eta(t_j)$

$$\eta(t_j) = \begin{cases} 1 & \boldsymbol{Post}[t_j, p_i] = 1 \text{ y } \mathbb{P}_{iM} > \tau_0 \\ 0 & \text{si no } (\boldsymbol{Post}[t_j, p_i] = 1 \text{ y } \mathbb{P}_{iM} > \tau_0) \end{cases} \tag{A.14}$$

De esta forma, con la restricción A.15 se evitarían todos los obstáculos al no dispararse las transiciones que lleven a obstáculos con una probabilidad superior al límite.

$$\eta^T \cdot \tilde{\sigma} = 0 \tag{A.15}$$

Por último, se introduce un término más a la función de coste para evitar los obstáculos con una menor probabilidad y favorecer las rutas que no la utilicen. Este término es negativo ya que la probabilidad logarítmica $\boldsymbol{w}$ es la complementaria a la probabilidad de obstáculos, y por tanto se busca maximizar este término.

$$J_{obs} = -N \cdot \boldsymbol{w}_M^T \cdot \boldsymbol{Post} \cdot \tilde{\sigma} \tag{A.16}$$

A.1.4. Evasión de colisiones

Al trabajar con entornos multi-robot es necesario estudiar las colisiones entre los equipos. Para ello se introduce la restricción A.17, la cual busca limitar el número de veces que los robots pasan por una misma celda. La constante b puede ser una variable dada por el usuario como parámetro de la simulación, o ser introducida en el problema de optimización para que sea lo más pequeña posible por medio del coste asociado J_{col}.

$$\boldsymbol{Post} \cdot \tilde{\sigma} \leq b \cdot \mathbf{1}_P \tag{A.17}$$

$$J_{col} = b \tag{A.18}$$

Por medio de la restricción introducida el sistema buscará minimizar el número de veces que los robots pasan por la celda. Si finalmente $b = 1$, no sería necesario preocuparse por posibles colisiones entre equipos.

A.1.5. Problema optimización completo

Si se juntan todas las expresiones anteriormente citadas, obtenemos un problema de optimización donde las incógnitas conforman el siguiente vector:

$$\boldsymbol{X} = \begin{Bmatrix} \tilde{\boldsymbol{\sigma}} \\ \boldsymbol{m} \\ \boldsymbol{x} \\ b \end{Bmatrix} \tag{A.19}$$

siendo:

- $\boldsymbol{m}$: marcado final del sistema
- $\tilde{\boldsymbol{\sigma}}$: número de veces que se dispara cada transición
- $\boldsymbol{x}$: estado final de cada una de las zonas definidas (etiquetas).
- b : número maximo de veces que los robots pasas por una misma celda

y el problema de optimización queda de la siguiente manera:

$$\begin{aligned} &\min_{\mathbf{m},\tilde{\sigma},\mathbf{m},b} \quad c(x) = J_\sigma + J_{\mathbb{P}} + J_o bs + J_c ol \\ &con \begin{cases} \mathbf{m} & = \mathbf{m}_k + \mathbf{C} \cdot \tilde{\sigma} \\ \eta^T \cdot \tilde{\sigma} & = 0 \\ \mathbf{A}_t ask \cdot \mathbf{x} & \leq \mathbf{b}_t ask \\ -\mathbf{W} \cdot \mathbf{m} + \tau_{\mathbb{P}} \cdot \mathbf{1}_M & \leq N \cdot \mathbf{x} \\ \mathbf{W} \cdot \mathbf{m} - & \leq N \cdot (\mathbf{1}_M - \mathbf{x}) \\ \mathbf{Post} \cdot \tilde{\sigma} & \leq b \cdot \mathbf{1}_P \end{cases} \end{aligned} \tag{A.20}$$

Bibliografía

[1] ACRE, "Tecnología dron en agricultura de precisión. DJI AGRICULTURE," https://grupoacre.es/tecnologia-dron-en-agricultura-de-precision-dji-agriculture/.

[2] R. Arias, "Robots que ayudan en la clasificación de paquetes," https://www.instalacioneslogisticas.com/noticias/997-robots-que-ayudan-en-la-clasificacion-de-paquetes.html.

[3] E. Montijano and C. Mahulea, "Probabilistic multi-robot path planning with high-level specifications using petri net models," in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*. IEEE, Aug. 2021. [Online]. Available: https://doi.org/10.1109/case49439.2021.9551515

[4] D. Padua, Ed., *Encyclopedia of Parallel Computing*. Springer US, 2011. [Online]. Available: https://doi.org/10.1007/978-0-387-09766-4

[5] C. Mahulea, R. González, E. Montijano, and M. Silva, "Planificación de trayectorias en sistemas multirobot utilizando redes de petri. resultados y problemas abiertos," *Revista Iberoamericana de Automática e Informática industrial*, vol. 18, no. 1, p. 19, Dec. 2020. [Online]. Available: https://doi.org/10.4995/riai.2020.13785

[6] D. Pickem, P. Glotfelter, L. Wang, M. Mote, A. Ames, E. Feron, and M. Egerstedt, "The robotarium: A remotely accessible swarm robotics research testbed," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, May 2017. [Online]. Available: https://doi.org/10.1109/icra.2017.7989200

[7] S. Wilson, P. Glotfelter, L. Wang, S. Mayya, G. Notomista, M. Mote, and M. Egerstedt, "The robotarium: Globally impactful opportunities, challenges, and lessons learned in remote-access, distributed control of multirobot systems," *IEEE Control Systems*, vol. 40, no. 1, pp. 26–44, Feb. 2020. [Online]. Available: https://doi.org/10.1109/mcs.2019.2949973

[8] L. Kalinovcic, T. Petrovic, S. Bogdan, and V. Bobanac, "Modified banker's algorithm for scheduling in multi-agv systems," in *2011 IEEE International Conference on Automation Science and Engineering*. IEEE, Aug. 2011. [Online]. Available: http://dx.doi.org/10.1109/CASE.2011.6042433

[9] J. B. G. Barreto, "Diseño e implementación de un algoritmo que evite colisiones en un sistema multi-robot utilizando el Modified Banker's Algorithm," https://zaguan.unizar.es/record/96262/files/TAZ-TFM-2020-1101.pdf.

[10] P. Gastin and D. Oddoux, "Fast LTL to büchi automata translation," in *Computer Aided Verification*. Springer Berlin Heidelberg, 2001, pp. 53–65. [Online]. Available: https://doi.org/10.1007/3-540-44585-4_6

[11] C. Mahulea and M. Kloetzer, "Robot planning based on boolean specifications using petri net models," *IEEE Transactions on Automatic Control*, vol. 63, no. 7, pp. 2218–2225, Jul. 2018. [Online]. Available: https://doi.org/10.1109/tac.2017.2760249