PROYECTO INTEGRADOR DE INGENIERÍA MECÁNICA

IMPLEMENTACIÓN DE UN CONTROLADOR HÍBRIDO EN UN SISTEMA ROBÓTICO AUTÓNOMO

Tomás Liendro

Dr. Marcelo Kuperman **Mgtr. Sebastián Zudaire**

Directores

Miembros del Jurado

Dr. Andrés Etchepareborda (División de Control de Procesos, CNEA)
Dra. Sol Pedre (División de Robótica y Automatización, CNEA)

Junio de 2020

Física Estadística Interdisciplinaria, Centro Atómico Bariloche

Instituto Balseiro
Universidad Nacional de Cuyo
Comisión Nacional de Energía Atómica
Argentina

INVENTARIO 24112
12.05.21
Biblioteca Leo Falicov

A mi familia

Índice de contenidos

Índice de figuras

Resumen

En este trabajo se presenta el proceso de diseño e implementación llevado a cabo para la construcción de un robot autónomo basado en controladores híbridos. Haciendo uso de la técnica de *planning*, se logró planificar misiones de patrullaje en zonas con y sin obstáculos (*motion planning*), misiones donde el robot debe emitir una alerta al ingresar a una región y misiones de ordenamiento de una caja en un entorno acotado (*task planning*). Se utilizó el software MTSA (*The Modal Transition System Analyser*) para la síntesis de controladores discretos «correctos por construcción» que satisfagan un modelo del sistema robótico real y las propiedades que en dicho sistema deben cumplirse. Además, se presenta una metodología que pretende sistematizar el proceso de diseño de robots basado en controladores híbridos a partir de un enfoque en el que la especificación de la misión tiene un rol central. El proceso de diseño fue validado mediante la realización de diferentes misiones con un prototipo robótico construido por manufactura aditiva (impresión 3D) y con un entorno de simulación diseñado para tal fin.

Palabras clave: PLANNING, CONTROLADORES HÍBRIDOS, SÍNTESIS DE CONTROLADORES, ROBÓTICA, ROBOT AUTÓNOMO

Abstract

This work presents the design and implementation process carried out for the development of an autonomous robot based on hybrid controllers. By making use of the *planning* technique, it was possible to generate plans for patrol missions in areas with and without obstructions (*motion planning*), missions in which the robot has to issue an alert when entering a specific region and box-sorting missions in an enclosed area (*task planning*). The MTSA (*The Modal Transition System Analyser*) software was used to synthesize correct-by-construction discrete controllers that satisfy a model of the real robotic system and a set of properties that must be fulfilled in said model. Furthermore, this project presents a methodology that intends to systematize the design process of robots based on hybrid controllers, through an approach focused on the mission specification. The design process was validated by means of different mission tests using a robotic prototype constructed from additive manufacturing (3D printing) and a simulation environment exclusively designed for these tests.

Keywords: PLANNING, HYBRID CONTROLLERS, CONTROLLER SYNTHESIS, ROBOTICS, AUTONOMOUS ROBOT

Capítulo 1

Introducción

"El comienzo es la parte más importante del recorrido".
— Platón, filósofo y polímata de la Antigua Grecia

1.1. Motivación

Es inevitable hoy en día mirar hacia atrás en el tiempo y no sentirse asombrado por los resultados del avance tecnológico de la humanidad a lo largo de la historia en materia de ciencia y tecnología. Avances revolucionarios como el proyecto *Curiosity* o los robots humanoides han sido y son las pruebas del progreso de este área del conocimiento que, al día de hoy, tiene una perspectiva de aplicación tan amplio que se espera que perdure y prospere por muchos años más.

El anhelo por explorar los límites de la capacidad humana, han llevado a científicos y tecnólogos a descubrir nuevos sectores del conocimiento inimaginables algunas décadas atrás. La inteligencia artificial, el Internet de las Cosas (*Internet of Things*), las telecomunicaciones y robótica son el foco de atención de las empresas y de los centros de investigaciones actuales en la búsqueda del avance tecnológico orientándose hacia un futuro basado en el desarrollo sustentable con mejores estándares de vida.

En el marco de la robótica, las empresas WAYMO de Google Inc., BMW y Volvo han sido pioneras en la transformación del clásico concepto del transporte de personas y cargas introduciendo los vehículos autónomos, o *self-driving vehicles*. Esta evolución ha sido posible gracias al desarrollo y mejoras de tecnologías de detección complejas como láser, radar, sistema de posicionamiento global y visión artificial, y de sistemas de control que interpretan la información e identifican la ruta apropiada, así como los obstáculos y la señalización relevante [1]. El problema de navegación en un entorno tan complejo donde debe reducirse la marcha en zonas urbanas o previo a una curva, detenerse para permitir el paso de los peatones u otros vehículos o detenerse en presencia

de la luz roja del semáforo, no son escenarios sencillos de modelar con las nociones básicas de «if(**condition**) then...else» de las técnicas de programación tradicionales, dado el sinnúmero de distintos escenarios posibles. Es necesario entonces, contar con herramientas más generales que puedan aglomerar todas las situaciones con las que un vehículo pueda llegar a encontrarse y que permitan razonar sobre estos.

1.2. Técnicas de *Planning*

Un enfoque al problema de navegación, puede ser tomado gracias a las contribuciones de los campos de la Teoría de Control y la Ingeniería en Software. Por un lado, desde la Robótica hay muchos enfoques que estudian el problema de la interacción entre las variables de un sistema robótico y las del entorno donde éste opera (por ejemplo en [2]). Por otra parte, desde la Ingeniería en Software se han desarrollado técnicas complejas de razonamiento automático basándose en métodos formales para expresar tareas a partir de especificaciones en lenguajes formales como *Linear Temporal Logic* (LTL), como por ejemplo en [3, 4].

Los métodos formales son técnicas útiles que surgen del área de estudio de la lógica y son utilizados por desarrolladores de software dado que permiten verificar la consistencia de modelos de sistemas en base a deducciones de las fórmulas lógicas, técnica que se conoce como *model checking* [5]. De manera similar a la *model checking*, surge el concepto de planificación automática, conocido como *planning*. Mientras que el problema de *model checking* busca verificar si un modelo cumple con las especificaciones determinadas (por ejemplo, que el programa nunca se bloquee), el problema de *planning* consiste en encontrar un «controlador» que satisface los requerimientos del modelo [1, 3, 6].

El objetivo del *planning* es encontrar una secuencia de acciones que permitan a un sistema cumplir con su objetivo. Para ello, es necesario contar con una definición del problema a partir de la determinación de cuál es la misión u objetivo del sistema (qué se quiere lograr) y en una abstracción del entorno y de las capacidades del sistema analizado (es decir, qué es lo que el sistema tiene permitido hacer) [7]. Gracias a los métodos formales, es posible realizar estas especificaciones en lenguajes de alto nivel similares al lenguaje natural de manera clara y libre de ambigüedades [8]. Cabe mencionar que el problema de *planning* tiene un estrecho vínculo con el *model checking* por lo que resulta directo pensar que el problema de *planning* puede ser abordado en algunos casos por *model checkers*, como por ejemplo el *Spin* o el *NuMsv* [9–11].

Existen diversas técnicas de modelado del entorno donde un robot opera. Algunas de estas técnicas modelan el entorno a partir de nociones continuas definiendo el entorno donde el robot opera, por ejemplo, como un campo potencial en el que los objetivos se representan con fuerzas atractivas y los obstáculos con fuerzas repulsivas [12], o bien

haciendo un muestreo aleatorio del espacio de trabajo donde se pretende encontrar un camino uniendo las muestras, de manera que el estado inicial y final del robot quede conectado en el espacio de configuraciones evitando las posiciones identificadas como configuraciones «prohibidas» (ver [13]). En este trabajo, la manera por la que se optó describir el espacio de trabajo es a partir de una técnica discreta que consiste en la discretización del entorno mediante celdas, como se verá en el Capítulo 4.

Debido a la técnica de modelado y herramientas que se usarán, las técnicas de *planning* que se desarrollarán en este trabajo forman parte de un «mundo discreto», el cual engloba un modelo abstracto del comportamiento deseado del sistema y un conjunto de requerimientos en cuanto a la interacción entre el sistema y el entorno ([3, 11, 14–16]). Acciones como *visitar una región* o *encender una cámara* implican nociones que «viven» en el «mundo discreto» e introducen la idea de que a los sistemas es posible representarlos en función de estados o eventos discretos, como por ejemplo *región alcanzada* o *cámara encendida*. Sin duda, bajo esta idea, el *planning* se encuentra lejos de la noción física de un robot, que se encuentra en lo que se referirá como el «mundo continuo». Debe tenerse en cuenta que, en lo que se referirá como «mundo continuo», están involucradas variables como nociones de desplazamiento continuo, señales de voltaje en los actuadores y sensores, que en general son procesadas mediante el uso de microcontroladores, que ciertamente no son continuos. Sin embargo, se procederá con la etiqueta de «mundo continuo» para referirse a las capas de más bajo nivel de abstracción del robot y para hacer referencia a nociones distintas que los estados del sistema, propios del *planning*. Con las definiciones anteriores, es necesario introducir un componente adicional que sirva como nexo entre el «mundo continuo» de un sistema robótico y el «mundo discreto» del *planning*. El componente encargado de vincular estos dos «mundos» se conoce como Controlador Híbrido y su tarea es traducir el plan discreto (resultado del *planning*) que cumple los objetivos, en términos de señales de entrada de los actuadores y lazos de control del robot, al mismo tiempo que interpreta las señales de los sensores y las traducen en términos de eventos del plan [3, 8].

1.3. Metodología de Diseño e Implementación

De lo mencionado anteriormente, surge que diversos componentes conforman el proceso de construcción de un robot que sea capaz de planificar misiones automáticamente, algunos de los cuales se basan en especificaciones discretas, otros en especificaciones continuas y otros que relacionan ambos «mundos». En [8] se presenta un enfoque para abordar el desarrollo de este tipo de sistemas distinguiendo y agrupando en tres grandes categorías los componentes que lo conforman según el grado de abstracción que posean. De este modo, el proceso de diseño e implementación de un sistema autónomo basado en controladores híbridos, se reduce al desarrollo de tres capas: Capa de Plani-

ficación, Capa de Control Híbrido y Capa del Robot. La Capa de Planificación hace referencia al proceso de construcción de un modelo abstracto del sistema y de la generación de los controladores de eventos discretos o Autómata. La Capa de Control Híbrido es la encargada de traducir los eventos discretos en señales continuas requeridas para los lazos de control y/o actuadores. Por ejemplo, la acción *avanzar* del controlador de eventos discretos debería ser interpretada de manera que el controlador híbrido genera las señales adecuadas en los actuadores encargados del movimiento del sistema para que este avance. Del mismo modo, la Capa de Control Híbrido recibe señales de los sensores y los traducen en eventos del controlador, como sería el caso de un voltímetro incorporado en el robot que indique una lectura de 1.1 V, la cual podría ser traducida por la Capa de Control Híbrido como el evento *low_battery*, para indicar que la batería del robot se está agotando. Finalmente, la Capa del Robot incluye el conjunto de actuadores, sensores y lazos de control que conforman el sistema robótico de más bajo nivel.

Numerosas contribuciones a nivel experimental fueron realizadas para abordar el problema de navegación con la herramienta de *planning* [3, 4, 8, 11, 17] y, en general, el enfoque de estos trabajos supone partir de un sistema robótico funcional y luego determinar qué misiones sería capaz de realizar el robot dadas sus funcionalidades. Esta metodología supone una limitación para el rango de misiones del robot dado que se encuentra fuertemente ligado a las capacidades de partida del robot y a veces resulta necesario recurrir a costosas técnicas de planificación para acondicionar el entorno a las capacidades del robot [18]. Este trabajo propone una metodología que consiste en realizar el camino inverso, es decir, si uno fuera a diseñar un sistema robótico para realizar un conjunto de tareas predeterminadas, se muestran los pasos a seguir para el proceso de diseño e implementación de un robot autónomo basado en controladores híbridos, haciendo uso de una herramienta de síntesis para resolver el problema de *planning* y generar automáticamente controladores «correctos por construcción».

1.4. Organización del Texto

Este trabajo de tesis está compuesto de diez capítulos. En el Capítulo 2 se abordan algunos conceptos teóricos que serán de utilidad para comprender el lenguaje específico y técnico empleado a lo largo del proyecto. El Capítulo 3 describe la metodología de trabajo seguida para el desarrollo integral de un sistema robótico autónomo basado en controladores híbridos. En el Capítulo 4, se trata el proceso de construcción de un modelo abstracto del sistema y del entorno, así como la manera de especificar misiones del robot en un lenguaje de alto nivel. Los Capítulos 5 y 8 hacen referencia al diseño e implementación de los módulos híbridos, respectivamente, siguiendo con el orden que sugiere la metodología de diseño detallada en en el Capítulo 3. Las etapas de diseño

e implementación del hardware (actuadores y sensores) del robot se abordan en el Capítulo 6 y, en el Capítulo 7, se presentan los lazos de control y de realimentación del hardware necesarios para comandar el robot. Finalmente, en el Capítulo 9 se muestran los resultados experimentales y la validación del proceso de diseño del robot y, en el Capítulo 10, se exponen las conclusiones principales del proyecto.

Capítulo 2

Marco Teórico

> *"Si tuviera ocho horas para cortar un árbol, pasaría las primeras seis afilando el hacha".*
>
> — Abraham Lincoln, político estadounidense y 16.° presidente de EE.UU.

En este capítulo se introducen los conceptos teóricos básicos del proyecto que ayudarán a la comprensión del proceso de diseño e implementación del sistema robótico que se propone construir.

2.1. Lenguajes de Alto Nivel

Los lenguajes de alto nivel surgieron en la década del 1950 como una herramienta útil para superar las barreras ligadas al conocimiento de la programación de bajo nivel. Estos lenguajes permiten una máxima flexibilidad al programador a la hora de abstraerse o de ser literal, y permiten un camino bidireccional entre el lenguaje máquina y una expresión casi coloquial entre la escritura del programa y su posterior compilación. Los lenguajes de alto nivel (como por ejemplo Java, PHP o Python), en lugar de tratar con registros, direcciones de memoria y las pilas de llamadas propios de la programación de bajo nivel (como por ejemplo con el lenguaje Assembler), se refieren a variables, matrices, objetos, funciones y otros conceptos de informática abstracta, lo que permite trabajar a un nivel de abstracción tal que ayuda a facilitar la tarea del programador. La ventaja principal de utilizar lenguajes de alto nivel se encuentra en la capacidad de integrar esquemas de distintos grados de abstracción a través de un lenguaje relativamente sencillo e intuitivo.

Supóngase un manipulador cuya tarea es separar y apilar cajas por su color, siendo las cajas indistinguibles en forma y tamaño. Esta tarea sería relativamente sencilla para un humano, quien solo necesita identificar los colores y poder alinear las cajas

para que, al apilarlas, estas no se caigan. No obstante, si se quisiera automatizar esta tarea para que un robot la realice, se encontraría con las interrogantes: *¿Cómo sabe el robot si se trata de una caja y no de elementos del entorno?* y *¿Si es la caja que debe manipular, cuál es la secuencia de acciones que debe realizar para lograr el objetivo?* Ciertamente, lo que para las personas es una tarea que requiere un corto periodo de entrenamiento, desarrollar un robot con estas mismas capacidades no es trivial e implica muchas más horas de trabajo sin mencionar todo el conocimiento de programación que el desarrollador debe poseer.

El ejemplo anterior ilustra el escenario clásico de la automatización de procesos, en los que se cuenta con un robot o manipulador, se define una tarea a realizar (clasificar y organizar cajas), se diseña eventualmente alguna herramienta que sea necesaria para llevar a cabo la tarea deseada y luego se programa el robot para tal fin. Los lenguajes de alto nivel sirven para facilitar la especificación de tareas al interpretar abstracciones complejas que se parecen más a las ideas que se pueden expresar en un lenguaje coloquial, alivianando el trabajo del desarrollador y reduciendo los tiempos de implementación. Para este escenario, se simplificaría mucho la programación del manipulador si se incorporase una librería, implementada por ejemplo en C, que comunica y comanda el movimiento de los motores, un algoritmo de control de velocidad y/o de movimiento (implementado quizás en otro lenguaje), y, mediante un lenguaje de mayor grado de abstracción por ejemplo Python, resolver el problema de cómo mover las cajas en la secuencia correcta para cumplir el objetivo llamando a las librerías correspondientes.

Si uno tomara el robot del ejemplo anterior y quisiera un escenario más complejo, se podría pensar en tareas como:

> «*separar cajas por su color y llevarlas a una linea de ensamblaje evitando colisionar con otros manipuladores que se encuentran operando cerca*».

Es claro que esta nueva especificación requerirá la reprogramación de varios de los componentes del escenario anterior. En particular, si uno programa de forma modular creando librerías para implementar tareas puntuales que deben realizarse (como el comando de los motores o las comunicaciones entre los sensores), es probable que gran parte del código pueda ser reutilizado y solo serán necesarias modificaciones en las partes de más alto nivel encargadas de la coordinación de los módulos. Los métodos formales ayudan a poder trabajar con estos escenarios de alto nivel y proveen de herramientas y técnicas para facilitar la implementación, verificando que estas implementaciones sean correctas en función de lo que especifica el usuario que quiere que haga su programa.

2.2. Métodos Formales

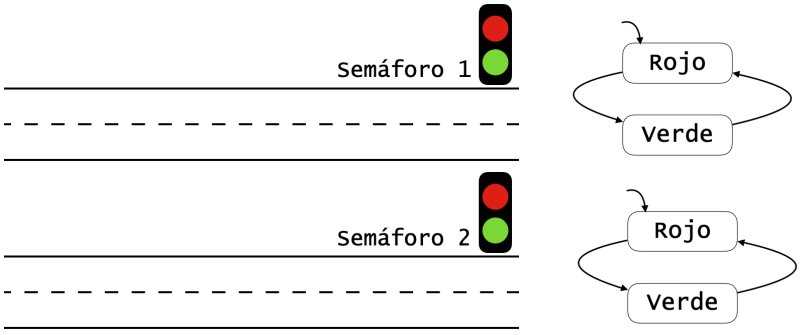
En las Ciencias de la Computación, un **método formal** comprende un conjunto de técnicas basadas en rigurosos modelos matemáticos para la especificación, desarrollo y verificación de sistemas de software o hardware [19]. El término «rigurosos modelos matemáticos» implica que las especificaciones utilizadas por los métodos formales son proposiciones expresadas en una lógica matemática y que la verificación formal se reduce a rigurosas deducciones en esta lógica, lo que hace que los diseños basados en estos métodos sean confiables y robustos. Los métodos formales permiten plantear de forma clara la especificación de un sistema, generando modelos que definen el comportamiento en términos del «qué debe hacer» y no del «cómo lo hace». Entre las formas más tradicionales de representación de un sistema se encuentran los lenguajes basados en modelos y estados, especificaciones algebraicas o especificación de comportamiento. En el presente trabajo se harán uso de las técnicas de modelado de sistemas a través de *Labelled Transition Systems* (Sistemas de Transición con Etiquetas) para definir un modelo abstracto y de alto nivel del robot y su entorno, y *Fluent Linear Temporal Logic* para la especificación de las misiones como propiedades lógicas a satisfacer.

2.3. *Labelled Transition Systems*

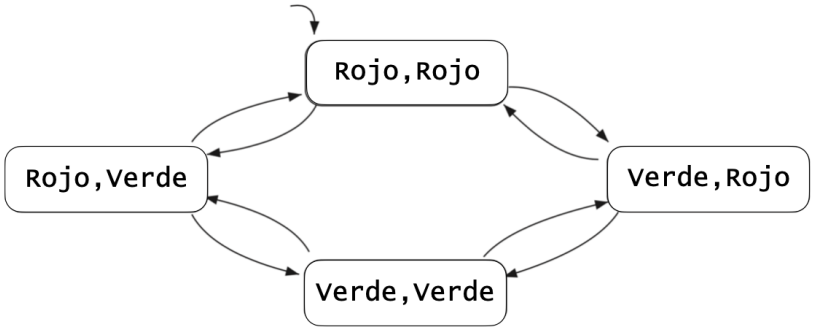
Los sistemas de transición de estados, TS por sus siglas en inglés, son herramientas frecuentemente empleadas en Ciencias de la Computación para modelar el comportamiento de los sistemas a partir de un conjunto de estados y de relaciones entre los estados, denominadas transiciones [5, 16]. El estudio de los TS puede abordarse desde la Teoría de Grafos, donde los estados juegan el rol de los nodos mientras que las transiciones se representan por aristas (o flechas). Siguiendo la nomenclatura de distintos autores ([5, 16]), se denominarán de manera indistinta a los estados del sistema como *proposiciones atómicas* (AP) y a las transiciones como *acciones* (Act). Cuando se tiene más de un estado, se colocan «etiquetas» (o *labels*) a las transiciones y los TS se denominan *labelled transition systems* (LTS). La idea de usar LTS es que se pueden representar esquemas complejos a partir de LTS más sencillos y componerlos bajo la operación de composición paralela ($\|$) [20].

Intuitivamente, cuando los LTS son independientes entre sí, el operador de composición paralela ($\|$) lo que hace es generar todas las posibles combinaciones de los estados de los LTS de modo que lo que se tiene es un nuevo LTS que contempla todas las configuraciones que el sistema puede tomar. Para entender mejor esto, supóngase un sistema conformado por dos semáforos independientes que controlan, por ejemplo, dos rutas paralelas, como se muestra en la Fig. 2.1a. Cada semáforo tiene dos estados posibles: rojo o verde y los LTS que modelan el sistema se ven en la parte derecha

de la Fig. 2.1a. El LTS resultante de la composición paralela entre ambos sistemas independientes se muestra en la Fig. 2.1b, donde se ven las posibles configuraciones que puede tomar el sistema de dos semáforos: semáforo 1 y 2 en rojo, semáforo 1 y 2 en verde, semáforo 1 en rojo y 2 en verde y semáforo 1 en verde y 2 en rojo [5].



(a) Semáforos independientes y sus LTS



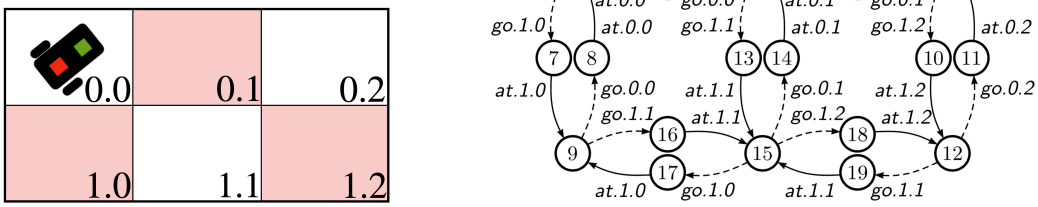
(b) Composición paralela de los LTS de los semáforos

Figura 2.1: Composición paralela de un sistema de dos semáforos independientes

En este trabajo, se distinguen dos clases de eventos en los LTS: **controlables** y **no controlables**. Por un lado, las acciones o eventos **controlables** definen las capacidades del robot como por ejemplo *ir a una región* o *encender un LED*. Por otra parte, los eventos **no controlables** son aquellos que no dependen de las capacidades del robot y describen la interacción del robot con su entorno, como por ejemplo los eventos *llegada a una región* o *batería baja del robot*. Mediante las acciones controlables y no controlables se describirán las asunciones realizadas sobre el entorno y se especificarán los requerimientos, como se detalla en el Capítulo 4.

Tómese por ejemplo un tablero de seis casilleros y un robot que se puede mover en él como se muestra en la Fig. 2.2a. Se llama *go.i.j* a la acción controlable que comanda al robot a desplazarse hacia el casillero *(i,j)* y *at.i.j* a la acción no controlable que ocurre cuando el robot ingresa en la locación *(i,j)*. Si además se supone que el robot únicamente cuenta con la capacidad de moverse desde su posición hacia alguno de los casilleros adyacentes, este escenario puede modelarse a través del LTS mostrado en la Fig. 2.2b, donde las líneas de punto se refieren a las acciones controlables y las líneas continuas a las no controlables. Cabe mencionar que la acción *at.i.j* puede modelarse

como un evento no controlable en tanto que este es el resultado de la acción *go.i.j* del robot y se relaciona con la posición que tiene el robot en un instante dado. Es decir, el robot podría responder a la orden *go.i.j* del controlador de eventos discretos activando los motores para alcanzar un punto específico del espacio, pero si el control de posición no es adecuado o el entorno juega en contra (por ejemplo las ruedas patinan) podría nunca ocurrir un evento *at.i.j* ya que el robot no llegará nunca a la posición *(i,j)*. De aquí surge que la acción *at.i.j* es un evento monitoreado del entorno y por lo tanto no es controlable.



(a) Tablero de seis casilleros **(b)** Representación gráfica de un LTS

Figura 2.2: Ejemplo de un LTS

2.4. *Fluent Linear Temporal Logic*

En la Sección 2.3 se comentó acerca de los sistemas de transición de estados (LTS) los cuales se pueden usar para describir un sistema en término de eventos discretos que hacen evolucionar al sistema de un estado a otro. Una manera de representar especificaciones que describen qué es lo que uno quiere lograr (cuál es el objetivo del robot en el entorno) es mediante el concepto de *lógica temporal*, más conocido como *temporal logic*. Este concepto, tomado del campo de la lógica, hace referencia a cualquier sistema de reglas y símbolos para representar proposiciones que permitan razonar sobre eventos temporales. En particular en este trabajo, se hace uso del lenguaje formal de alto nivel *Fluent Linear Temporal Logic* (FLTL) [21], el cual es una variante de la lógica temporal lineal (LTL) que usa *fluents* para describir los estados a partir de secuencias de acciones. Un *fluent* fl está definido por un conjunto de eventos que hacen que sea verdadero ($Set_\top$), un conjunto que lo hace falso ($Set_\bot$) y una condición inicial (v), que puede ser verdadera ($\top$) o falsa ($\bot$): $fl = \langle Set_\top, Set_\bot, v\rangle$ [16]. Para simplificar la notación, se definirá el *fluent* fl asociado a un evento ℓ de la siguiente manera: $fl = \langle \ell, Act \setminus \{\ell\}, \bot\rangle$.

El FLTL es un formalismo conveniente para especificar y verificar propiedades en sistemas reactivos [22]. El objetivo de utilizar el lenguaje FLTL consiste en poder expresar de forma clara, libre de ambigüedades y en un lenguaje de alto nivel los

requerimientos que se desea que un sistema cumpla. Esto puede ser usado como *input* (junto con el modelo del sistema en LTS) para aplicar la técnica de *model checking* o como se verá en la próxima sección como entrada para *planning*. Los operadores básicos del FLTL comprenden los operadores de la lógica de primer orden $(\wedge, \vee, \neg)$ y se le suman los siguientes operadores para referirse al tiempo: $(\Box)$ siempre, $(\Diamond)$ eventualmente y $(\mathbf{W})$ hasta [5].

Para ejemplificar la idea anterior, una especificación en FLTL aplicable al escenario de la Fig. 2.2 podría ser una misión en la que el robot deba patrullar entre las locaciones 0.0 y 1.2 evitando en todo momento la celda 1.1. Para poder hablar del estado del robot se introducen los *fluents*:

$$\begin{aligned} En00 &= \langle at.0.0, \{at.0.1, at.0.2, at.1.0, at.1.1, at.1.2\}, \top \rangle \\ En12 &= \langle at.1.2, \{at.0.0, at.0.1, at.0.2, at.1.0, at.1.1\}, \bot \rangle \\ En11 &= \langle at.1.1, \{at.0.0, at.0.1, at.0.2, at.1.0, at.1.2\}, \bot \rangle \end{aligned} \tag{2.1}$$

El *fluent* denominado $En00$ tiene el valor inicial *Verdadero* dado que el robot se encuentra en esa posición inicialmente, mientras que los *fluents* $En12$ y $En11$ se activarán solo cuando ocurra el evento $at.1.2$ y $at.1.1$, respectivamente, y tomarán el valor *Falso* en cualquier otro caso. Con estos *fluents* se puede expresar la propiedad lógica que representa la especificación del problema de patrullaje. Una manera de expresar la misión de patrullaje es mediante la condición temporal de «siempre eventualmente» lograr el objetivo. Esto da una idea de que siempre en algún momento del futuro deberá ocurrir un evento, por ejemplo el evento $at.1.2$. Por otro lado, para especificar la restricción para prohibir la celda 1.1, podría pensarse en términos de la lógica temporal lineal como «nunca llegar a la celda 1.1», que, haciendo uso de la simbología adecuada, se escribe como «siempre *no* llegar a la celda 1.1». La misión que implica el patrullaje del robot por los casilleros 0.0 y 1.2 y la prohibición de la celda 1.1 se puede escribir en términos de los *fluents* de la ecuación 2.1 como:

$$\varphi = \Box(\Diamond En00) \wedge \Box(\Diamond En12) \wedge \Box(\neg En11). \tag{2.2}$$

2.5. Problema de *Planning*

En la Sección 2.2, se comentó acerca de los lenguajes formales y de las ventajas que implica su uso para razonar sobre sistemas complejos y modelarlos con LTS. En lo que sigue, se presentan las nociones básicas de la técnica de *planning* que se utilizó para este trabajo. Debe recordarse que esta herramienta deriva de un extenso formalismo que involucra los métodos formales y otras disciplinas, por lo que para un enfoque más detallado con teoremas se sugiere consultar [16].

El problema de *planning* se trata de la búsqueda de estrategias que permitan y aseguren que un sistema lleve a cabo su misión, dentro de un contexto en el que se cumplen ciertas suposiciones o asunciones, o bien determinar que no existe tal estrategia. En este trabajo, para encontrar o sintetizar planes discretos que permitirán tener éxito en la misión, se realizará una descripción del sistema mediante un modelo en LTS, el cual estará conformado por acciones controlables y no controlables, mientras que la descripción de qué es lo que se quiere lograr, es decir cuál es la misión, se especificará haciendo uso de *fluents* (FLTL), según se comentó en los párrafos anteriores. El modelo en LTS y la especificación en FLTL serán entonces las entradas para la herramienta computacional MTSA (*The Modal Transition System Analyser*) [23], la cual será la encargada de obtener los controladores discretos que garantizan (por construcción) el cumplimiento de las propiedades que se especificaron, o bien determinar que no es objetivo realizable.

El problema de *planning* resulta ser más costoso que el de *model checking*, en general el costo crece según $2-EXPTIME$ (es decir, 2^{2^n}, donde n es el tamaño del problema entendido como el tamaño del modelo más el de las especificaciones). Sin embargo, para un caso particular de especificaciones de la forma *Generalized Reactivity 1* (GR(1)), existen algoritmos polinomiales (en costo) que lo resuelven [24]. Las especificaciones en GR(1) consisten en especificaciones de ***safety*** y de ***liveness***. Brevemente, las especificaciones de ***safety*** se utilizan para indicar eventos que se espera que siempre ocurran, o bien, que nunca ocurran, como es el caso para prohibir una región («nunca llegar a la celda 1.1» en el ejemplo de la Fig. 2.2a). Por su parte, las especificaciones de ***liveness*** hacen referencia a eventos que se espera que ocurran «siempre eventualmente». Estas últimas se utilizan en general para especificar la misión del robot como por ejemplo «siempre eventualmente visitar la celda 1.2», para especificar la tarea de patrullaje en el caso de la Fig. 2.2a. En este trabajo se hace uso de las especificaciones de tipo GR(1) para trabajar con problemas cuyo costo computacional crece polinomialmente con la complejidad del modelo en LTS.

Capítulo 3

Metodología del Diseño e Implementación

> *"Divide las dificultades que examinas en tantas partes como sea posible para su mejor solución".*
>
> — René Descartes, filósofo y científico francés

El proceso de diseño e implementación de un sistema robótico basado en controladores híbridos ha sido abordado por distintos trabajos [3, 4, 8, 11, 17], integrando los componentes desde la Capa de Planificación hasta el desarrollo del hardware. El enfoque tradicional de este proceso parte de contar con una plataforma robótica parcial o totalmente funcional, tanto a nivel software como de hardware, lo que ayuda a reducir en gran medida los tiempos de implementación vinculados al desarrollo de una plataforma robótica con sus respectivos sistemas de control. En este escenario, el problema del desarrollo de un robot basado en controladores híbridos se reduce a encontrar un modelo abstracto para la capa de especificación, teniendo en cuenta las capacidades del robot y el espacio de trabajo, que sea compatible con el rango de aplicaciones que se propone realizar [6]. El desafío principal de este enfoque, consiste en desarrollar un modelo cuyo espacio de estados posibles no sea tan grande de manera que el problema no se torne inmanejable para los recursos computacionales, pero que no sea tan simplificado para no perder nivel de detalle causando que el plan falle por una descripción deficiente del sistema. Los problemas de síntesis que se intentarán abordar en este trabajo contendrán especificaciones de la forma GR(1) por lo que su complejidad crecerá polinomialmente con el número de estados posibles [25].

Este trabajo presenta un enfoque alternativo. Se propone determinar en una primera instancia el rango de misiones que el robot realizará definiendo la capa de mayor grado de abstracción (Capa de Planificación), y luego continuar hacia las capas subsiguientes implementando los componentes necesarios para obtener finalmente un

robot operativo que garantizará contar con las capacidades para llevar a cabo las tareas inicialmente propuestas.

3.1. Arquitectura de los Componentes

Para el proceso de diseño e implementación integral del robot basado en controladores híbridos que se propone construir en este trabajo, es necesario primero definir una metodología de trabajo que permita sistematizar tareas de diseño del robot tanto a nivel software como hardware. El sistema robótico resultante garantizará que cuenta con las funcionalidades necesarias para llevar a cabo el rango de tareas especificadas en la Capa de Planificación mediante la traducción de un plan discreto de alto nivel a los sistemas de bajo nivel.

En la Fig. 3.1 se muestran los componentes que forman parte del proceso completo de diseño del robot que se construirá. Como puede verse en esta figura, se hace una primera distinción de las etapas de proceso de diseño en tres grupos según el grado de abstracción de los componentes [8, 11]. En el nivel de abstracción más alto se encuentra la Capa de Planificación, en la que se toma un modelo abstracto del sistema y se sintetiza el controlador de eventos discretos (o Autómata), luego una Capa de Control Híbrido, que traduce las especificaciones del plan generado en las referencias y comandos para los lazos de control y actuadores, respectivamente. Finalmente, en el nivel de abstracción más bajo se encuentra la Capa del Robot, conformado por los sensores, actuadores y lazos de control del robot. A continuación se describen los componentes de cada una de estas etapas refiriéndose a tales con una fuente alternativa.

Capa de Planificación

Esta capa toma un conjunto de Asunciones del entorno, una descripción del Espacio de Trabajo y las Especificaciones del Usuario para construir un plan que satisfaga el modelo del sistema y que verifique las propiedades que en él deben cumplirse. Las abstracciones del modelo y la Especificación del Usuario son plasmadas en un lenguaje formal, por ejemplo LTS o FLTL, que será procesado por un algoritmo de síntesis para obtener los controladores. Finalmente, el Enactor interpreta el plan generado y realiza los llamados a los Módulos Híbridos correspondientes de la capa de Control Híbrido para ejecutar las tareas planificadas.

Capa de Control Híbrido

Esta capa se encarga de «traducir» las acciones del controlador discreto de la etapa anterior en términos de las señales de referencia de los lazos de control y de los actuado-

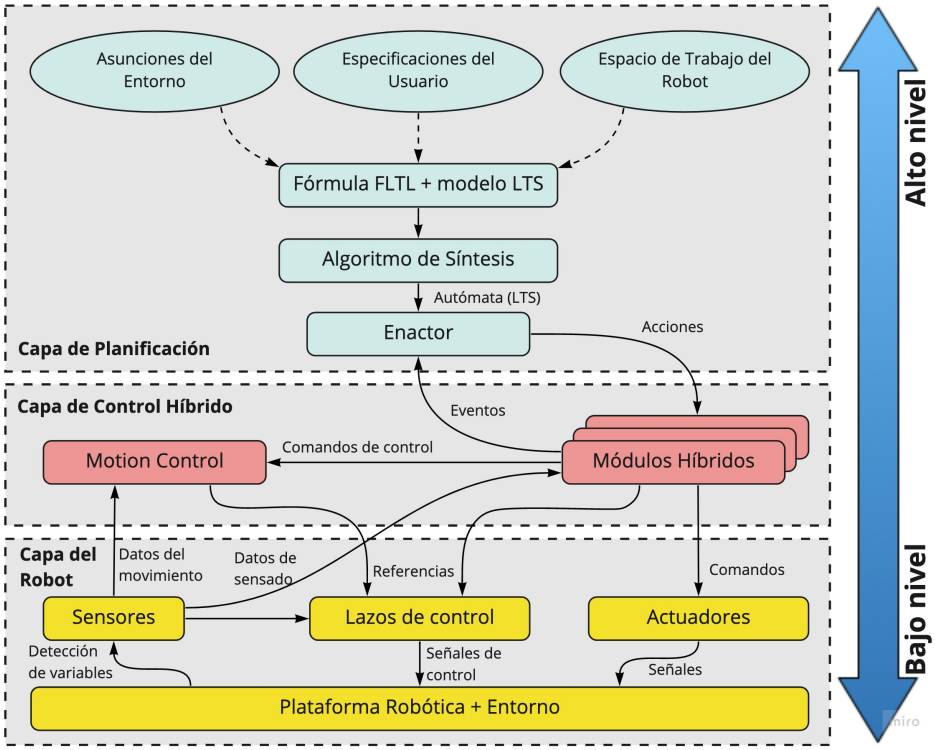


Figura 3.1: Arquitectura de los componentes de un sistema robótico basado en controladores híbridos.

res del robot mediante los Módulos Híbridos. Los Módulos Híbridos también toman señales que provienen de los sensores y las interpretan como eventos no controlables del plan. La determinación de qué Módulos Híbridos deben implementarse en el robot se definen en función de las tareas que el robot debe realizar y los eventos que debe sensar y se detallan en el Capítulo 5.

Capa del Robot

A la capa de más bajo nivel, conocida como Capa del Robot, la integran los Actuadores, Sensores y Lazos de Control que se encargan de detectar el entorno y de ejecutar las acciones que les ordenan los Módulos Híbridos. En esta capa, los Lazos de Control toman como señales de entrada una referencia provista por el correspondiente Módulo Híbrido y la señal proveniente de los sensores para generar la acción de control adecuada sobre los actuadores.

3.2. Metodología de Diseño

La metodología de diseño e implementación utilizada en este trabajo comienza definiendo en una primera etapa del proyecto el rango de actividades a realizar por el robot y, en base a estas especificaciones, desarrollar los componentes necesarios para poder desarrollar el sistema robótico. Esta manera de proceder tiene algunas ventajas,

siendo la más importante el hecho de que el rango de misiones no se encuentra limitado por las propias capacidades del robot, sino que, al suponer como punto de partida las misiones que debe cumplir el robot, se diseñan y construyen los demás elementos de modo que cumplan con las especificaciones iniciales. De esta manera, se pueden construir robots y asegurar que sus capacidades satisfacen los requerimientos de la especificación de la misión.

En la Fig. 3.2 se muestra la secuencia de diseño e implementación seguida para el desarrollo del robot autónomo que se propone construir. En esta secuencia se parte de la capa de mayor grado de abstracción (Capa de Planificación) siguiendo hacia las capas de más bajo nivel. No obstante, la última etapa del proceso consiste en la implementación de la Capa de Control Híbrido, lo que permite flexibilizar el proceso de diseño e implementación de la Capa del Robot. Es preciso notar que la etapa que implica el diseño de los lazos de control debe ser realizada una vez que se cuente con una plataforma robótica funcional con los actuadores y sensores adecuados ya montados.

En las subsecciones siguientes se describirá brevemente en qué consiste cada parte del proceso.

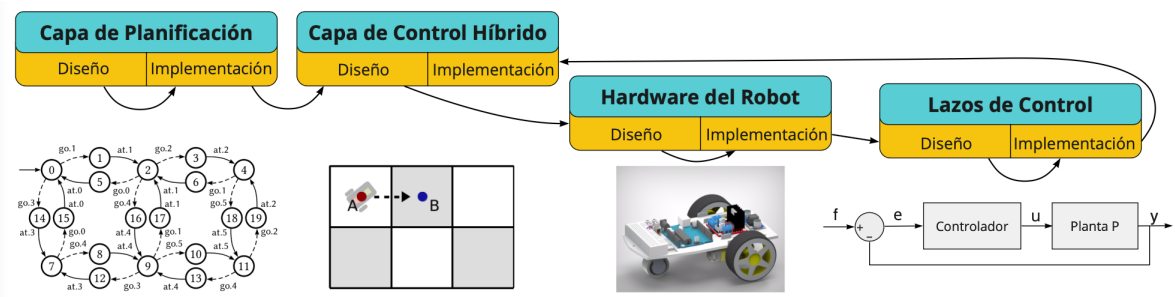


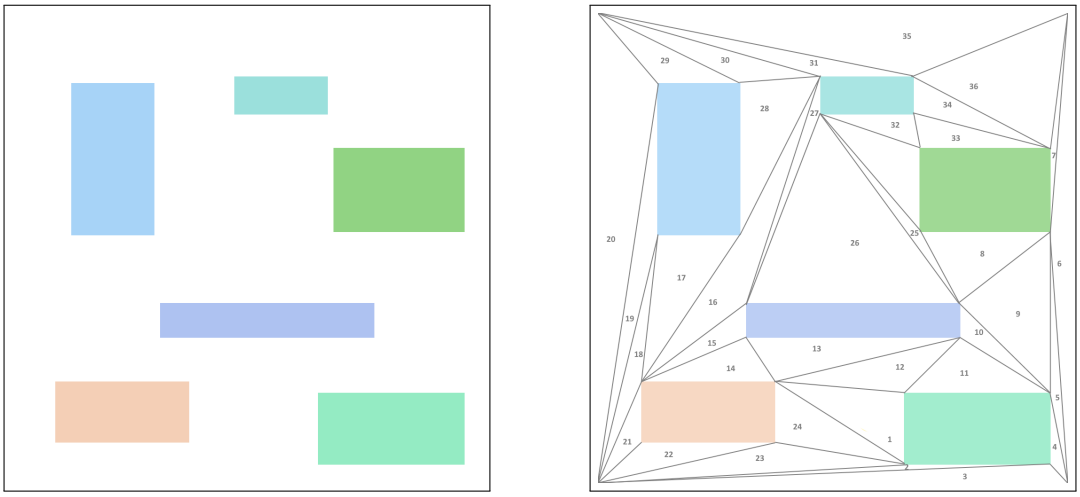
Figura 3.2: Metodología de diseño e implementación del sistema robótico autónomo

3.2.1. Capa de Planificación

El primer paso del proceso de diseño es la especificación del conjunto de tareas a realizar por el robot. En esta etapa, es necesario describir el Espacio de Trabajo del robot y un conjunto de Asunciones en un entorno admisible que sean compatibles con la Especificación del Usuario o misión. Por ejemplo, si se tiene un robot con la capacidad de empujar cajas en un entorno, una Asunción del comportamiento de la caja sería que cada vez que el robot intenta ocupar la celda donde se encuentra la caja, entonces logra empujarla en la dirección de avance del robot. De manera similar, otra Asunción válida sería decir que si la caja está en un entorno acotado y se encuentra en una esquina, entonces el robot no será capaz de moverla.

Para las misiones que implican el desplazamiento del robot en un entorno acotado, es una práctica habitual realizar una abstracción del Espacio de Trabajo basado en la

discretización por celdas. Autores como [11] sugiere que para entornos bidimensionales generales, se haga uso de una división del espacio con celdas triangulares, por ser estos eficientemente computables, mientras que [26] aprovecha la geometría rectangular de un espacio para naturalmente proponer una división por celdas rectangulares. En la Fig. 3.3 se muestra cómo sería la discretización por celdas triangulares en un entorno con obstáculos. Nótese que es conveniente colocar «etiquetas» a las celdas para poder identificarlas más fácilmente.



(a) Entorno original **(b)** Discretización del entorno

Figura 3.3: Discretización del espacio bidimensional en celdas triangulares

Para la implementación de la Capa de Planificación, es necesario condensar los modelos abstractos de las Asunciones realizadas y del Espacio de Trabajo en un LTS, y luego, mediante un algoritmo de síntesis, se generará un controlador de eventos discretos «correctos por construcción» que garantizará el cumplimiento de la misión, al menos en este nivel de abstracción. Múltiples algoritmos de síntesis demostraron ser útiles para trabajar con entornos discretizados por celdas ([4, 26]) por lo que resulta razonable utilizar este enfoque para modelar misiones que impliquen el desplazamiento de un robot dentro de un entorno acotado.

El software de síntesis garantizará que el robot se comportará como se desea sólo si las Asunciones del Entorno permanecen siendo válidas durante la operación del robot. Hay diversos estudios dedicados a modelar sistemas en los que las Asunciones del Entorno varían, como el caso en el que la descripción del sistema real es válido pero ocurre alguna falla en su funcionamiento [16], o en los que dado un conjunto de condiciones que se presentan, la misión del robot cambia [27]. Un ejemplo de esto último sería el caso en que si se detecta que el robot tiene batería baja, la nueva misión del robot cambiaría a «regresar a la estación de carga». En lo que sigue se supondrá que tanto las asunciones del entorno como las especificaciones iniciales permanecen invariantes en todo momento.

El elemento final de esta capa, denominado Enactor, es el encargado de tomar el plan generado por la herramienta de síntesis y realizar los correspondientes llamados a los Módulos Híbridos cada vez que se habilita una acción controlable en la secuencia del plan, como por ejemplo una acción *go.i*, a la vez que queda a la espera de la ocurrencia de eventos no controlables durante la misión, como lo sería un evento *low.battery*, para indicar que la batería del robot se está agotando.

3.2.2. Capa de Control Híbrido (Diseño)

El paso siguiente a la implementación de la Capa de Planificación es el diseño de la Capa de Control Híbrido. El enfoque clásico a la hora de diseñar esta etapa, consiste en trabajar con controladores de bajo nivel y con una estructura de hardware preexistentes y funcionales. Sin embargo, contar con una plataforma de este tipo tiene desventajas a la hora de la implementación de los controladores híbridos. Supóngase que para implementar un comando del movimiento, se decidió utilizar un auto de 4 ruedas. En este caso, sería necesario el cómputo de algoritmos relacionados con el radio mínimo de giro para ir de un punto a otro definiendo trayectorias de Dubins o de Reeds-Shepp [28]. Este tipo de complejidades pudo haber sido evitadas eligiendo de manera adecuada el hardware del robot incluyendo, por ejemplo, ruedas omnidireccionales [29]. Por este motivo es que se propone que antes de continuar con la implementación del hardware del robot, primero se especifique la estrategia de control para el movimiento del robot de acuerdo con la abstracción realizada.

Vale la pena mencionar que ciertas Acciones del plan pueden ser implementadas de manera trivial por los Módulos Híbridos, como por ejemplo una acción *drill.fast* que implicaría establecer una referencia en el controlador de la velocidad de giro de un taladro (por ejemplo a 3000 rpm). Para otras Acciones, los Módulos Híbridos actúan directamente sobre el Actuador, como en el caso de encender un LED o una cámara. No obstante, cuando los módulos híbridos deben realizar tareas más complejas que fijar una referencia en algún lazo de control o generar una señal para activar o desactivar un LED, suelen implementarse bloques de control adicionales. En particular, cuando el robot debe cumplir tareas de navegación, un bloque de control denominado Motion Control será el encargado de llevar a cabo la estrategia de movimiento del robot.

3.2.3. Capa del Robot

En la etapa de más bajo nivel deben seleccionarse los componentes que conformarán la plataforma del robot. En esta etapa se determinan los Actuadores, los Sensores y la fuente de alimentación necesarios para el proyecto de acuerdo a las misiones especificadas en la Capa de Planificación y a las estrategias de control propuestas en

el diseño de la Capa de Control Híbrido. Además, se deben seleccionar las unidades de procesamiento de información como microprocesadores o microcontroladores e implementar las interfaces de comunicación. Por ejemplo, si uno quisiera construir un drone para vigilar un espacio abierto, la Capa del Robot implicaría el diseño de la plataforma robótica, o chasis, y la selección de actuadores (como motores o sistema de iluminación), fuentes de alimentación y de sensores de altitud, posición y orientación (por ejemplo GPS o de inercia) y alguna unidad de procesamiento de datos.

3.2.4. Controladores y Actuadores Directos

Una vez diseñada y construida la plataforma robótica del proyecto es necesario controlarla. Los sistemas de control del robot dependen directamente de la etapa de diseño de los módulos híbridos, debido a que en dicha etapa se diseña la manera en que el robot navegará en el entorno y se prevén las capacidades que se requieren implementar, como por ejemplo activar una alarma. Los controladores de movimiento del robot dependerán de la estrategia implementada en el Motion Control y deberán asegurar que el robot alcance la posición de referencia haciendo uso de la información provista por los sensores del sistema. Esto constituye lo que se conoce como controlador de lazo cerrado, en el que la acción de control se calcula teniendo en cuenta el valor de la referencia y el valor actual. Por ejemplo, si uno quisiera llevar la temperatura de una habitación pequeña a un valor fijo, supóngase $24°C$, contando únicamente con un calefactor, uno podría continuamente medir la temperatura actual de la habitación y ajustar la potencia que el calefactor debe entregar en función de qué tan «lejos» se encuentra la temperatura de la temperatura de referencia.

Por otra parte, en esta etapa se incluye el control de los actuadores directos. Tal como el nombre lo sugiere, estos actuadores no requieren de la implementación de un lazo de control, sino que lo que importa es el valor lógico de estos componentes: encendido o apagado.

3.2.5. Capa de Híbrida (Implementación)

En la etapa final del proceso de diseño se integran todos los componentes para obtener un controlador híbrido completamente funcional. En esta fase, deben implementarse los módulos híbridos y los algoritmos de control de movimientos definidos en la etapa de diseño de la capa híbrida con el hardware y los lazos de control del robot.

Capítulo 4

Capa de Planificación

"Los límites del lenguaje son los límites de la mente".
— Ludwig Wittgenstein, filósofo y matemático inglés

4.1. Problema de *Planning*

Siguiendo la metodología propuesta en la Sección 3.2 para el diseño e implementación de un sistema robótico basado en controladores híbridos, se presenta a continuación la técnica empleada en este trabajo para obtener un modelo abstracto del robot y sintetizar un controlador discreto que satisfaga el modelo del sistema y las especificaciones del usuario. Se hace uso de LTS para describir formalmente las asunciones del entorno y las capacidades del robot y FLTL para determinar el comportamiento esperado por el sistema, es decir, la misión.

4.1.1. Especificación de la Misión

Cuando un usuario se propone a especificar una misión para el robot, inicialmente la tarea se puede expresar en un lenguaje coloquial estructurado como en [8]. Por ejemplo, se podrían especificar misiones como:

visitar la habitación 1 y visitar la habitación 2 o la 3,

en la cual el robot deberá navegar hasta la habitación 1 y, una vez allí, se dirigirá a la habitación 2 o 3. De manera similar, uno podría pensar en misiones que no necesariamente impliquen el desplazamiento del robot, como por ejemplo:

si se está en la habitación 5 entonces reproducir música.

Estos comportamientos basado en eventos discretos, pueden ser expresados en FLTL y luego ser interpretados por la herramienta de síntesis que generará los controladores correspondientes. En este trabajo se hará uso del software *The Modal Transition System*

Analyser (MTSA), desarrollados por grupos de investigación de Ingeniería en Software del *Imperial College London* y de la Universidad de Buenos Aires, para sintetizar las misiones del robot [23].

En [30] se presenta un conjunto de especificaciones aplicables a robots móviles a partir de una recopilación de misiones del campo de la robótica. Tomando a [30] como referencia, en este trabajo se abordarán las misiones denominadas como *Core Movement Patterns* (misiones de patrullaje y visita de regiones), *Avoidance* (evasión de regiones prohibidas) y *Reaction* (por ejemplo, encender una alarma cuando se entra a una región). Como una extensión a las misiones de tipo *Core Movement Patterns*, se propone además la tarea en la cual el robot sea capaz de acomodar cajas en un entorno acotado, como se explicará en secciones posteriores.

4.1.2. Discretización y Modelo del Entorno

El paso siguiente para resolver el problema de *planning* consiste en la descripción del entorno donde operará el robot, según se comentó en la Sección 2.5.

El robot que se propone desarrollar realizará misiones de corta duración (alrededor de 30 min) en una región 2D de aproximadamente (2 m × 3 m). Dada la naturaleza del entorno, y tomando como referencia el trabajo de [26], se optó por utilizar una discretización en celdas rectangulares cuyo tamaño dependerá de las capacidades del robot para maniobrar dentro de una celda antes de pasar a la siguiente sin ocupar una celda que no haya sido prevista. Este tipo de discretizaciones se adelantaron en la Sección 2.3. La asunción más importante del entorno en este modelo es que el robot no podrá pasar de una celda a otra que se encuentra en la diagonal de la celda de partida. Esta restricción sirve para garantizar que el robot no pase por regiones que no estaban previstas en el plan sintetizado. Esto se ve en la Fig. 2.2b donde no existen relaciones de adyacencia entre casilleros en diagonal, es decir, el sistema no puede pasar del nodo 0 (celda *(0,0)*) al 15 (celda *(1,1)*) con solo una acción *go.i.j*.

4.1.3. Misiones de Patrullaje y Detección de Obstáculos

Si al problema de la Fig. 2.2b se le agrega un obstáculo, por ejemplo en la ubicación *(1.1)*, y además se especifica como misión que el robot visite indefinidamente las celdas *(1.0)* y *(0.2)*, el esquema del nuevo entorno del robot se muestra en la Fig. 4.1, donde las locaciones a visitar se indican con las letras A y B y las flechas se refiere a la asunción del entorno realizada referida a que el robot solo puede desplazarse por celdas adyacentes.

La fórmula FLTL que describe este problema se relaciona por un lado con la misión de patrullaje (φ_1) y por otro con el requerimiento de evasión de los obstáculos (φ_2). De esta forma, la fórmula FLTL que modela el sistema puede escribirse como:

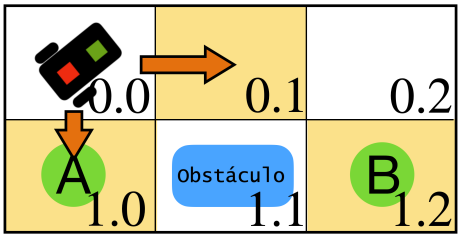


Figura 4.1: Entorno rectangular discretizado por celdas rectangulares con un obstáculo identificado en azul.

$$\begin{aligned} \varphi_1 &= (\square\Diamond \mathit{at.1.0}) \wedge (\square\Diamond \mathit{at.1.2}) \\ \varphi_2 &= (\square\neg \mathit{at.1.1}) \end{aligned} \tag{4.1}$$

El LTS de la Fig. 2.2b y las fórmulas FLTL de la ecuación 4.1 son las entradas de la herramienta de síntesis mediante la cual se generará un controlador discreto que satisfaga tanto el modelo como las propiedades que deben cumplirse, es decir, φ_1 y φ_2. Es interesante notar que el controlador de eventos discretos para el problema de la Fig. 4.1 (ver Fig. 4.2) resulta ser un subespacio del LTS que describe el espacio de trabajo (Fig. 2.2b). De aquí se ve que simplemente mediante la especificación φ_2, que prohíbe que el robot llegue a la locación *(1,1)*, se puede modelar el comportamiento del robot, como lo sería la evasión de obstáculos en este escenario.

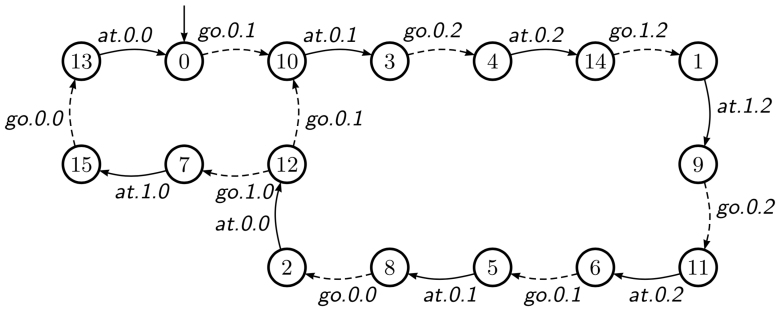


Figura 4.2: Controlador de eventos discretos para la misión de patrullaje en un entorno discretizado en (2×3) celdas con un obstáculo en la locación *(1,1)*

4.1.4. Misiones de Patrullaje y Detección de Obstáculos con Alarma

Otra misión que se propone lograr y que se engloba en el conjunto de misiones definidos como *Reaction* según [30], consiste en la activación de un sistema de alarma sonora y luminosa cuando el robot ingresa a una región. Este escenario podría ser útil en el caso de un robot que opera cerca de personas y, con el fin de proteger tanto al robot como a las personas, es deseable emitir una alerta en dicha zona. Al agregar una capacidad del robot que no se relaciona con el movimiento del robot en el

entorno, es necesario definir un LTS adicional que contemple las acciones de activación y desactivación de la alarma, y luego, mediante la operación de composición paralela ($\|$), se construye el controlador de eventos discretos del problema. El LTS que modela el sistema «alarma» se muestra en la Fig. 4.3. donde las acciones *alert.on* y *alert.off*

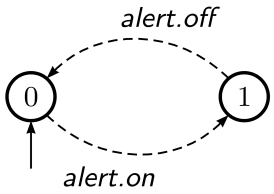


Figura 4.3: Autómata que describe la capacidad del robot de activar y desactivar una alarma

son acciones controlables que llevan al sistema a los estados en los que la alarma está activada y desactivada, respectivamente. Notar que, en el caso de Fig. 4.3, el sistema parte del estado en el que la alarma está apagada y las lineas punteadas indican que son acciones controlables.

Para ilustrar lo anterior, se muestra en la Fig. 4.4 el mismo escenario de la Fig. 4.1 y se distinguen dos zonas. Por un lado está la zona indicada como Zona de Peligro (fila inferior), en la que el robot debe activar la alarma, y por otro lado está el resto del entorno (fila superior). La misión del robot en este caso continua siendo que patrulle entre las locaciones *(1,0)* y *(1,2)*, ambas contenidas en la Zona de peligro.

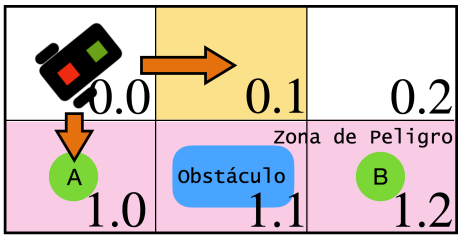


Figura 4.4: Entorno discreto para una misión de patrullaje con activación de alerta

Este escenario puede modelarse con los *fluents* ***AtBot***, que tomará el valor *Verdadero* ($\top$) cuando el robot esté en la fila inferior de la grilla y *Falso* ($\bot$) en otro caso, ***AtTop***, que será *Verdadero* cuando el robot esté en la fila superior de la grilla y *Falso* en otro caso, y el *fluent* ***Alert***, que se activará cuando ocurra la acción *alert.on* y se apagará con *alert.off* (ecuación 4.2).

$$\begin{aligned} AtBot &= \langle \{at.1.0, at.1.1, at.1.1\}, \{go.0.0, go.0.1, go.0.2\}, \bot \rangle \\ AtTop &= \langle \{at.0.0, at.0.1, at.0.2\}, \{go.1.0, go.1.1, go.1.2\}, \top \rangle \\ Alert &= \langle alert.on, alert.off, \bot \rangle \end{aligned} \tag{4.2}$$

Adicionalmente, se define la propiedad φ_3 para indicar que la alarma se activará una vez que el robot entre a la Zona de Peligro, que es equivalente a decir que se activará

cuando el *fluent* *AtBot* tome el valor *Verdadero* (ver ecuación 4.3). Notar que en la expresión de la ecuación 4.3 se introduce el operador lógico temporal ◯ que se lee como «siguiente» e indica el estado sucesor inmediato. Por ejemplo, cuando *AtTop* y *Alert* toman el valor *Verdadero*, es decir el robot está en la fila superior con la alarma encendida, entonces debe ocurrir la acción *alert.off* para apagar la alarma.

$$\begin{aligned} \varphi_1 &= (\square\Diamond \mathit{at.1.0}) \wedge (\square\Diamond \mathit{at.1.2}) \\ \varphi_2 &= (\square\neg \mathit{at.1.1}) \\ \varphi_3 &= \square\Big(\big((\mathit{AtBot} \wedge \neg \mathit{Alert}) \Rightarrow \bigcirc \mathit{alert.on}\big) \wedge \big((\mathit{AtTop} \wedge \mathit{Alert}) \Rightarrow \bigcirc \mathit{alert.off}\big)\Big). \end{aligned} \tag{4.3}$$

4.1.5. Misiones de Ordenamiento de Bloques

La última misión que se propone lograr con el robot es la tarea del ordenamiento de bloques o cajas en el espacio de trabajo, en la cual el robot será capaz de mover la caja empujándola hasta llevarla a la posición deseada. La Fig. 4.5 muestra la secuencia que realizaría el robot para lograr ubicar la caja en la posición deseada (locación *(1,2)*) para el entorno de la Fig. 4.1.

Esta especificación puede considerarse una extensión de la misión de patrullaje en tanto que el movimiento de la caja se produce cuando el robot intenta ocupar la posición actual de la caja. Para lograr esta misión, es necesario variar el modelo del entorno para incorporar la propiedad de que cuando el robot intenta ocupar el lugar de la caja, esta se mueva a la posición correspondiente. Tal como se describe el movimiento de la caja, es claro que el robot no contará con una capacidad llamada «mover la caja», sino que el desplazamiento de la caja es el resultado del avance del robot hacia la posición de la caja. Por dicho motivo, para modelar este problema, resulta razonable incorporar una nueva acción no controlable *goBox.i.j*, que represente el movimiento de la caja en el entorno.

Para la tarea del ordenamiento de las cajas en un entorno acotado es necesario realizar algunas variaciones al modelo del entorno para incorporar que cuando el robot intenta ocupar una celda ocupada por la caja, esta se mueve a la posición que le corresponde, como se muestra en la Fig. 4.5. Para esto, se define un LTS que contempla todas las posibles posiciones a las que podría ir la caja y se definen los LTS que permiten identificar cuál sería la posición final de la caja dependiendo desde qué celda se aproxima el robot. Es decir, si el robot se aproxima desde la celda *(1,0)* e intenta ir a la posición de la caja en *(1,1)*, entonces la caja se mueve a la celda *(1,2)*. De igual manera, si la caja se encuentra en una esquina, hay que modelar que el robot no conseguirá mover la caja. Los modelos en LTS que describen el problema de ubicar una caja en una posición determinada no se muestran en este texto dado que resultaron ser extensos, por lo que se adjuntan como material complementario en los archivos

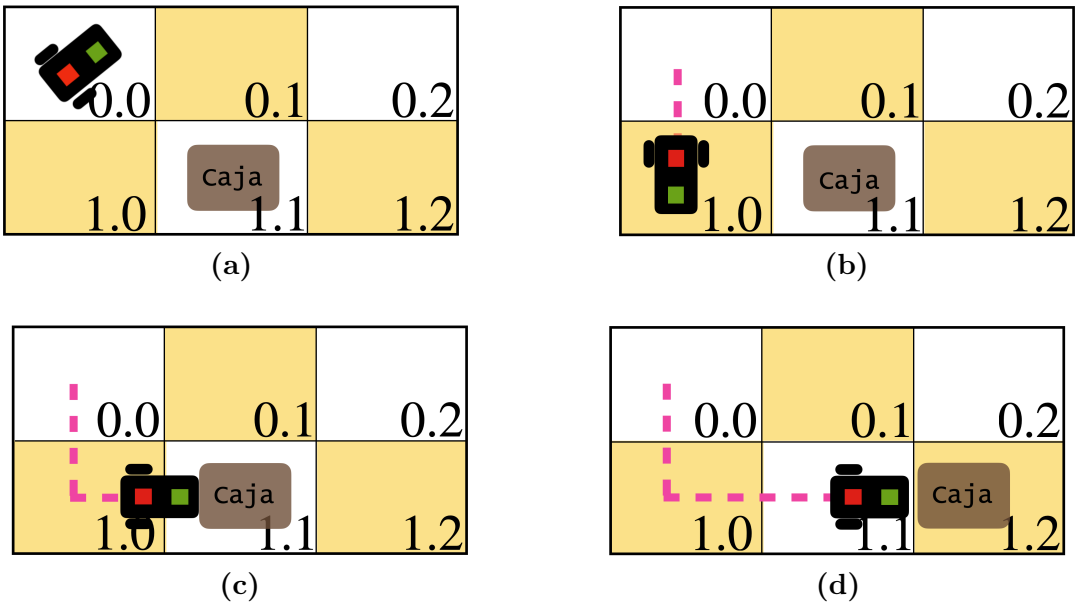


Figura 4.5: Secuencia de ordenamiento de una caja en un entorno discreto

misionCajaSO.lts y *misionCajaCO.lts*. La manera de especificar la misión de «*llevar la caja a la posición (h,k)*» se logra simplemente mediante la definición de una propiedad φ_4 como:

$$\varphi_4 = \Box(\Diamond goBox.h.k). \tag{4.4}$$

4.2. Controladores de Eventos Discretos

Los controladores de eventos discretos obtenidos mediante la herramienta de síntesis contienen una secuencia con las acciones controlables y no controlables del problema que garantizan el cumplimiento de la misión. En la Fig. 4.6 se presenta un ejemplo de un controlador para el problema del ordenamiento de cajas de la Fig. 4.5. Este controlador cuenta con trece estados posibles indicados con una letra «Q» y seguido del número asignado a cada estado. El controlador que se muestra en la Fig. 4.6 maneja una sintaxis un poco distinta a la que se viene manejando, en la que las acciones *go.i.j* se escriben como *go[i][j]*, las acciones *at.i.j* se escriben como *arrived[i][j]* y las acciones *goBox.i.j* se escriben como *goBox[i][j]*.

La interpretación de estos controladores parte de tomar como punto de partida el estado Q0. En este estado, el controlador verifica si tiene alguna acción controlable o no controlable para procesar. A partir del estado Q0 genera la acción controlable *go[1][0]*, lo que lleva al sistema ahora al siguiente estado Q1. En Q1, el controlador debe esperar la ocurrencia del evento no controlable *arrived[1][0]*, para poder pasar al estado Q5, y así sucesivamente. Cabe mencionar que en el caso de la ocurrencia de un evento no previsto en la secuencia del controlador, por ejemplo que suceda el evento *arrived[1][1]* antes del evento *goBox[1][2]* en el estado Q9, o bien que ocurra un evento

```
Process:
        Controller
States:
        13
Transitions:
        Controller = Q0,
        Q0          = (go[1][0] -> Q1),
        Q1          = (arrived[1][0] -> Q5),
        Q2          = (arrived[0][0] -> Q4),
        Q3          = (arrived[0][1] -> Q7),
        Q4          = (go[0][1] -> Q3
                      |go[1][0] -> Q6),
        Q5          = (go[1][1] -> Q9),
        Q6          = (arrived[1][0] -> Q10),
        Q7          = (go[0][0] -> Q2
                      |go[0][2] -> Q8
                      |go[1][1] -> Q11),
        Q8          = (arrived[0][2] -> Q12),
        Q9          = (goBox[1][2] -> Q11),
        Q10         = (go[0][0] -> Q2
                      |go[1][1] -> Q11),
        Q11         = (arrived[1][1] -> Q4),
        Q12         = (go[0][1] -> Q3).
```

Figura 4.6: Ejemplo de un controlador de eventos discretos para el problema de ordenamiento de una caja en un entorno acotado generado con la herramienta de síntesis

que no haya estado modelado en el LTS como lo sería que cuando el robot avanza a la posición *(1,1)* no esté modelado el avance de la caja a la locación *(1,2)*, el Enactor abortará la misión enviando un mensaje de **error** y frenando el movimiento del robot.

El Enactor es el último componente de la Capa de Planificación. Este elemento debe tomar el controlador de eventos discretos y, en función de la acción que continúa en la lista por ejemplo una acción *alert.on*, se hace el llamado al módulo híbrido correspondiente y, viceversa, cuando un módulo híbrido envía una señal de la ocurrencia de un evento no controlable, este es interpretado por el Enactor, el cual da lugar a la siguiente acción de la secuencia del controlador.

Los archivos correspondientes a la implementación de las misiones anteriores en un entorno de $(N \times M)$ celdas y el archivo que implementa el Enactor se adjuntan como material complementario del proyecto.

Capítulo 5

Módulos Híbridos (Diseño)

> *"Es sencillo hacer que las cosas sean complicadas, pero difícil hacer que sean sencillas".*
>
> — Friedrich W. Nietzsche, filósofo y poeta alemán

En este capítulo se describe la etapa de diseño de los módulos híbridos que utilizará el robot para ejecutar las acciones del plan generado en la capa de planificación. Es necesario definir un módulo híbrido por cada funcionalidad del robot y capacidad de sensado, por lo que en total se describirán tres módulos, correspondientes al desplazamiento del robot, la activación de una alarma y la capacidad de ordenamiento de cajas.

5.1. Módulo de Desplazamiento

El primer módulo que se presenta se corresponde con la capacidad de desplazamiento del robot en un entorno acotado discretizado como el de la Fig. 4.1. Este módulo se encargará de establecer una referencia en el controlador de posición del robot, interpretando la coordenada *(i,j)* de la acción *go.i.j* del controlador de eventos discretos como una posición del espacio bidimensional (x, y). Para esto, es necesario que este módulo llame a una función que traduzca la coordenada *(i,j)* en alguna unidad de medida que represente el espacio físico donde opera el robot. Una forma de pensar la relación de una celda con un punto del espacio es suponer que cuando el controlador de eventos discretos da la orden *go.i.j*, entonces el robot se dirija al centro de la celda. De esta manera, las coordenadas de la celda *(i,j)* se traducen como el punto (x_i, y_j) que se ubica en el centro de la celda como se indica en la Fig. 5.1.

Luego de que la acción *go.i.j* indique cuál es la nueva referencia de la posición que debe alcanzar el robot, el mismo debe desplazarse hasta dicha posición garantizando que en ningún momento pase por una región no prevista por el controlador. Esto se

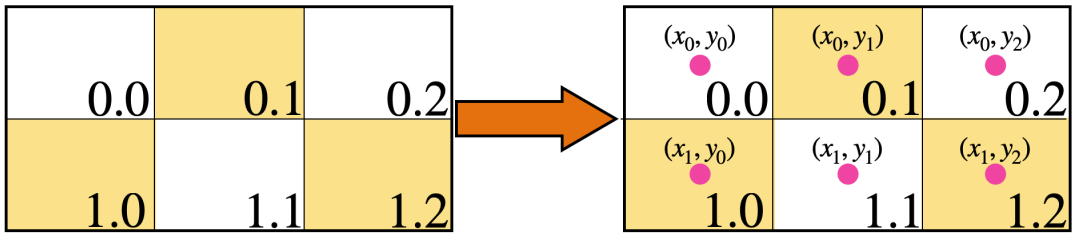


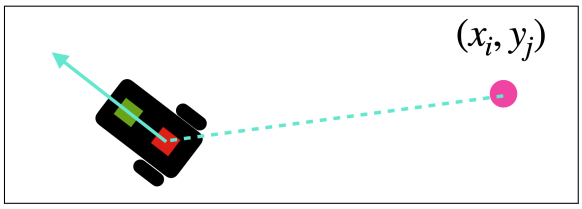
Figura 5.1: Noción física de las coordenadas *(i,j)* del entorno discreto

garantiza por un lado estableciendo las condiciones de adyacencia adecuados, como se indicó en la Sección 4.1.2, y por otra parte, desarrollando una estrategia para el desplazamiento del robot que le permita maniobrar en su paso de una celda a otra sin pasar por una celda que no corresponda. La estrategia de desplazamiento del robot se llevará a cabo mediante un bloque de control adicional al que se denominará *Motion Control*. El bloque *Motion Control* tomará la posición de referencia (x_i, y_j) y la posición actual del robot (x_R, y_R) y llevará a cabo las maniobras que le permiten alcanzar la referencia. En esta etapa, es fundamental determinar un esquema básico de la plataforma robótica, debido a que de esto dependerá la estrategia de movimiento que debe realizar el bloque de *Motion Control*. Para este trabajo se propone utilizar una plataforma robótica con dos ruedas motrices y una tercera rueda que servirá de apoyo, como en [31] y las maniobras que se proponen comprenderán los siguientes pasos:

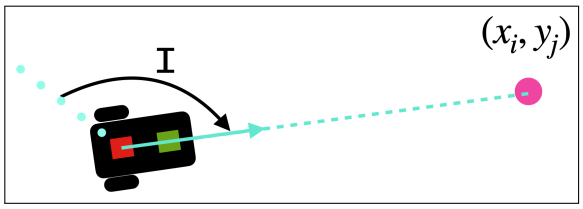
1. orientar el robot hasta quedar alineado con el punto de referencia (x_i, y_j),
2. avanzar en línea recta a velocidad constante hasta alcanzar la posición deseada.

Esta secuencia, que se puede ver esquematizada en la Fig. 5.2, permitirá implementar el módulo correspondiente al desplazamiento del robot de manera sencilla en una primera etapa del proyecto. Notar que de haber optado por un robot de cuatro ruedas, se requeriría el cómputo de algoritmos de trayectoria de Dubins o de Reeds-Shepp [28], complejizando los esquemas de control para esta etapa de prototipado del proyecto. Si por el contrario, se hubieran utilizados ruedas omnidireccionales, no habría resultado necesario implementar ningún bloque de control del movimiento, en tanto que el robot se desplazaría de manera indistinta hacia adelante o hacia un costado y solo habría bastado con establecer una referencia a los lazos de control. Sin embargo, el uso de ruedas omnidireccionales fue descartado en este proyecto dado los elevados costos que implicaba su adquisición.

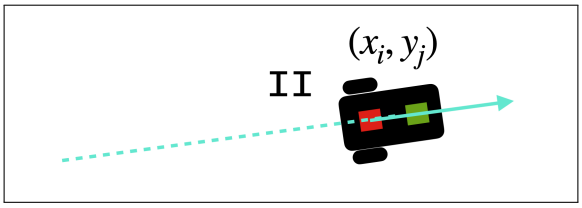
Una vez que el robot llega a la posición (x_i, y_j) de referencia, el módulo híbrido debe enviar el mensaje de la llegada del robot a la posición deseada y agregar el evento de llegada *at.i.j* correspondiente en la cola de acciones del controlador de eventos discretos para que se proceda a la siguiente acción.



(a) Configuración inicial del robot. La flecha verde apunta en el sentido de orientación del robot.



(b) El robot gira hasta alinearse con el objetivo.



(c) El robot avanza hasta estar lo suficientemente cerca del objetivo.

Figura 5.2: Secuencia de movimientos del robot para alcanzar la posición (x_i, y_j)

5.2. Módulo de Alerta

El siguiente módulo necesario en la capa híbrida es el correspondiente a la acción de activar o desactivar una alarma, indicado con las acciones controlables *alert.on* y *alert.off* del autómata. Este módulo es más simple que el módulo anteriormente descripto, en tanto a que actúa directamente sobre el actuador del robot que habilita o deshabilita la alarma y no requiere de un bloque adicional de control como el *Motion Control*. Este módulo, toma de la lista de eventos del controlador de eventos discretos la acción *alert.on* o *alert.off* y genera la señal sobre el actuador correspondiente para activar o desactivar la alarma.

5.3. Módulo de Ordenamiento de Cajas

El último módulo necesario es el encargado de monitorear el evento de arribo de la caja que el robot empuja (evento no controlable) a la celda deseada. Al igual que la asunción hecha de que el robot siempre se encuentra contenido dentro de una celda (salvo en el caso de que se esté trasladando y pase a la celda siguiente) y la condición de adyacencia del entorno, la caja que el robot empuja también debe satisfacer estas consideraciones para respetar el modelo planteado para esta misión en la Sección 4.1.5.

La idea de este módulo es que detecte continuamente la posición de la caja y la referencia que busca alcanzar el robot. Cuando se verifique que el robot se está dirigiendo hacia la celda ocupada por la caja, el módulo se activará y deberá detectar la distancia que hay entre la caja y la posición a la que la caja se dirige, de manera que cuando la caja se encuentre lo suficientemente cerca del objetivo, el módulo indicará

su arribo al ENACTOR mediante la acción *goBox.i.j*.

5.4. Esquema de Control Híbrido

El esquema de funcionamiento de los tres módulos descriptos se muestra en el diagrama de flujo de la Fig. 5.3, donde se ve cómo se organizarían a grandes rasgos los distintos componentes de la capa de control híbrido. En el Capítulo 8 se explican en mayor detalle la manera en que funciona cada módulo híbrido y cómo fueron implementados.

En esta figura se ve que el ENACTOR es el componente encargado de llamar a los distintos módulo híbridos vinculado a las acciones controlables del robot. El módulo de alarma se activa cuando ocurre el evento *alert.on* o *alert.off*, y actúa directamente sobre los actuadores del sistema activando o desactivando el sistema de alarma, como se mencionó en la Sección 5.2. El módulo de desplazamiento se activa frente a una acción *go.i.j* y tiene dos posibilidades. Por un lado, si la referencia que debe alcanzar el robot no coincide con la posición de la caja, entonces se procede al bloque de control *Motion Control:Robot* para generar las referencias en los lazos de control y lograr que el robot alcance la posición *(i,j)*. Cuando el robot llega a la locación *(i,j)* deseada, el módulo retorna el evento *at.i.j* al ENACTOR. Por el contrario, si las coordenadas *(i,j)* de la acción *go.i.j* coinciden con la posición de la caja, el algoritmo entiende que se trata de una acción para mover la caja. De esta manera, se activa el módulo de ordenamiento de cajas, el cual mediante el bloque de *Motion Control:Caja* implementa la lógica para llevar la caja a la posición final. Cuando la caja llega a su destino, se devuelve el evento *goBox.h.k* al autómata, donde *(h,k)* son las nuevas coordenadas de la posición de la caja. Una vez que la caja fue ubicada en la locación indicada *(h,k)*, el robot se dirige hacia la celda *(i,j)* indicada por el ENACTOR inicialmente. Cabe mencionar que la manera en la que cada bloque de *Motion Control* fija la referencia en el lazo de control, dependerá de la implementación de cada módulo y se verá con mayor detalle en el Capítulo 8.

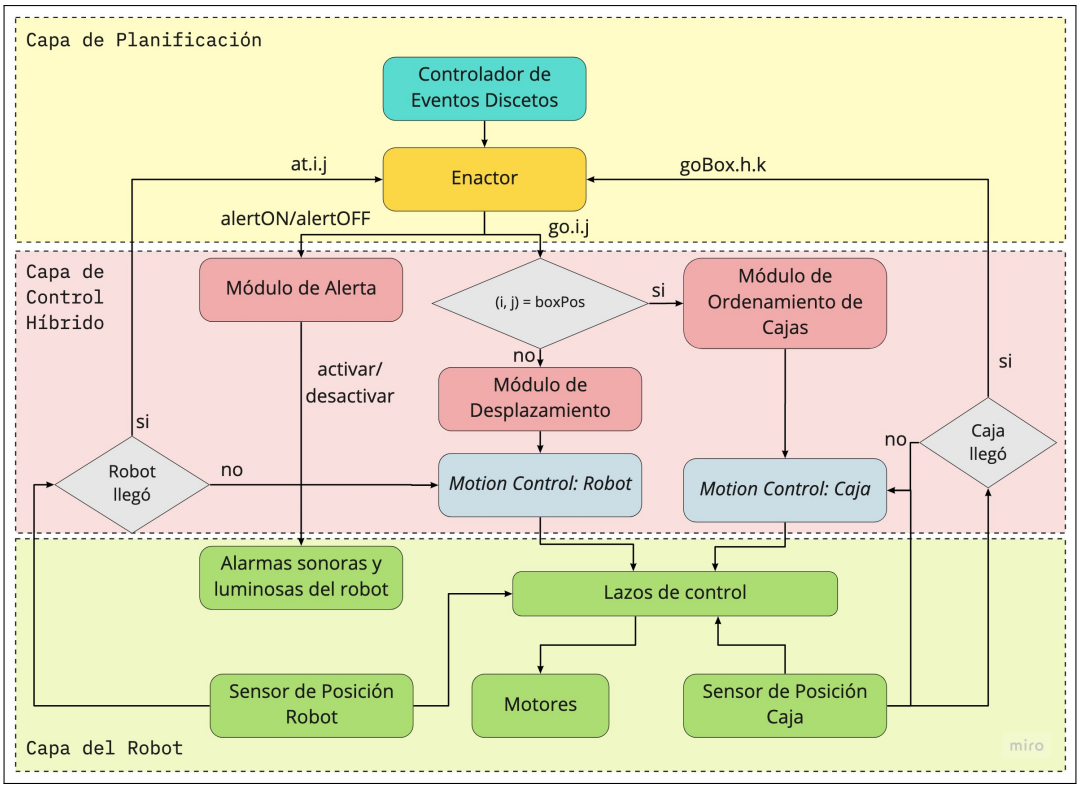


Figura 5.3: Diagrama de flujo del funcionamiento de la capa de control híbrido

Capítulo 6

Hardware y Sensores

> *"Miré a mi alrededor y no encontré el auto de mis sueños por lo que decidí construirlo yo mismo".*
>
> — Ferdinand Porsche, ingeniero alemán y fundador de Porsche

En este capítulo se analizan los componentes de más bajo nivel necesarios para el desarrollo del robot basado en controladores híbridos, el cual fue denominado Tom. Aquí se incluye la selección de los circuitos de alimentación, actuadores, sensores y componentes para el procesamiento de datos. El diseño del chasis y carrocería del robot se realizó utilizando el software Solid Edge 2019 Student Version.

6.1. Montaje de Pruebas

Para realizar las pruebas del robot a desarrollar es menester contar con una idea de los subsistemas que conforman el proyecto. Por este motivo, se presenta a continuación un esquema del montaje que permitirá determinar los componentes necesarios para la construcción del robot y con el cual se harán los ensayos para verificar el correcto funcionamiento del sistema. Para mayor claridad, se referirá a los elementos que constituyen el sistema con una FUENTE ALTERNATIVA.

En la Fig. 6.1 se muestra un esquema del montaje de pruebas propuesto para realizar los ensayos del robot. Tal como puede verse en esta figura, una cámara web, a la que se referirá como WEBCAM, será utilizada como sensor para detectar la posición y orientación del robot y los obstáculos en el entorno. Para poder determinar la posición y orientación del robot, se colocará encima de la plataforma robótica dos elementos distintivos (rojo y verde), con los cuales se identificarán las partes delantera y trasera del robot que permitirán estimar su posición y orientación mediante el método de procesamiento de imágenes que se detalla en la Sección 6.2. La WEBCAM se conectará a un

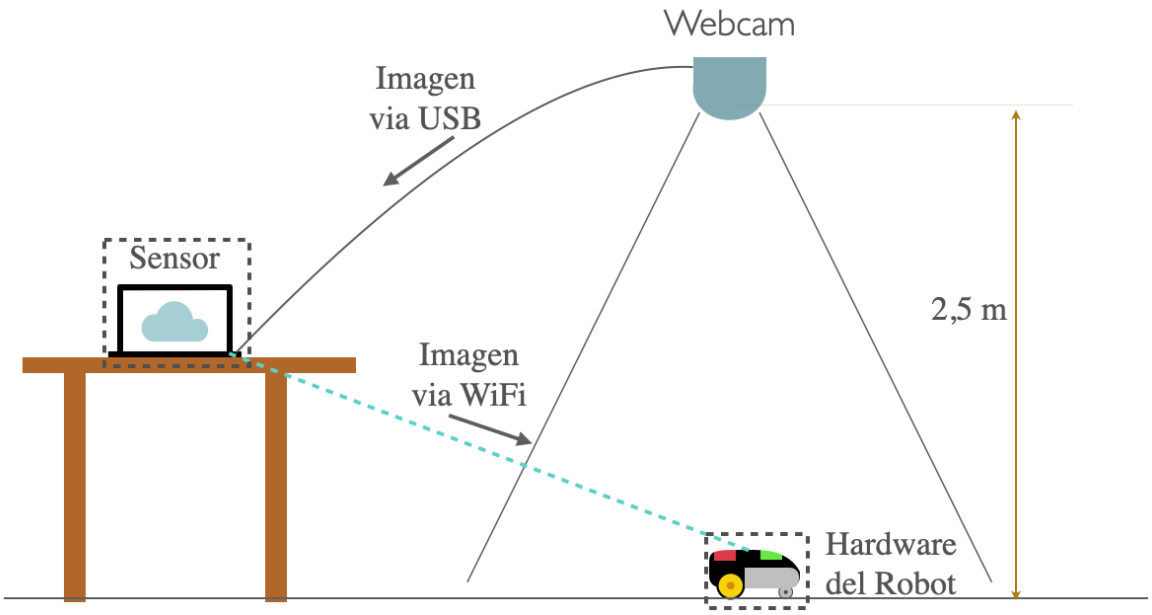


Figura 6.1: Esquema del montaje de pruebas del robot

ordenador, LAPTOP, por medio de un cable USB y se denominará al conjunto WEBCAM +LAPTOP como SENSOR. El SENSOR capturará las imágenes que se toman con la WEBCAM y las enviará al ROBOT mediante conexión WiFi. El ROBOT, mediante una placa Raspberry Pi montada en la plataforma robótica, recibirá las imágenes provenientes del SENSOR, a una frecuencia medida de 12 imágenes por segundo (12 FPS), las procesará y ejecutará el algoritmo de control que permite al robot cumplir la especificación solicitada, tal como se indica en la Sección 5.1. La idea de hacer un procesamiento de imágenes *onboard* permite tener la libertad de que, en un trabajo futuro, pueda independizarse del conjunto WEBCAM +LAPTOP, que ciertamente tiene una capacidad de procesamiento de imágenes mayor que la Raspberry de la plataforma robótica, y evaluar la posibilidad de conectarse a una WEBCAM IP que envía imágenes sin la necesidad de una computadora que las transmita, o bien montar una cámara en la plataforma robótica y hacer el procesamiento de imágenes *onboard*, consiguiendo una unidad robótica más compacta.

6.2. Procesamiento de Imágenes

Para medir la orientación del robot se usa la WEBCAM que se encuentra montada como se indica en la Fig. 6.1, la cual captura las imágenes del robot y del entorno y las envía a la Raspberry Pi, la cual realizará el procesamiento de imágenes correspondiente para la detección de la orientación y posición del robot. La Fig. 6.2 muestra un esquema del tipo de imagen que captura la WEBCAM, con el origen de coordenadas en la esquina superior izquierda y el sentido positivo de giro del robot definido positivo en el sentido

horario.

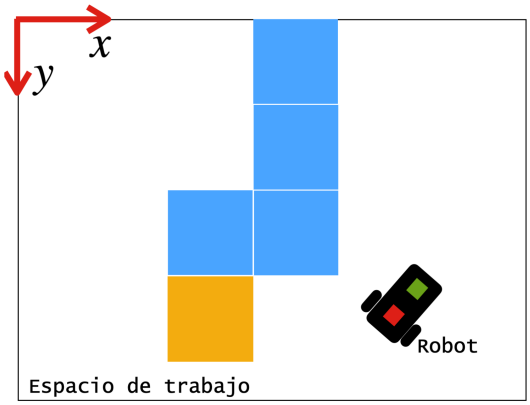


Figura 6.2: Imagen que capturaría la WEBCAM

El procesamiento de imágenes llevado a cabo se basa en la aplicación de máscaras para la identificación de cada uno de los colores relevantes de la imagen. En primer lugar, se definieron los colores rojo y verde para reconocer la parte trasera y delantera del robot, respectivamente. Los obstáculos fueron definidos en color azul y, finalmente, la caja que el robot deberá empujar será identificada con el color marrón. El resultado de aplicar las máscaras para los colores definidos se puede ver en la imagen 6.3

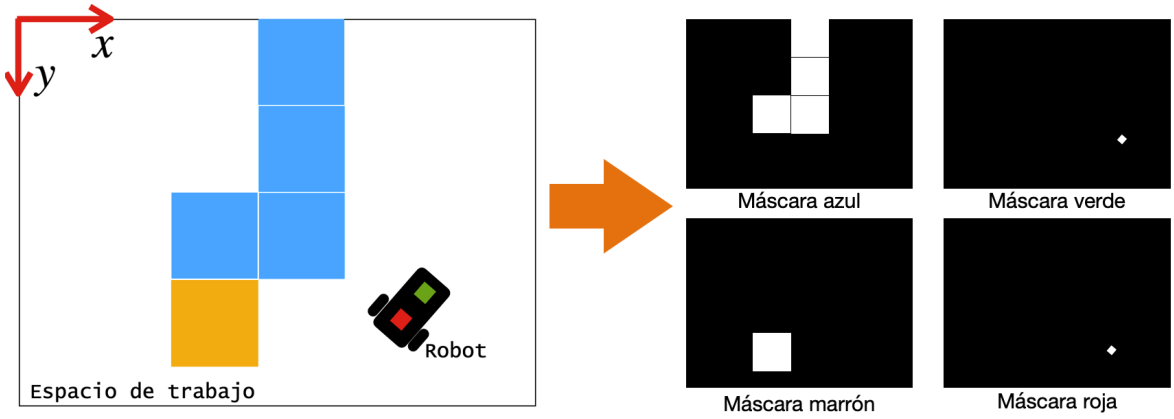


Figura 6.3: Máscaras aplicadas a cada uno de los colores definidos del sistema

La Raspberry Pi detecta las posiciones de las marcas roja y verde del robot y traza un vector (vector verde de la Fig. 6.4) entre ambas, tomando como origen del vector la marca roja. Se toma como valor angular de referencia un vector orientado en el sentido horizontal de la imagen (vector rojo paralelo al eje x de la Fig. 6.4). De esta manera, se puede calcular el ángulo de orientación del robot mediante la función matemática $atan2$ pasando como parámetros las coordenadas del vector de orientación del robot con respecto al vector de referencia. Para determinar la posición del robot, se toma el punto medio del vector de orientación, calculando el promedio simple de los centros de los cuadrados rojo y verde ubicados encima del robot. En la Fig. 6.4 se indica con un círculo celeste el centro del robot con las coordenadas (x_R, y_R).

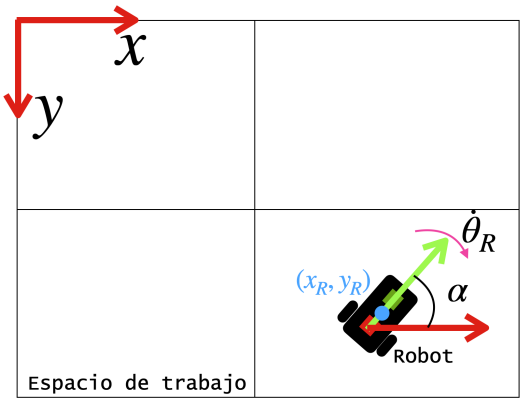


Figura 6.4: Esquema de la imagen luego de la identificación de la orientación y posición del robot

Para la detección de obstáculos, se realiza un procedimiento similar aplicando una máscara que detecta el color azul. La imagen resultante se utilizará para luego determinar que las celdas de discretización del entorno que contengan algún elemento azul en su interior sean consideradas como celdas «prohibidas». De manera similar se aplica lo anterior para determinar la locación de la caja que el robot deberá empujar.

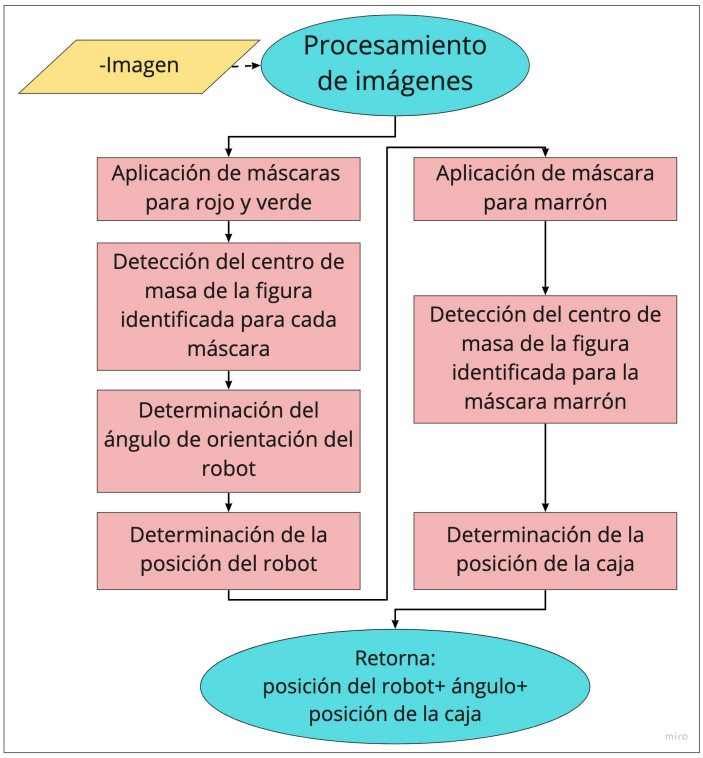


Figura 6.5: Diagrama de flujo del algoritmo de procesamiento de imágenes

El diagrama de flujo del algoritmo para el procesamiento de imágenes se muestra en la Fig. 6.5 y se midió que el tiempo que tarda la Raspberry en realizar el procesamiento de imágenes fue de 13 ms (80 FPS). Cabe mencionar que como en este trabajo se supone que los obstáculos son estáticos, no es necesario aplicar la máscara azul en todos los

ciclos de procesamiento de imágenes, sino que basta con realizarlo una vez antes de realizar la síntesis de los controladores para detectar los obstáculos en sus posiciones iniciales.

6.3. Componentes Electrónicos del Robot

Siguiendo con el esquema de montaje descripto, los componentes montados en el sistema Robot deben ser tales que permitan la conexión remota con el Sensor, la recepción de las imágenes provenientes del Sensor, el procesamiento de imágenes necesario para estimar la posición y orientación de la plataforma robótica y la detección de los obstáculos. Finalmente, hay que definir un conjunto de actuadores que permitan que el Robot se mueva y realice las tareas especificadas.

En primer lugar, las tareas de comunicación y procesamiento de imágenes se optó que sean realizadas por un microprocesador. En particular en este trabajo, se empleó una Raspberry Pi 3 modelo B+ con el sistema operativo Raspbian v3.11 montada sobre el Robot, que se conectará vía WiFi con el sensor y recibirá las imágenes capturadas por la webcam. La Raspberry Pi ejecutará un algoritmo de procesamiento de imágenes realizado en Python 3.7 que le permitirá identificar la posición y orientación del robot y los obstáculos en cada momento. Con la posición inicial del robot y la ubicación de los obstáculos iniciales, la Raspberry correrá por única vez el algoritmo que ejecutará las rutinas para sintetizar el controlador de eventos discretos. Es decir, se verificará la capacidad del robot de cumplir con las especificaciones solicitadas y se planificará una estrategia para cumplir la misión.

En segunda instancia, es necesario contar con un controlador de los motores para ambas ruedas motrices. Siguiendo las buenas prácticas del control, se propuso que un microcontrolador se encargue de comandar los motores. Una placa Arduino UNO R3 posee un microcontrolador ATmega328P y resulta una opción conveniente al ser un producto comercial, de fácil acceso y que posee un software libre y una comunidad numerosa y activa. El controlador puente H L298N permitirá comandar los motores en tanto que permite invertir el sentido de giro de los motores manejando una tensión de hasta nueve veces la tensión nominal de salida soportada por la placa Arduino, que es de solo unos 5 V. En la Fig. 6.9 se muestra un modelo 3D de los motores y se muestra cómo se ensamblan a la plataforma robótica.

Finalmente, la alimentación del robot consta de un juego de seis baterías AA recargables que, estando completamente cargadas, entregan una tensión de 7.2 V conectándolas en serie. El consumo de corriente esperado en el Robot es de aproximadamente 1.5 A para el arranque y de 0.7 A para el funcionamiento en condiciones normales. De esta aproximación se estima que las baterías utilizadas, que responden a la curva de descarga que se ve en la Fig. 6.6 (obtenidas del fabricante), permitan una autonomía

del robot de aproximadamente 45 minutos de funcionamiento ininterrumpido.

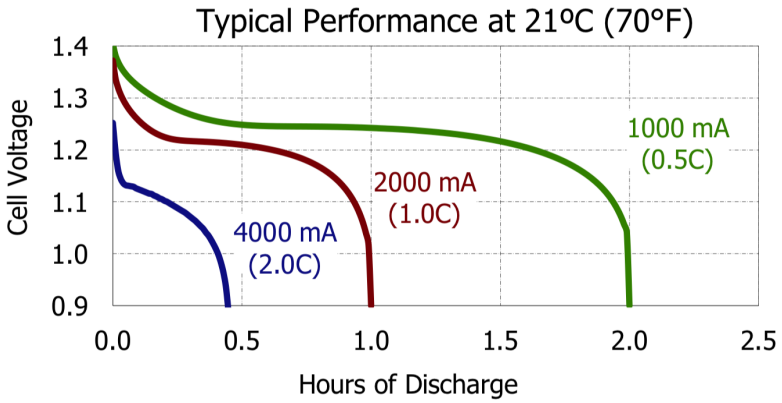


Figura 6.6: Curva de descarga de baterías ENERGIZER NH15-2000 (HR6)

El esquema de conexión completo se muestra en la Fig. 6.7, donde las líneas rojas representan las conexiones por las que circula mayor corriente, que son las encargadas de la alimentación de la placa Arduino, el puente H y la Raspberry Pi. Las líneas negras, por su parte, muestran los canales de flujo de datos y manejan corrientes bajas. La línea punteada entre Robot y Sensor indica la conexión inalámbrica entre estos sistemas. En esta figura se ve que las baterías entregarán una tensión de 7.2 V. Sin embargo, la tensión nominal de entrada de la Raspberry Pi es de 5.1 V con una variación de ±0.1 V, por lo que es necesario utilizar un regulador de tensión que reduzca la tensión de alimentación que recibirá la Raspberry Pi, manteniéndola dentro del rango de funcionamiento.

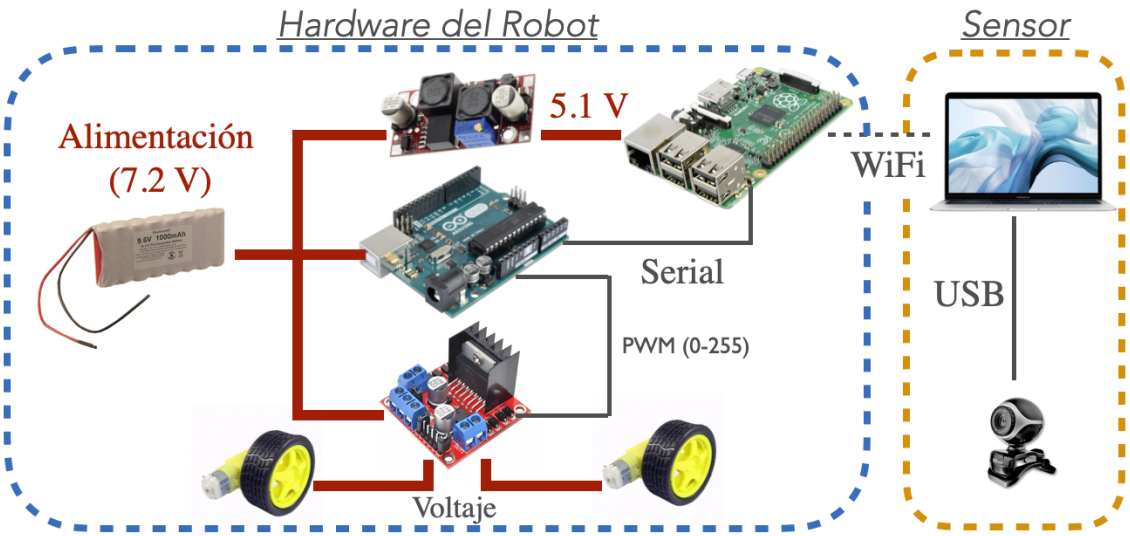


Figura 6.7: Esquema de conexión de los componentes electrónicos del robot

El regulador de tensión que se utilizará es el regulador ajustable LM2596S ADJ DCDC con el que se fijará la tensión de entrada de la Raspberry PI a 5.1 V. Indicado

con líneas negras, se ve que el Arduino controlará el puente H mediante pulsos de PWM (*Pulse Width Modulation*), gracias a los cuales puede establecerse la velocidad de giro de las ruedas. En un principio, uno podría pensar en alimentar la entrada del puente H tomando directamente la salida de los 5 V del Arduino, sin embargo, esto no se hará dado que alimentar el puente H desde el juego de baterías, da la posibilidad de conseguir un mayor torque en los motores.

Los módulos híbridos descriptos en el Capítulo 5 se ejecutarán en la Raspberry Pi, a excepción del bloque denominado Motion Control el cual se implementará en el Arduino para una mejor integración con los lazos de control. Estos módulos proporcionarán las señales para activar o desactivar los actuadores del robot (motores y sistemas de alarma). Para conseguir que los módulos híbridos interactúen con los actuadores y sensores del sistema robótico, es necesario establecer un canal de comunicación entre la Raspberry Pi (que ejecuta los módulos híbridos) y la placa Arduino (que controla los actuadores del robot). Este medio de comunicación se establecerá a través una conexión serial entre ambas placas, dado que este método permite un flujo bidireccional de la información. Mediante la conexión serie, el módulo de desplazamiento establecerá la referencia en los controladores de lazo cerrado que corren en la placa Arduino y, del mismo modo, cuando el robot esté lo suficientemente cerca de su referencia enviará un mensaje de llegada a la Raspberry Pi por este canal.

6.4. Diseño de la Plataforma Robótica

6.4.1. Chasis y Carrocería

En la Sección 5.1 se comentó acerca de la conveniencia de utilizar una plataforma robótica de tres ruedas, con dos ruedas motrices y una tercera rueda libre que le aporta estabilidad al robot. El chasis es el componente estructural que debe contener todos los componentes que deben montarse sobre el robot. En el diseño del chasis del robot se busca adecuar la distribución del peso de manera que se asegure una fricción mínima entre las ruedas motrices y la superficie de apoyo y evitar así que las ruedas deslicen. Además, se busca que el diseño resulte en una plataforma compacta y portable.

En la Sección 6.3 se describieron los componentes electrónicos necesarios para el funcionamiento del robot. Como criterio de diseño se propuso dejar acceso libre a los puertos de entrada y salida de la Raspberry Pi y del Arduino para poder realizar las pruebas para la puesta a punto. También se decidió ubicar el juego de baterías de manera que sea sencillo retirarlas para su carga y volver a colocarlas. Asimismo, se colocó un interruptor de encendido para todo el sistema montado sobre la plataforma robótica.

Para poder montar todos los elementos electrónicos y mecánicos, se propuso un

diseño del chasis como el que se muestra en la Fig. 6.8. Este diseño con dos «niveles» permite conseguir un robot compacto y con un fácil acceso a las baterías y al interruptor de encendido.

(a) Chasis: nivel superior **(b)** Chasis: nivel inferior

Figura 6.8: Modelo del chasis del sistema robótico

Los dos niveles se ensamblan mediante tres bulones y los motores DC se acoplan con un sistema de «ranura-pasador» donde dos pasadores se encastran en las ranuras del chasis y se ubica el motor en el medio. Luego con dos bulones por motor se asegura la posición del motor al chasis, como se muestra en la Fig. 6.9.

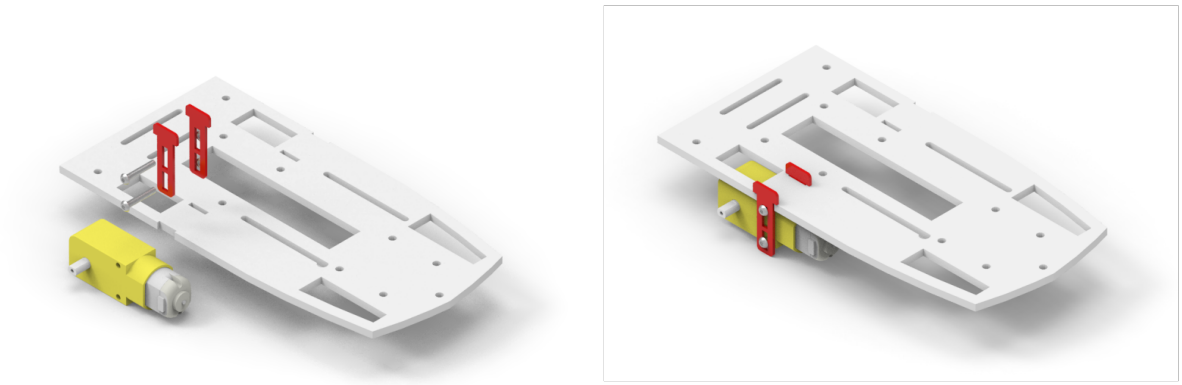
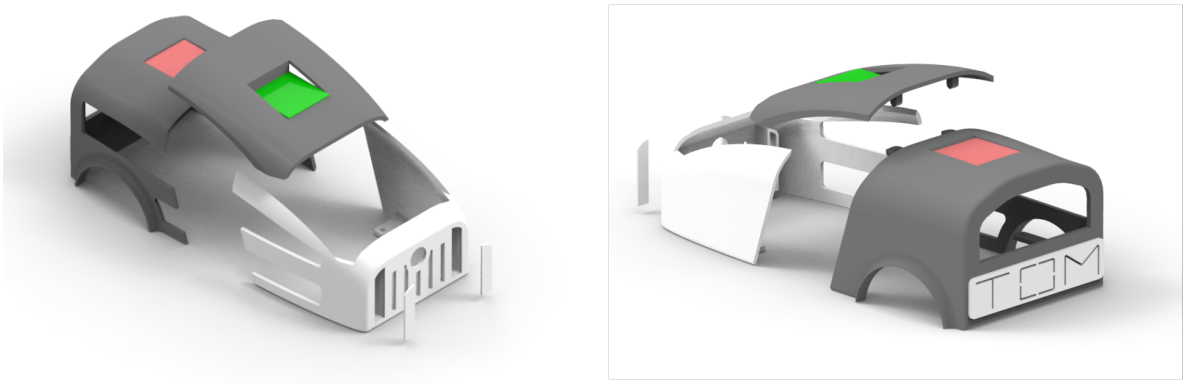


Figura 6.9: Sistema de sujeción de los motores

Se propuso que la fabricación del chasis y la carrocería se lleve a cabo mediante manufactura aditiva de plástico, más precisamente por impresión 3D. Por este motivo, y siguiendo las buenas prácticas de este método de fabricación, se definió el espesor de las plataformas que conforman el chasis de 4 mm y el espesor de la pared de la carrocería de 3 mm.

El último componente del diseño del hardware del robot es la carrocería, el cual es el componente que envuelve al robot, previene que los golpes afecten los componentes montados y evita que estos se ensucien con polvo del ambiente. Para la carrocería se contempló la posibilidad de conectar diodos LED en la parte posterior para realizar las pruebas de reacción mencionadas en la Sección 4.1.4. Para el diseño de la carrocería, se

dejaron libres los accesos para el interruptor de encendido y de los puertos de entrada de las placas Raspberry Pi y Arduino, dado que el robot se encuentra en etapa de prototipado y es necesario tener todos los puertos de entrada de las placas a disposición. Además, se diseñó un mecanismo de encastre de la parte superior frontal de la carrocería que al levantarse deja al descubierto las baterías del robot de manera que su reemplazo sea sencillo y rápido. Para la fabricación y montaje de la carrocería fue necesario diseñar la carrocería en tres partes independientes que se muestra en la Fig. 6.10. Finalmente, se diseñaron ópticas que se montarán por encastre en el frente de la carrocería y servirán para proteger los LEDs que se montarán en la misma posición.

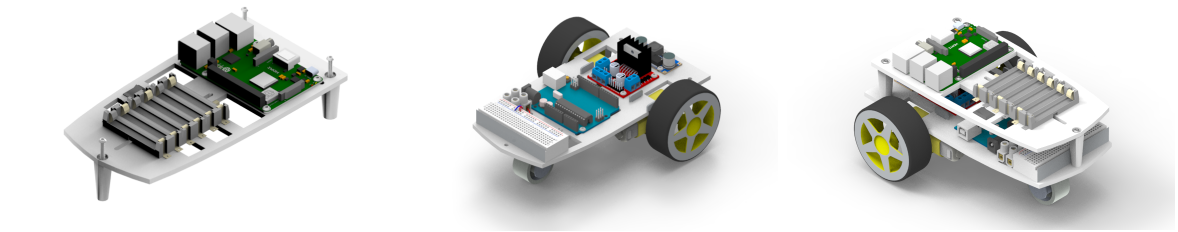


(a) **(b)**

Figura 6.10: Modelo de la carrocería

6.4.2. Modelo del Ensamble

Para ver la disposición final de los elementos que componen el Robot, se hizo un modelo de la plataforma del robot con los componentes electrónicos y mecánicos montados y con la carrocería ubicada en su lugar. En las Fig. 6.11a y 6.11b se pueden ver los componentes electrónicos y mecánicos ensamblados en el chasis por niveles y en la Fig. 6.11c se muestra el ensamble de ambos niveles sin la carrocería.



(a) Ensamble de componentes: nivel superior **(b)** Ensamble de componentes: nivel inferior **(c)** Modelo del ensamble total sin carrocería

Figura 6.11: Ensamble de los componentes electrónicos en cada nivel y ensamble total sin carrocería

Finalmente en la Fig. 6.12 se ve el ensamble real de la plataforma robótica con la carrocería diseñada, en la cual se puede apreciar que en la parte superior se ubicaron las marcas de colores para la detección de la orientación y posición del robot.

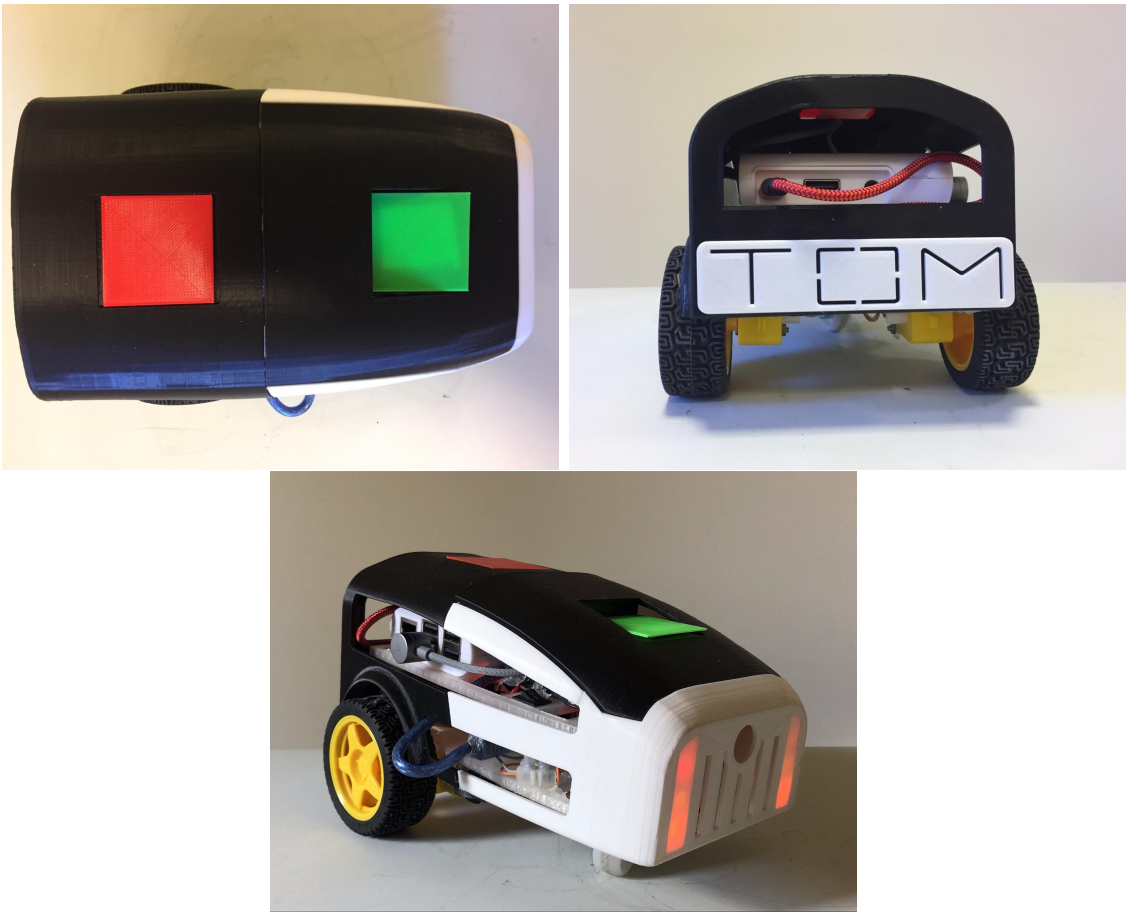


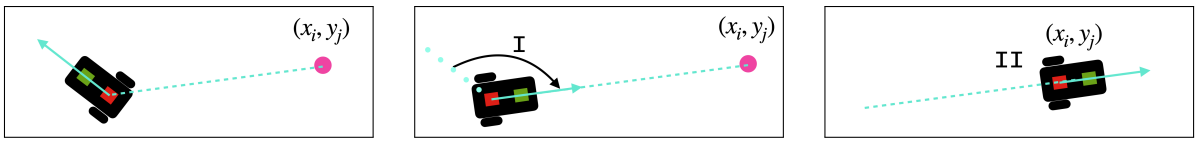
Figura 6.12: Fotos del robot TOM construido.

Capítulo 7

Lazos de Control

> *"Mi sensación de progreso se basa en la ilusión de que las cosas a mi alrededor no van a cambiar y que finalmente he logrado un poco de control sobre ellas".*
>
> — Hugh Prather, escritor estadounidense

Una vez completada la etapa de construcción de una plataforma robótica y de la selección y montaje de los componentes electrónicos y mecánicos del robot, es posible diseñar los controladores del sistema. En este capítulo se presentan los pasos realizados para el diseño e implementación de los lazos de control que servirán para controlar el movimiento de la plataforma robótica haciendo uso de los actuadores y sensores presentados en el Capítulo 6. Los lazos de control con los que contará el robot se encargarán de llevar el sistema desde un origen hasta una posición deseada del espacio y serán comandados por el bloque de control de movimiento Motion Control que se introdujo en la Sección 5.1. En la Fig. 7.1 se muestra la secuencia de control propuesta, la cual implica por un lado el control de ángulo del robot, para alinearlo con el punto que se pretende alcanzar, y por otro lado, el control de posición del robot cuando avanza el línea recta hasta el objetivo. En las secciones posteriores se muestran los pasos a seguir para el diseño de los controladores de ángulo y posición del robot.



(a) Configuración inicial.

(b) El robot se alinea con el objetivo.

(c) El robot avanza hasta el objetivo.

Figura 7.1: Secuencia de movimientos del robot para alcanzar la posición (x_i, y_j)

7.1. Controlador de Ángulo

7.1.1. Identificación de la Planta

El primer paso para el diseño del controlador del ángulo de orientación del robot es la identificación de la planta. Mediante la identificación de la planta se pretende encontrar un modelo matemático del sistema dinámico que relacione la entrada u del sistema y salida y, a partir de un conjunto de datos medidos [32, 33]. Dos maneras clásicas de abordar este problema es mediante la caracterización **paramétrica** y **no-paramétrica** del sistema. El primer método, parte de suponer conocido el comportamiento del sistema y se propone un modelo matemático con parámetros a determinar de manera que el modelo se ajuste al comportamiento del sistema real. Por otro lado, la caracterización **no-paramétrica** es una técnica para determinar el comportamiento del sistema dinámico sin hacer una preselección de los posibles modelos. Existen múltiples métodos automáticos para la identificación de la planta como el método de mínimos cuadrados [34]. No obstante, para controlar el robot Tom, se realizó la identificación **paramétrica** manualmente para simplificar esta etapa en una primera instancia del proyecto.

7.1.2. Modelo Matemático del Robot

La identificación de la planta parte de proponer un modelo matemático que represente el sistema físico con el que se va a trabajar. En [33], se presenta el modelo matemático de un motor a partir de los cuales se logrará hallar una relación entre la tensión de entrada que recibirán cada motor y la posición angular del eje del motores. Por otra parte se plantea la relación entre el giro de los motores y el giro del robot. Con estas relaciones, se pretende hallar una expresión que vincule la señal de entrada del sistema (voltaje en los motores) con el ángulo que gira el robot.

Los sistemas conformados por motores pueden abordarse tanto desde una perspectiva mecánica como eléctrica. Desde el punto de vista mecánico, los motores ejercen un torque τ produciendo una aceleración $\ddot{\theta}_m$ en el eje, donde el subíndice m se utiliza para referirse al motor. Suponiendo que se sujeta una rueda al eje de cada motor y teniendo en cuenta que los motores estarán girando a velocidad $\dot{\theta}_m$, aparecerá una fuerza viscosa, producto de la fricción mecánica entre la rueda y la superficie de contacto, caracterizada por un coeficiente b. Siguiendo con [32, 33], el torque que ejercen los motores puede ser considerado proporcional a la corriente que circula por la armadura del motor i_a, con un coeficiente de proporcionalidad K_t. Con lo anterior, se obtiene una expresión para el balance de torques del sistema Motor la cual se muestra en la ecuación 7.1.

$$J_m\ddot{\theta}_m + b\dot{\theta}_m = \tau = K_t i_a, \tag{7.1}$$

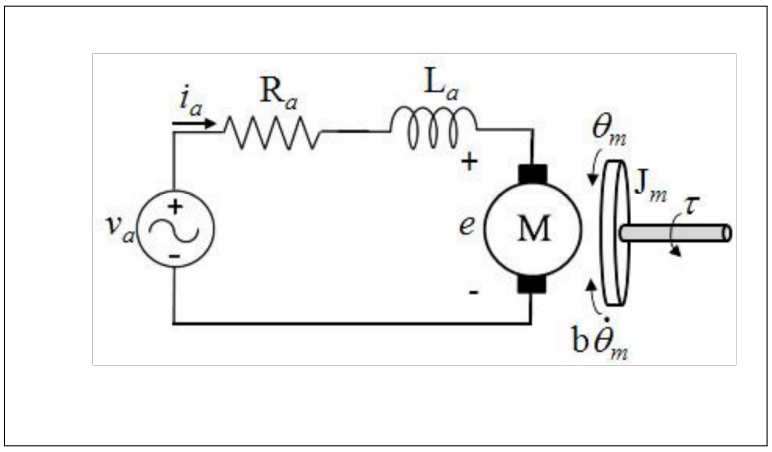


Figura 7.2: Representación circuital de la armadura de un motor de corriente continua

donde el término J_m es el momento de inercia asociado al giro de un rueda sujeta al motor.

Desde el punto de vista eléctrico, el motor es una bobina por la que circula una corriente i_a. Debido a esta corriente, se establece un campo magnético que interacciona con un campo magnético fijo en el motor, esto produce un torque y el correspondiente giro del eje. Al girar el motor, se genera una tensión inducida e que se opone a la tensión aplicada v_a. Aplicando la regla de mallas de Kirchhoff al circuito de la armadura del motor que se muestra en la Fig. 7.2, se obtiene la ecuación 7.2 que relaciona la corriente que circula por la bobina con la tensión aplicada en sus bornes.

$$L_a \frac{di_a}{dt} + R_a i_a = v_a - e, \tag{7.2}$$

donde el término R_a es la resistencia del bobinado y L_a es la inductancia de la armadura. Siguiendo la bibliografía consultada, la fuerza contraelectromotriz e puede suponerse proporcional a la velocidad de giro del motor $\dot{\theta}_m$, de manera que la ecuación 7.2 resulta:

$$L_a \frac{di_a}{d_t} + R_a i_a = v_a - K_e \dot{\theta}_m, \tag{7.3}$$

donde K_e es la constante proporcional de la fuerza contraelectromotriz.

Luego de determinar las ecuaciones de la dinámica del motor, es necesario relacionar el comportamiento de los motores con el giro que estos producen a la plataforma robótica. Para ello, se halla una expresión para el ángulo que rota el robot cuando las ruedas realizan una vuelta completa, y se obtiene que la relación entre el giro de la ruedas y el giro del robot se da por la expresión 7.4,

$$\theta_R = \frac{\theta_m r_w}{l}, \tag{7.4}$$

donde r_w es el radio de la rueda sujeta al motor, l es la distancia del centro del robot al punto de contacto de la rueda con el piso y θ_R es el ángulo de giro del robot. Notar que para que esta relación sea válida, es necesario garantizar que las ruedas no deslicen en la superficie de trabajo.

Combinando las ecuaciones 7.1, 7.3 y 7.4 y despreciando el aporte de la parte inductiva del motor, que suele tener una respuesta demasiado rápida para los tiempos característicos del sistema mecánico, se puede obtener la expresión 7.5. Esta ecuación relaciona el valor de tensión de entrada $V(s)$ del sistema con la variable que se desea controlar, es decir, el ángulo $\theta_R(s)$, expresada en el dominio de la frecuencia mediante la transformada de Laplace. Como se aprecia en esta última ecuación, el modelo obtenido para la caracterización del sistema real es un modelo de segundo orden. Cabe mencionar que de no haber despreciado la contribución de la inductancia del motor, se habría conseguido un modelo más complejo de tercer orden. Sin embargo, se considera que para una primera estimación del modelo del robot, tomar $L_a \approx 0$ se considera una aproximación válida.

$$\frac{\theta_R(s)}{V(s)} = \frac{r_w \frac{K_t}{J_m l}}{s(s + \frac{K_e K_t}{J_m})} = \frac{K_s}{s(s+D)}, \tag{7.5}$$

donde $K_s = r_w \frac{K_t}{J_m l}$ y $D = \frac{K_e K_t}{J_m}$.

7.1.3. Caracterización Paramétrica del Sistema Robótico

El proceso de identificación de la planta consiste en hallar un modelo matemático del sistema, al que se denominará Planta P, como una relación entre la entrada u y salida y del sistema. En este trabajo, para la identificación del comportamiento dinámico del sistema, se propone medir la orientación del robot cuando se introduce una señal de entrada de PWM (*Pulse Width Modulation*) constante en cada motor de manera que estos giren a la misma velocidad pero en sentido contrario. La señal de PWM permitirá regular la velocidad de giro de los motores ajustando el ancho de los pulsos de tensión (de amplitud de 5 V) que se aplica a los bornes de cada motor. Los motores, a su vez, cuentan con resistencias y capacitancias que actúan como filtro pasa bajo de la señal de entrada, por lo que la tensión efectiva que recibirán será la componente media de la señal de entrada. Esto significa que, modulando el ancho del pulso de 0 a 100 % del período del pulso, se puede generar la señal $V(s)$ entre 0 a 5 V. Se puede suponer que existe una relación lineal entre la tensión $V(s)$ y la señal $PWM(s)$ de manera que lo que se hará es tratar de identificar la planta:

$$\frac{\theta_R(s)}{PWM(s)} = \frac{K'_s}{s(s+D)}, \tag{7.6}$$

donde K'_s es la constante K_s multiplicada por el factor de proporcionalidad entre $V(s)$ y $PWM(s)$.

Para medir el ángulo de orientación del robot, cada vez que el robot reciba una imagen proveniente desde la LAPTOP, este realizará el procesamiento de imágenes descripto en la Sección 6.2 y se obtendrá como resultado el valor del ángulo de orientación y la posición del centro de masa del robot. La detección de la posición en esta etapa de caracterización del robot, servirá para poder estimar el radio de giro del robot y realimentar el proceso de diseño de la discretización del entorno para determinar el tamaño mínimo de la celda de discretización de la Sección 4.1.2.

En la Fig. 7.3 se muestran la estimación del ángulo para las posiciones correspondientes a 0°, 90°, 180° y −90° medidos en el sentido de giro horario por el algoritmo de procesamiento de imágenes. De aquí se ve que el error de estimación es de aproximadamente ±4°.

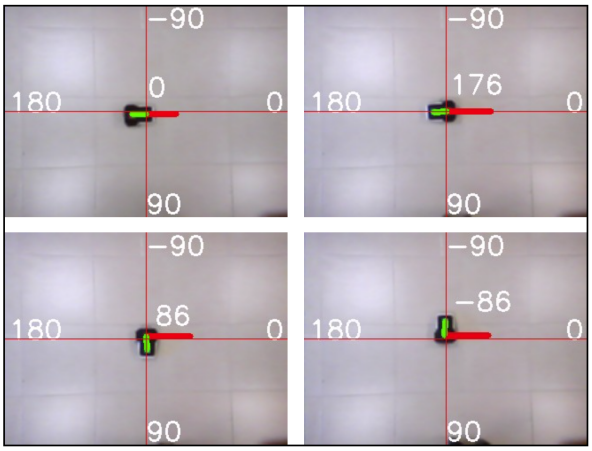


Figura 7.3: Estimación del error de medición del ángulo de orientación del robot

Cuando los motores reciben un valor fijo de PWM, con distinto signo en cada motor, el robot comienza a girar. Se determinó que hay una zona muerta para valores de PWM inferiores a 135, para los cuales los motores no pueden vencer la inercia del robot. Siendo 255 el máximo valor de PWM, el valor de 135 de la banda muerta muestra que los motores en esta configuración se encuentran bastante exigidos debido al peso del robot. De todas maneras, se propuso continuar trabajando sin realizar modificaciones en el diseño del robot para lograr cubrir los objetivos de este trabajo y se deja para trabajo futuro un análisis más detallado de estos componentes. Una manera de trabajar con este tipo de comportamiento no lineal es intentando linealizar la planta definiendo una variable PWM' como:

$$PWM(s)' = (PWM(s) - 135) \times \textit{signo del sentido de giro}, \tag{7.7}$$

donde el sentido de giro queda definido por el puente H que comanda los motores. Con

esta nueva variable, el sistema definido en la ecuación 7.5 puede considerarse lineal y resulta:

$$\frac{\theta_R(s)}{PWM'(s)} = \frac{K'_s}{s(s+D)}. \tag{7.8}$$

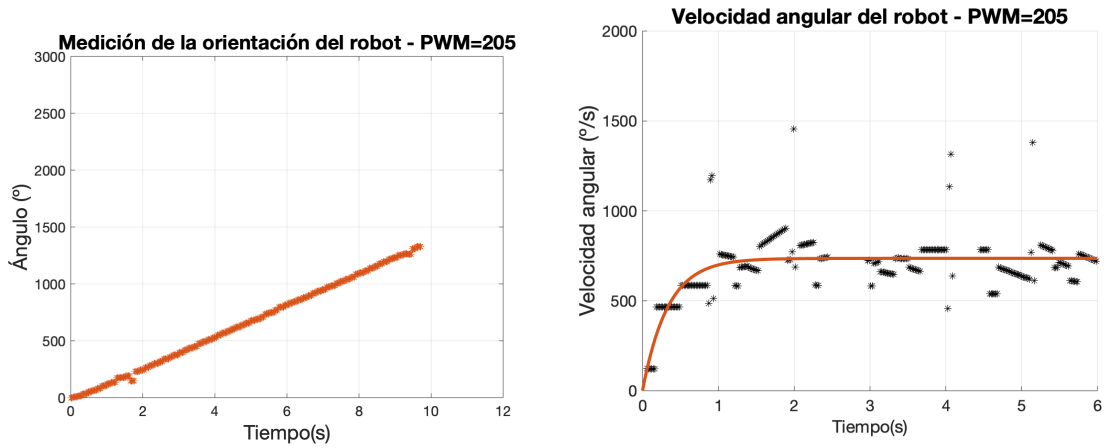
Para la caracterización de la planta y el diseño de controladores, resulta conveniente que el sistema con el que se está trabajando tenga un comportamiento que tienda a un valor límite o estacionario. Para poder identificar esta planta entonces, se propone caracterizar la velocidad angular del sistema frente a una entrada de PWM' fija. Derivando la expresión de 7.8 en el plano de Laplace, se obtiene la expresión para la velocidad angular de la ecuación 7.9.

$$\frac{\dot{\theta}_R(s)}{PWM'(s)} = \frac{K_s}{(s+D)} \tag{7.9}$$

La caracterización de este tipo de plantas, en las cuales la respuesta tiende a un valor estacionario, resulta conveniente desde el punto de vista de que el valor estacionario al que converge el sistema será igual a $PWM' \times \frac{K'_s}{D}$, y, además, porque el lugar del polo D se puede determinar de manera que el comportamiento de la planta sea similar al del sistema real en el régimen transitorio.

En la Fig. 7.4a se muestran las mediciones del ángulo realizadas cuando el robot está girando debido a una señal de entrada de PWM' de 50 . Este valor de PWM' se tomó como referencia para la caracterización de la planta dado que se encuentra a la mitad del rango de valores de PWM' (entre 0 y 120). Derivando numéricamente los resultados obtenidos de la medición de la posición angular con el método de diferencias finitas hacia atrás, y graficando la respuesta que tendría el modelo matemático de la ecuación 7.9 frente a una señal de PWM' de 50, se pueden ajustar los parámetros K'_s y D (ver Fig. 7.4b). Los valores de K'_s y D que se hallaron fueron de $K'_s = 44$ y $D = 3$, respectivamente. Se observó que evaluando la respuesta del modelo de la velocidad angular del sistema para los valores de PWM' de 0 y de 120, es decir los valores extremos del rango de PWM', los valores de K'_s y D varían en menos de un 5 %. Por este motivo y para simplificar esta etapa de diseño, se optó por considerar constantes los parámetros del modelo de segundo orden K'_s y D, lo que permite suponer un comportamiento lineal de la planta en el rango de valores de PWM' de 0 a 120.

Para calcular el radio de giro del robot se confeccionó una gráfica del plano XY que representa la superficie donde gira el robot. En la Fig. 7.5 se muestra la trayectoria realizada por el centro de la plataforma robótica de donde se concluye que tiene un radio de giro de aproximadamente 15 px. Teniendo en cuenta este radio de giro y el tamaño en píxeles que el robot ocupa en la imagen, alrededor de (30×25) píxeles, se determina que el tamaño de celda de la discretización del entorno sea de 60 píxeles para



(a) Mediciones de la posición angular del robot para un valor de PWM' de 50

(b) Resultados de la derivación numérica de los datos de la posición angular frente a una señal de entrada de PWM' de 50

Figura 7.4: Caracterización de la planta

asegurar que el robot esté contenido dentro de la celda cuando realiza las maniobras para desplazarse de una celda a la siguiente.

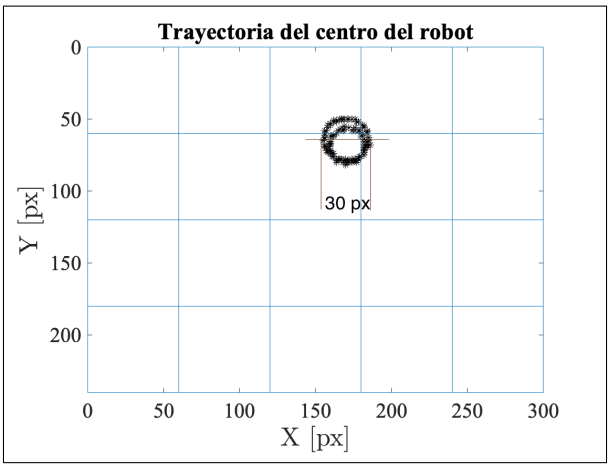
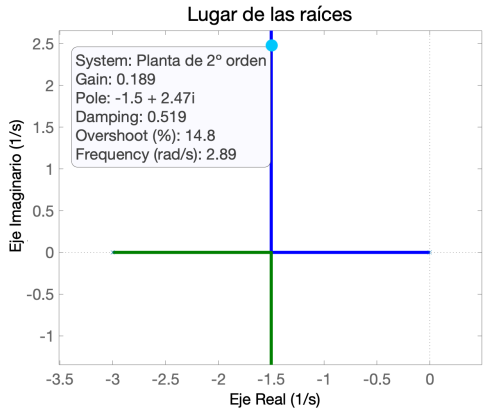


Figura 7.5: Medición del radio de giro del robot

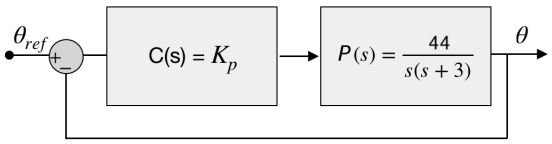
7.1.4. Diseño de un Controlador de Ángulo

El enfoque de control que se usó para este trabajo solo considera el uso de la señal del sensor de posición angular para controlar la orientación del robot, debido a que es una planta simple de segundo orden. El controlador de ángulo que se propone realizar es un controlador proporcional, de manera que el esquema de control no resulte muy complejo en una primera etapa del proyecto. El lugar de las raíces del sistema para la planta de segundo orden (ecuación 7.8) se muestra en la Fig. 7.6a. De aquí se decidió que la ganancia del sistema sea de $K_p = 0{,}19$ para tener un sobrepico no mayor al $M_p = 15\,\%$. Un esquema del lazo de control de ángulo se observa en la Fig. 7.6b, donde

el lazo de realimentación se obtiene del procesamiento de imágenes que la Raspberry recibe desde la WEBCAM.



(a) Lugar geométrico de las raíces del modelo de segundo orden linealizado de la planta



(b) Diagrama de bloques del controlador de ángulo para el modelo de segundo orden linealizado de la planta

Figura 7.6: Controlador del ángulo de orientación del robot

Debido a la naturaleza no lineal del sistema real, para la implementación del controlador de ángulo, la acción de control estará determinada por el cálculo del error del ángulo multiplicado por la constante del controlador $K_p = 0{,}19$ sumado al valor de la zona muerta, de 135 en valor de PWM. El sentido de giro de los motores se determina en función de qué sentido de giro requiere el camino más corto para alinearse con el objetivo. Para esto, se mide el ángulo que debe girar el robot para alinearse con el objetivo como la diferencia entre el ángulo de orientación del robot y el ángulo que se forma desde la marca roja del robot y el objetivo, ambos medidos desde la horizontal. En el caso de que esta diferencia resulte en un valor fuera del rango $(-180°; 180°]$ se debe sumar o restar 360° al valor obtenido de modo que quede contenido dentro de este rango. Por ejemplo, si el ángulo de orientación del robot es 0° y la referencia de ángulo es de 190° medido en sentido horario, entonces la diferencia entre ambos resulta −190° por lo que se debe sumar 360°, es decir, 170°. Si el valor resultante es menor a cero, entonces el robot debe girar en sentido antihorario, mientras que si es mayor a cero, entonces deberá girar en el sentido horario. En el caso ilustrado en la Fig. 7.7, el robot debería girar en el sentido horario en tanto que el ángulo indicado como α_{dif} producirá la trayectoria más corta para alinearse con el objetivo y está medido en el sentido horario de giro del robot.

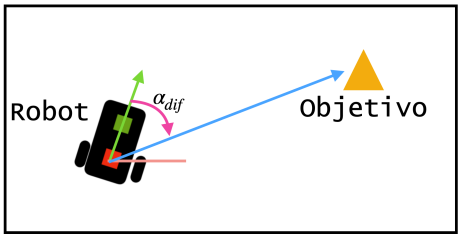


Figura 7.7: Determinación del sentido de giro del robot

7.2. Controlador de Posición

Una vez que el robot está alineado con el punto que tiene que alcanzar, debe avanzar en línea recta hasta llegar lo suficientemente cerca del objetivo. En este trabajo el controlador de posición por el que se optó utilizar toma tres valores de PWM': 15, 5 y 0, es decir, con el robot detenido. Este controlador mide la distancia del centro del robot hasta el objetivo y determina tres zonas. La primera zona se define como aquella cuya distancia al objetivo es mayor a 15 píxeles y le corresponde un valor de PWM' de 15. A la tercera zona le corresponde el valor de PWM' de 0 y se define como la zona contenida en un círculo de radio de 10 píxeles. Cuando el robot ingresa a esta zona se considera que el robot ha alcanzado la referencia (x_{ref}, y_{ref}) y se detiene. La segunda zona está contenida en el anillo de radio externo 15 píxeles y de radio interno 10 píxeles, como se muestra en la Fig. 7.8. A la segunda zona le fue asignado un valor de PWM' de 5 de modo que la desaceleración del robot no sea tan brusca a medida que se acerca al objetivo.

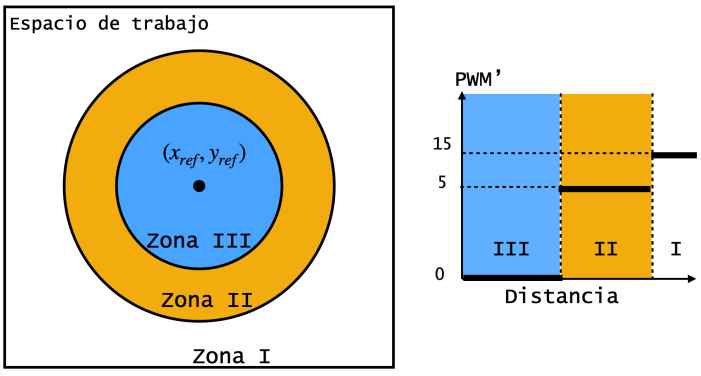


Figura 7.8: Zonas del controlador de posición

7.3. Integración de los Controladores

El esquema de control de la Fig. 7.9 muestra la implementación de la estrategia de *Motion Control* descripta en el Capítulo 5 para el módulo de desplazamiento del robot. Esta secuencia de control toma como parámetros de entrada la posición final que

debe alcanzar el robot y la información que devuelve el algoritmo de procesamiento de imágenes definido en la Sección 6.2, es decir, la posición del robot y la orientación que tiene. Por lo tanto, se tiene que el lazo de control debe esperar que se reciba una imagen, se realice el correspondiente procesamiento para detectar la posición y orientación del robot y, finalmente, que la Raspberry Pi envíe esta información al Arduino que es donde corren los lazos de control. El resultado de esto es que, el lazo de control en su totalidad, corre a una frecuencia de aproximadamente doce ciclos por segundo (12 Hz), que es la frecuencia a la que se reciben las imágenes mediante la conexión WiFi entre la LAPTOP y la Raspberry, que ciertamente es más baja que la frecuencia a la que se pueden procesar las imágenes (80 Hz).

El algoritmo del lazo de control comienza con el **Controlador de ángulo**, el cual determina el ángulo que debe rotar el robot, denominado **dif**. Cuando el robot está orientado con respecto al objetivo dentro de un rango de ±8°, el controlador de ángulo se detiene y se procede con el control de posición. A medida que el robot avanza hacia el objetivo, puede ocurrir que el robot ya no quede alineado dentro de los ±8°, por lo que se debe corregir el ángulo antes de continuar avanzando.

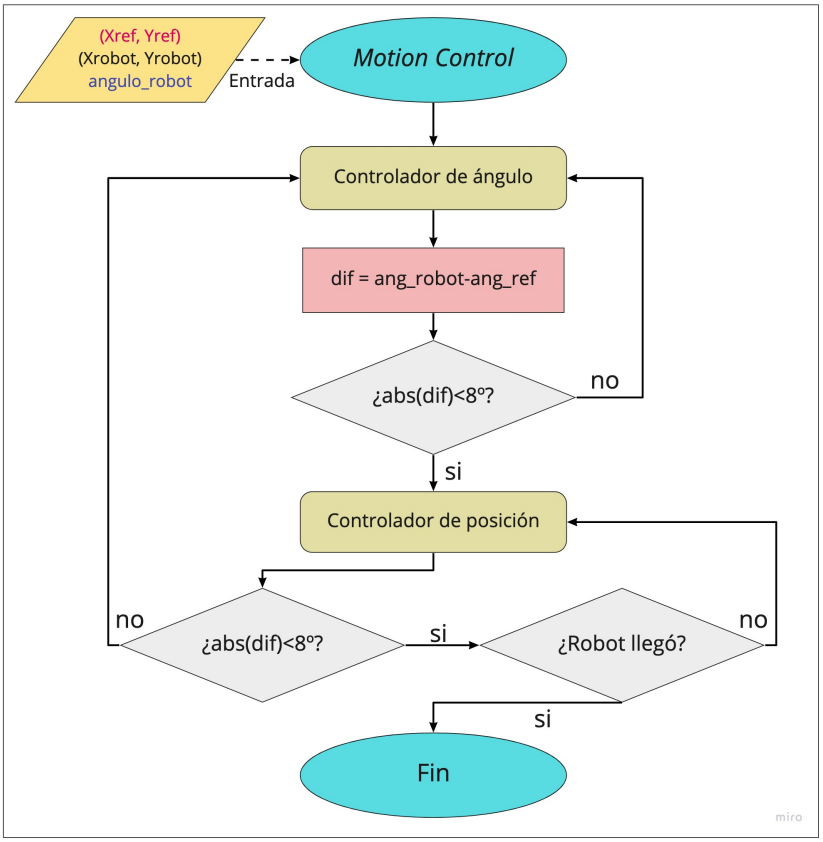


Figura 7.9: Esquema de control del movimiento del robot

El hecho de que el robot acomode primero su ángulo antes de avanzar permite tener un mayor control de las trayectorias que realiza el robot y maniobrar en un espacio más reducido lo que permite reducir el tamaño de la celda de la discretización del entorno.

Para esta estrategia de control, se vio que el robot tiene un movimiento «oscilante» debido, en parte, a una frecuencia de ejecución del lazo de control relativamente baja (12 Hz) y a un esquema control que alterna el control de ángulo y el de avance, acomodando el ángulo antes de continuar avanzando. El desarrollo de otras estrategias de control de movimiento, como por ejemplo el cómputo y seguimiento de trayectorias, se deja como trabajo futuro.

Finalmente, en la Fig. 7.10 se muestra la trayectoria realizada por el centro de masa del robot siguiendo el esquema de control descripto en la Fig. 7.9 para alcanzar un objetivo (punto anaranjado). Se indica con una flecha roja la orientación inicial del robot y el punto que debe alcanzar el robot se fijó en la posición $(1600, 800)mm$ partiendo del origen en A. De las pruebas realizadas, se obtuvo que el robot se detuvo en la posición B (indicado con un asterisco) cuyas coordenadas son $(1573, 773)mm$, lo que implica un error de $72mm$, equivalente a 10 píxeles de la imagen capturada por la WEBCAM.

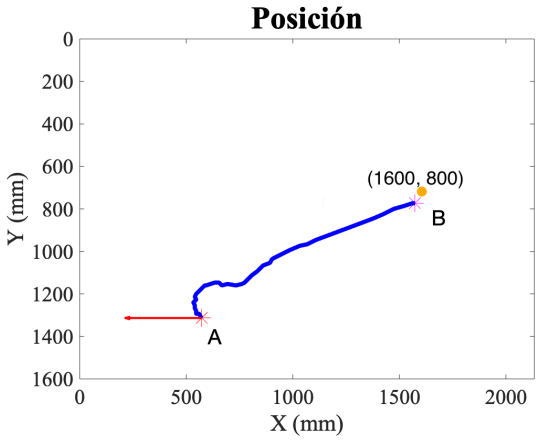


Figura 7.10: Trayectoria realizada por el robot para alcanzar un punto del espacio de trabajo.

De la Fig. 7.10 se ve que toda la rotación que debe realizar el robot para alcanzar el objetivo se produce al principio del proceso de desplazamiento, y luego, a medida que avanza, se realizan pequeñas correcciones. De esto surge que resulta conveniente controlar en primer lugar el ángulo del robot, y esperar a que el robot quede alineado en $\pm 8°$ de la referencia antes de avanzar, dado que de esta manera se tiene un mayor control de la trayectoria que realizará el robot y se ayuda a prevenir que el robot pase por celdas no previstas cuando va hacia otra celda.

Capítulo 8

Módulos Híbridos (Implementación)

"El todo es más que la suma de sus partes".
— Aristóteles, filósofo y polímata de la Antigua Grecia

En este capítulo se muestra el procedimiento realizado para la implementación de los módulos híbridos introducidos en el Capítulo 5. Esta etapa es la última fase del proceso de construcción del robot, debido a que la implementación de los módulos híbridos pretende integrar todos los componentes del proyecto para lograr una plataforma robótica completamente funcional. Además, realizar esta etapa en la última fase del proyecto permite cierta flexibilidad a la hora de diseñar el hardware del robot y los lazos de control.

Los módulos híbridos se implementaron en la Raspberry Pi en el lenguaje de programación Python v3.7, y el *Motion Control* y los lazos de control de los que depende se implementaron en la placa Arduino.

8.1. Módulo de Desplazamiento

El primer módulo que se implementó, se refiere a la capacidad principal del robot que es el desplazamiento en el espacio de trabajo. Tal como se explicó en la Sección 5.1, el robot cuenta con una estrategia de movimiento para alcanzar una posición de referencia desde su posición inicial. Para llevar a cabo esta estrategia, se propuso implementar lazos de control para el ángulo de orientación y un controlador de posición, como se explicó en el Capítulo 7. El módulo híbrido encargado de la capacidad de desplazamiento del robot, recibirá del ENACTOR la referencia que el controlador discreto indica que se debe alcanzar como parte de la misión del robot. Por ejemplo, si la acción que sigue en la lista de eventos del autómata es *go.1.0*, el ENACTOR interpretará que la acción

go implica un llamado al módulo híbrido del desplazamiento y pasa como parámetro la coordenada *(1,0)*, que se refiere a la celda *(1,0)* del espacio discretizado. El módulo híbrido, deberá traducir esta referencia de celda en términos de una coordenada del espacio de trabajo del robot (x, y) y procederá a fijarla como nueva referencia a alcanzar por los lazos de control de ángulo y posición. En el Capítulo 5 se vio que una forma de interpretar una coordenada de una celda en términos de una coordenada del espacio de trabajo continuo es tomando como referencia el punto medio de la celda, como se ve en la Fig. 5.1. Cuando el robot alcanza la referencia dentro de un radio admisible, menor a 10 píxeles de la imagen capturada por la webcam, el controlador de posición del Arduino envía un mensaje a la Raspberry a través del puerto serie, indicando su arribo al objetivo, y el módulo híbrido agrega el evento *at.1.0* a la lista del eventos del controlador discreto. Lo anterior se puede ver ilustrado en la Fig. 8.1.

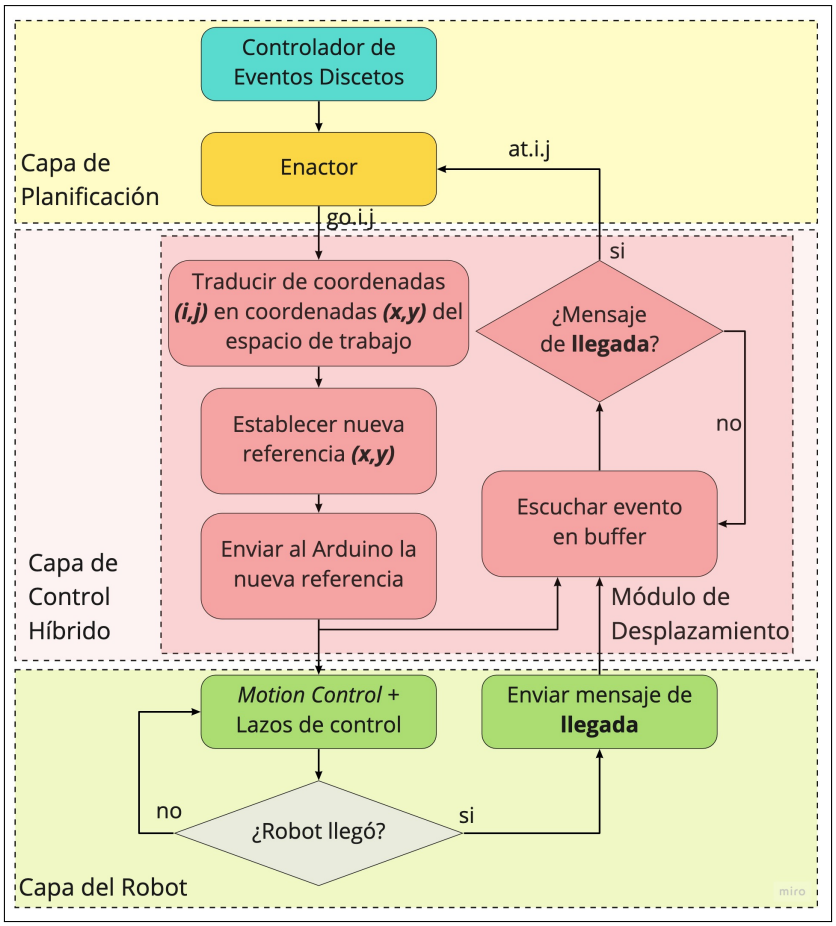


Figura 8.1: Descripción del módulo de desplazamiento

En caso de que el ENACTOR reciba un evento, por ejemplo *at.i.j*, que no esté previsto en el plan, entonces despliega un mensaje de error indicando que no es el evento esperado y el programa termina deteniendo al robot.

8.2. Módulo de Activación de Alarma

En segundo lugar, se encuentra el módulo híbrido encargado de traducir los eventos *alert.on* y *alert.off* en señales de los actuadores del robot para generar una alerta de luces y sonidos cuando el robot entra a una «zona de peligro». Este módulo es llamado desde el ENACTOR, el cual, al detectar que el evento que sigue en la lista de acciones es *alert.on* o *alert.off*, modifica el valor de una variable **estado_alerta** a *Verdadero* o *Falso*, respectivamente. La variable **estado_alerta** es enviada como parámetro de entrada del Arduino, el cual, dependiendo del valor de esta variable, activa o desactiva los actuadores correspondientes al sistema de alarma. La Fig. 8.2 esquematiza el proceso de activación o desactivación de la alarma mediante este módulo.

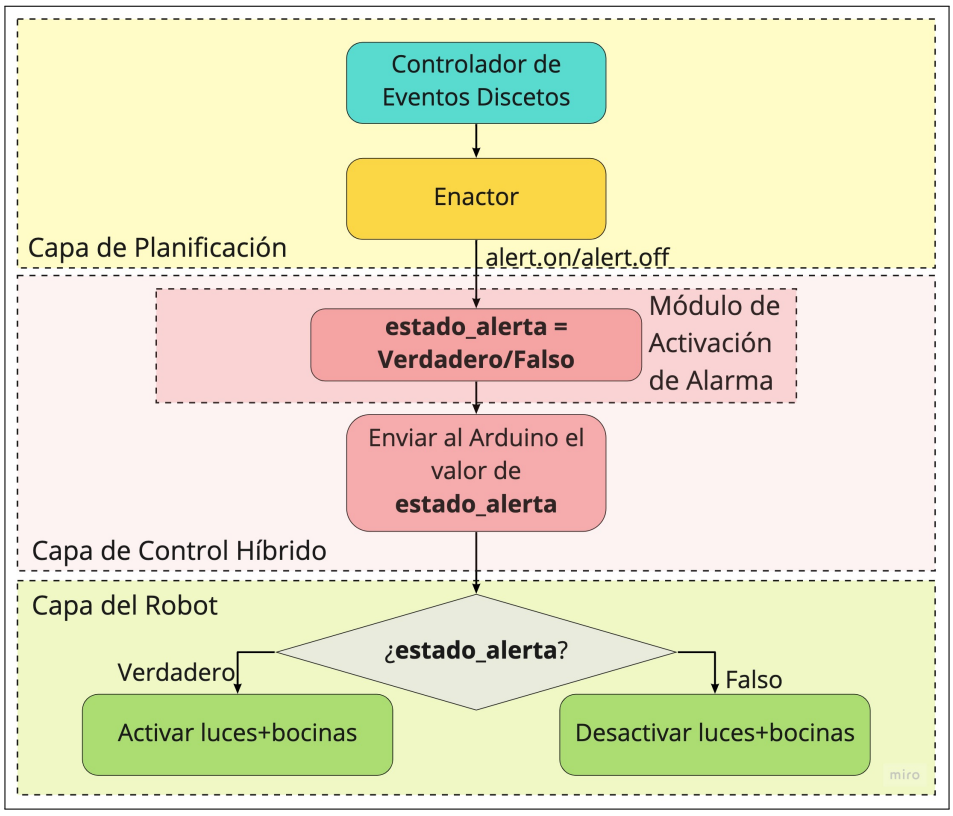


Figura 8.2: Descripción del Módulo de Alerta

8.3. Módulo de Ordenamiento de Cajas

La tercera tarea que se propone para el robot es el ordenamiento de una caja en el espacio de trabajo. Esta misión implica que el robot reconozca la posición de una caja y que planifique cómo llevarla a una posición deseada. Dado que el robot solo puede empujar la caja para conseguir moverla, es necesario sincronizar la secuencia de eventos que hacen que el robot mueva la caja para que no ocurran otros eventos antes de lograr ubicar la caja. Esto significa que, si la caja está en la posición por ejemplo *(0,1)*, el

robot podrá moverla solo si ocurre el evento *go.0,1.*, es decir, el robot deberá ir a la celda ocupada por la caja. Sin embargo, como el robot debe llevar la caja a la siguiente celda, a la *(0,2)* si el robot se aproxima desde la celda *(0,0)*, debe asegurarse que la caja queda contenida en la celda de destino antes de que ocurra el evento *at.0.1*. Esto es porque el controlador de eventos discretos espera la ocurrencia del evento *goBox.0.2* inmediatamente después de la acción *go.0.1*, y luego del evento *goBox.0.2* recién debe ocurrir el evento *at.0.1*. Debido a esta secuencia es que el módulo de ordenamiento de cajas debe «bloquear» la ocurrencia de un evento *at.i.j* hasta que ocurra el evento *goBox.0.2*.

Por lo comentado en el párrafo anterior es que el módulo híbrido realizará el siguiente algoritmo de control:

1. cuando el robot recibe como referencia la celda que actualmente ocupa la caja, debe activarse este módulo y bloquear la ocurrencia de cualquier evento *at.i.j*;

2. el módulo debe identificar desde qué lado se aproxima el robot para establecer las coordenadas de la celda objetivo de la caja (i_{boxRef}, j_{boxRef});

3. se calcula la distancia que hay desde el centro de la caja hasta el centro de la celda (i_{boxRef}, j_{boxRef});

4. el robot continúa avanzando tomando como referencia la celda que actualmente ocupa la caja. De esta manera el robot siempre buscará ocupar la posición de la caja y continuará empujándola indefinidamente;

5. cuando la distancia entre la caja y la celda objetivo sea menor a 15 px en la imagen capturada por la webcam, se agrega el evento $goBox.i_{boxRef}.j_{boxRef}$ a la lista de eventos;

6. una vez agregado el evento $goBox.i_{boxRef}.j_{boxRef}$, se desbloquea la ocurrencia de los eventos *at.i.j* y se cambia la referencia del robot por la referencia original que se corresponde con la posición inicial de la caja.

En la Fig. 8.3 se muestra la secuencia de control anterior en un diagrama de flujo. De esta imagen se ve que luego de que la caja es ubicada en la posición final, con coordenadas $(i_{boxRef}.j_{boxRef})$, se hace un llamado al Módulo de Desplazamiento y este llevará al robot a la posición correspondiente en la celda *(i,j)*.

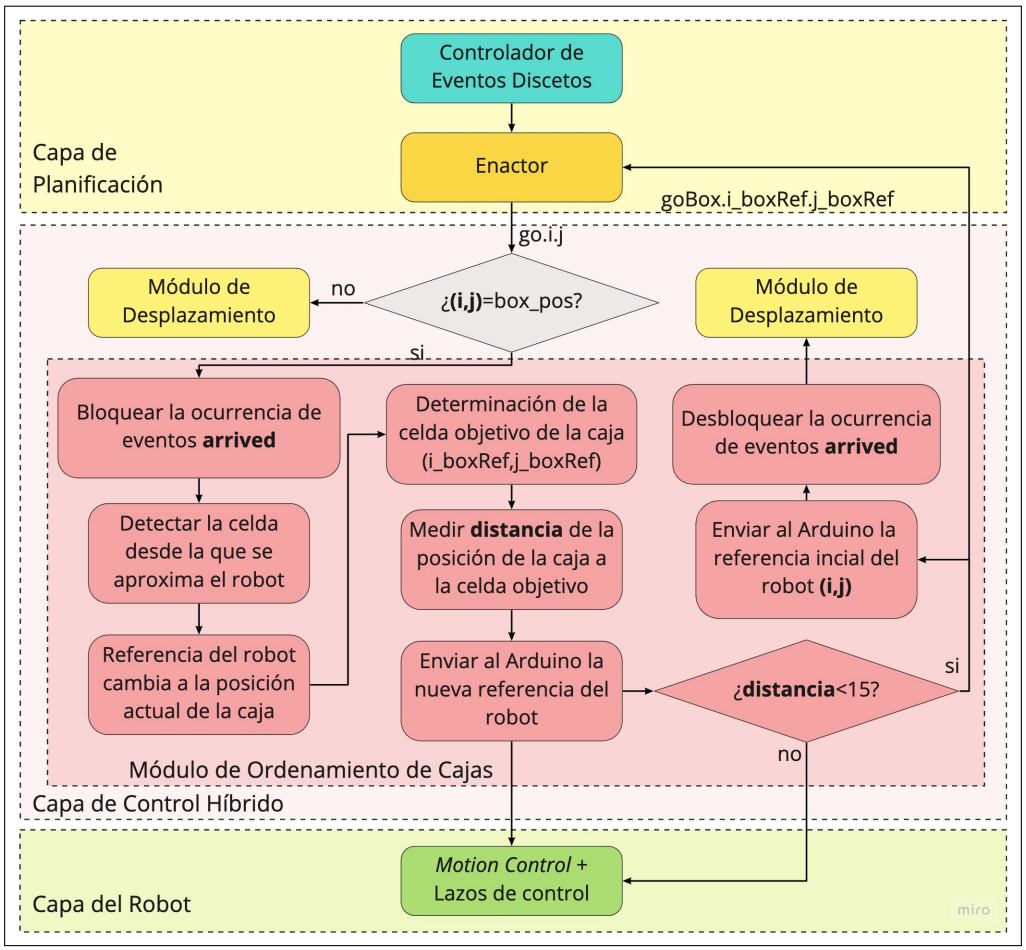


Figura 8.3: Descripción del Módulo de Ordenamiento de Cajas

8.4. Integración de los Módulos Híbridos

El robot está compuesto por una placa Raspberry Pi y una Arduino UNO que se comunican entre sí por conexión serie. Mientras los procesos de conexión vía WiFi con la LAPTOP y el procesamiento de imágenes se realiza en la Raspberry, el *Motion Control* y los lazos de control se ejecutan en el microprocesador de la placa Arduino, según se indicó en la Sección 6.3. Para llevar a cabo todas las tareas de monitoreo y control, se implementaron en total cuatro procesos que se ejecutan en paralelo durante el funcionamiento del robot, aprovechando el procesador de cuatro núcleos de la Raspberry Pi.

El **Proceso I**, es el primer proceso implementado y se encarga de la inicialización de las conexiones serie entre el Arduino y la Raspberry y de la conexión WiFi con la LAPTOP. Este proceso sintetiza el controlador discreto en función de las especificaciones precargadas y de la configuración inicial del escenario donde operará el robot, detectando la posición inicial del robot, los obstáculos y la caja que eventualmente el robot debería ordenar. El tiempo que el programa tardará en generar un plan, dependerá de

la misión que se desea llevar a cabo. Por ejemplo, para una misión de patrullaje con detección de obstáculos, el tiempo que tarda el programa MTSA en sintetizar el controlador es de aproximadamente 2 s, mientras que para generar el plan para la misión de ordenamiento de una caja se vio que se tarda aproximadamente 39 s. Una vez finalizada la etapa de incialización se ejecuta el ENACTOR, el cual recibe como parámetro de entrada una lista de funciones que implementan los módulos híbridos para que sea capaz de llamar las acciones controlables del modelo.

El segundo proceso (**Proceso II**) es el encargado del procesamiento de las imágenes que la Raspberry recibe vía WiFi desde la LAPTOP. Este proceso recibe como *input* la imagen capturada por la WEBCAM y devuelve la posición y orientación del robot y la posición de la caja. Por otro lado, este proceso envía al Arduino la referencia que debe alcanzar el robot dado por las coordenadas *(i,j)* que le indica el Módulo de Desplazamiento o el de Ordenamiento de Cajas, según corresponda.

En tercer lugar, el **Proceso III** se encarga de la comunicación serie entre el Arduino y la Raspberry. Este proceso escucha continuamente los mensajes que envía el Arduino referidos al evento de arribo del robot a la posición final según la estrategia de *Motion Control* diseñada.

Finalmente, cuando el ENACTOR envía la acción *go.i.j*, el **Proceso IV** detecta si las coordenadas *(i,j)* de dicha acción coinciden o no con la posición que actualmente ocupa la caja. En caso afirmativo se activa el Módulo de Ordenamiento de Cajas, que monitorea continuamente la posición del robot y de la caja, y activa la lógica que permite el ordenamiento de la caja, determinando las referencias adecuadas. El Módulo de Ordenamiento de Cajas se encarga de sincronizar las acciones *goBox.h.k* y *at.i.j*, de manera que cuando la caja ha llegado a su posición se envía al ENACTOR el evento *goBox.h.k*, y luego, este módulo, llama al Módulo de Desplazamiento, el cual fijará las coordenadas *(i,j)* originales, para dar lugar a la ocurrencia del evento *at.i.j*. Un esquema básico de cómo se comunican los distintos procesos se muestra en el diagrama de flujo de la Fig. 8.4[1].

El software se programó usando como referencia los diagramas de flujo y arquitectura de componentes propuestos, pero debido a restricciones de tiempo se alcanzó una calidad de prototipo en el mismo. Como se tuvo que programar múltiples procesos en la Raspberry con comunicación entre los mismos, con el Arduino y el servidor de imágenes, se tornó un problema de programación concurrente relativamente complejo y el diagrama mostrado en la Fig. 8.4 simplifica varias relaciones entre módulos y procesos, en particular debido al uso de memoria compartida. Una de las principales diferencias con el diseño arquitectónico propuesto en la Fig. 5.3 es que se logró una implementación unificada de *Motion Control* que se puede emplear por distinta manera por los

[1]Los valores de tiempos de comunicación y procesamiento de varios de los procesos involucrados no pudieron ser medidos por no contar con el robot real durante la emergencia sanitaria del COVID-19.

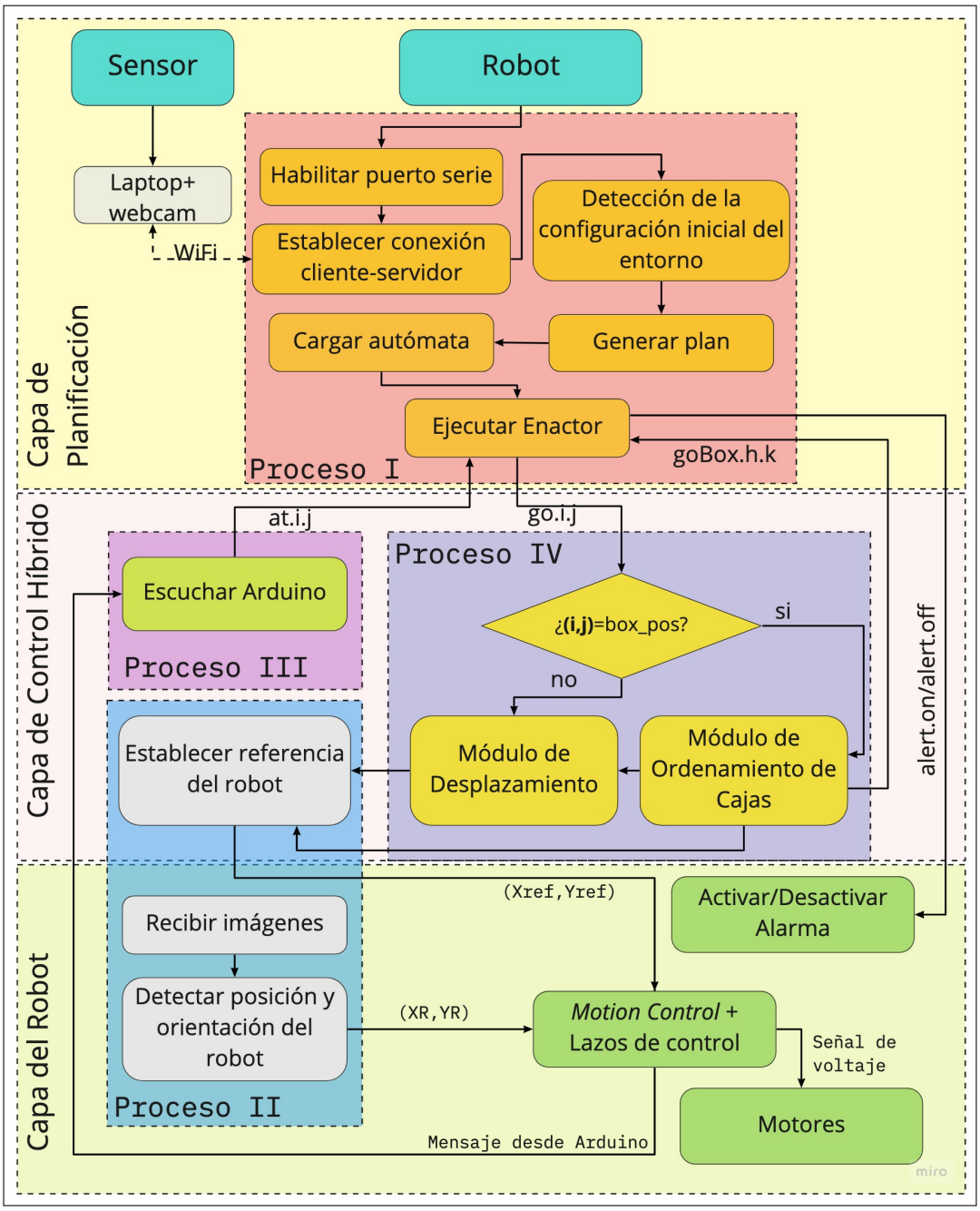


Figura 8.4: Esquema de integración de los módulos híbridos

Módulos de Ordenamiento y de Desplazamiento para lograr el *Motion Control:Robot* y *Motion Control:Caja*. Además, se tomó la decisión de implementar este módulo en el Arduino debido a su estrecho vínculo con los lazos de control, con lo cual en la Fig. 8.4 se lo ubicó en la Capa del Robot a pesar de sus claros componentes híbridos. Queda para trabajo futuro lograr un diseño e implementación de software más cuidadoso en esta instancia para generar mayor robustez en el sistema robótico.

Capítulo 9

Resultados

> *"La inteligencia consiste no sólo en el conocimiento, sino también en la destreza de aplicar los conocimientos en la práctica".*
>
> — Aristóteles, filósofo y polímata de la Antigua Grecia

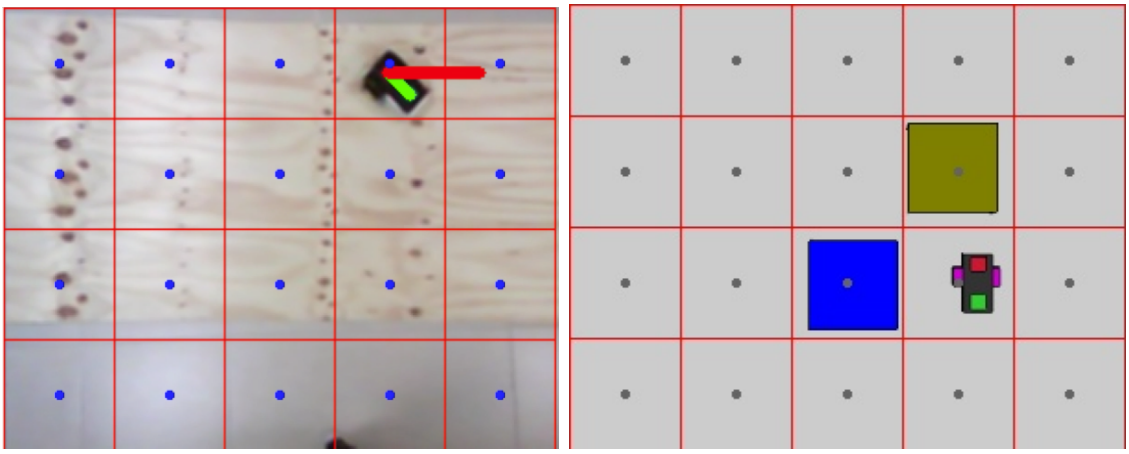
En este capítulo se exponen los resultados obtenidos que validarán el proceso de diseño e implementación propuesto para la construcción del robot Tom. Las pruebas de validación fueron realizadas en parte en condiciones de laboratorio con el robot real y en parte en un entorno de simulación desarrollado para tal fin[1]. Se adjuntan videos y más fotos de las pruebas realizadas como material complementario del proyecto.

9.1. Descripción del Sistema

El espacio de trabajo del robot está limitado por el campo de visión de la cámara web montada en la parte superior del laboratorio, como se muestra en la Fig. 6.1. La imagen obtenida con la webcam tiene una resolución de 320×240 píxeles y abarca un área de aproximadamente $2,1 \times 1,60$ metros. Teniendo en cuenta las dimensiones de la imagen con la que se trabajará y el radio mínimo de giro que se halló en la Sección 7.1.3 (de 15 píxeles), se determinó que el tamaño de grilla óptimo para trabajar es de 60×60 píxeles, lo que resulta en que el espacio de trabajo quede dividido en 4×5 celdas. Las celdas se indicarán a partir del valor 0 y en primer lugar se especifica la fila y luego la columna, de modo que la celda superior izquierda se indica con las coordenadas *(0,0)* y la inferior derecha con *(3,4)*. En la Fig. 9.1 se muestra la imagen capturada por la cámara real y la del entorno virtual, en la que las cajas azules representan los obstáculos del entorno y el bloque marrón que se ve en el entorno simulado la caja que el robot deberá ordenar en alguna de sus misiones. Cabe mencionar que el simulador del robot

[1] La posibilidad de continuar ensayando las misiones en un entorno real se vio interrumpida por la emergencia sanitaria del COVID-19.

fue desarrollado exclusivamente para intentar reconstruir el sistema real dado que no fue posible continuar con las pruebas en condiciones de laboratorio, por lo que no forma parte del alcance del presente trabajo y no se brindarán detalles en profundidad sobre cómo funciona[2]. El simulador fue implementado utilizando la librería *Panda3d* de Python para la visualización 3D y como motor de física [35], y fue diseñado para integrar los códigos ya desarrollados aplicando algunas leves modificaciones, compilando el código del Arduino para ejecutarlo como un proceso independiente y simular el Arduino y su comunicación serial con el resto de los procesos. El simulador cuenta con la misma planta identificada del sistema, la cual trata de simular el comportamiento del sistema real.



(a) Imagen capturada por la webcam en el entorno real

(b) Imagen capturada por la webcam en el entorno simulado

Figura 9.1: Captura de imágenes de los espacios de trabajo real y simulado

9.2. Validación de Misiones del Robot

En esta sección se muestran las pruebas realizadas para un conjunto de misiones correspondientes a las clasificaciones de *Core Movement Patterns*, *Avoidance* y *Reaction*, según se comentó en el Capítulo 4.

9.2.1. Misiones de Patrulla

Patrulla Simple

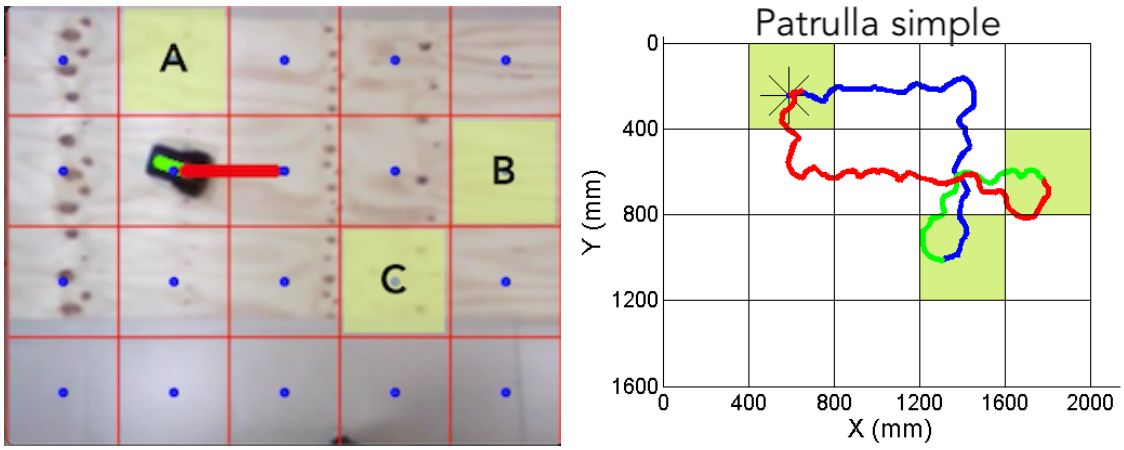
La primera misión de patrullaje que se realizó consiste en que el robot visite tres zonas distintas del espacio por tiempo indefinido en un entorno libre de obstáculos. En la Fig. 9.2a se muestra una imagen real del robot en un instante arbitrario con las tres

[2]El simulador fue desarrollado en conjunto con los responsables del proyecto para poder continuar ensayando distintas misiones para el robot.

zonas por las que patrullará (A, B y C), y en la Fig. 9.2b la trayectoria que realizó para lograr la misión. Para esta misión, el robot parte de la celda *(0,1)* indicada con un asterisco en la Fig. 9.2b, y cada tramo de la trayectoria del robot se indica con un color diferente, comenzando por el camino indicado en azul. Para la especificación de esta misión, se necesitan definir las acciones controlables *go* y *arrived* (*at*) y la descripción de la misión en FLTL se determina de manera similar a lo presentado en la Sección 4.1.3 mediante la siguiente fórmula lógica:

$$\varphi_1 = (\square\Diamond at.0.1) \wedge (\square\Diamond at.1.4) \wedge (\square\Diamond at.2.3) \tag{9.1}$$

En el link [36] se presenta un video de la misión realizada por el robot real.



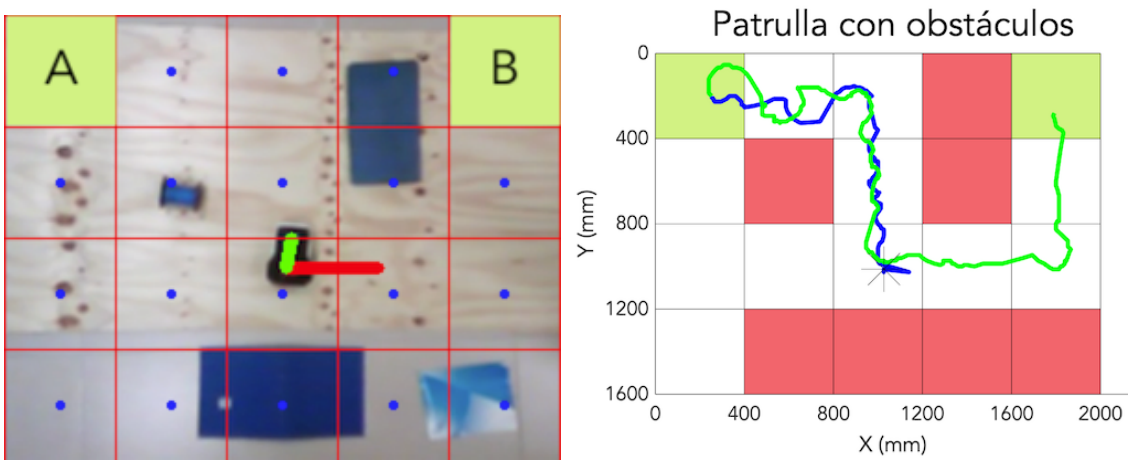
(a) Imagen capturada por la webcam en el entorno real

(b) Gráfico de la trayectoria

Figura 9.2: Misión de patrullaje entre tres posiciones sin obstáculos

Patrulla con Detección de Obstáculos

En segundo lugar, se realizó una prueba de patrullaje del robot en un entorno con obstáculos que el robot debe evitar. La fórmula lógica en FLTL que describe la misión de patrullaje del robot es similar a la ecuación 9.1 y se definen además requerimientos de *safety* φ_2 que prohíben que el robot entre a una celda que contiene un obstáculo, como se explicó en la Sección 4.1.3. La Fig. 9.3a muestra el escenario para el caso en que el robot debe patrullar entre las regiones A y B esquivando los obstáculos identificados en color azul. En este caso, el robot parte de la posición *(2,2)* e inicialmente se dirige hacia la celda *(0,0)* (trayectoria azul) y luego hacia la celda *(0,4)* (trayectoria verde), cumpliendo la misión consignada. En [37] se deja una grabación realizada de esta misión.



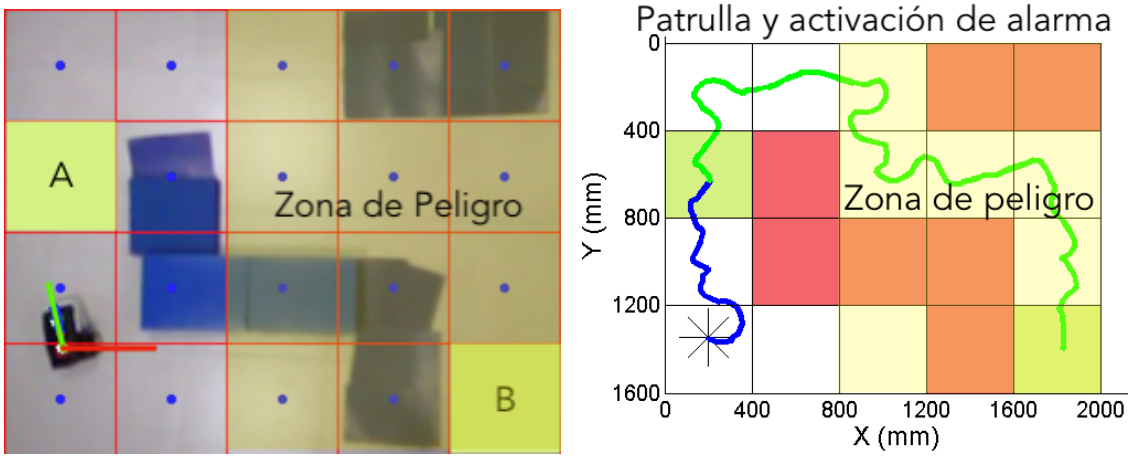
(a) Imagen capturada por la webcam en el entorno real

(b) Gráfico de la trayectoria

Figura 9.3: Misión de patrullaje en un entorno con obstáculos

Patrulla con Detección de Obstáculos y Señal de Alerta

Es importante destacar que no todas las misiones que se pueden planificar deben implicar movimiento. La técnica de *planning* es una herramienta general que permite planificar el comportamiento de los sistemas en múltiples aspectos. Por tal motivo, en este trabajo se quiso evidenciar el potencial de esta técnica incorporando capacidades al robot que no impliquen únicamente movimiento, por ejemplo la capacidad de encender una alarma sonora y luminosa cuando se ingresa a una región. En la Fig. 9.4 se muestran los resultados obtenidos para esta misión en la que se identifica como «zona de peligro» a la región amarilla, zona en la que el robot activa la señal de alerta. Los resultados de esta misión también se presentan como grabaciones en el link [38].



(a) Imagen capturada por la webcam en el entorno real

(b) Gráfico de la trayectoria

Figura 9.4: Misión de patrullaje y activación de un sistema de alerta en un entorno con obstáculos

9.2.2. Misiones de Ordenamiento de Cajas

A continuación se presentan escenarios más complejos en los que el robot debe ser capaz de ordenar una caja llevándola a una posición específica tanto en un espacio libre de obstáculos como en uno donde debe evitarlos. Esta tarea resulta de interés dado que uno podría pensar en implementar este tipo de tareas, por ejemplo, en un depósito donde el robot debe despejar una zona del camino o bien llevar alguna caja de mercadería a alguna ubicación en particular. Las dos misiones que se presentan fueron realizadas en el espacio de trabajo simulado.

Ordenamiento de una Caja en un Entorno sin Obstáculos

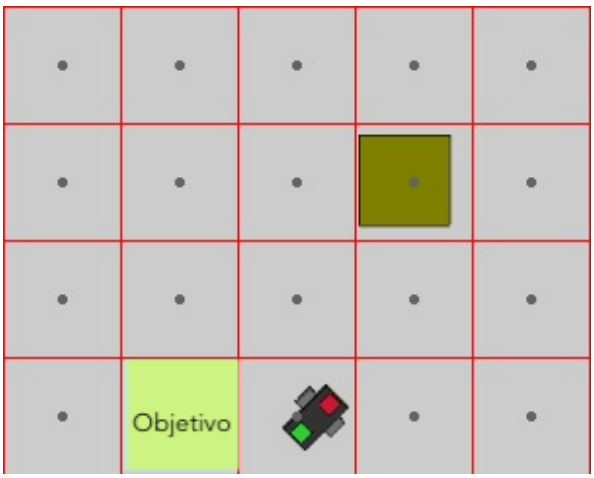


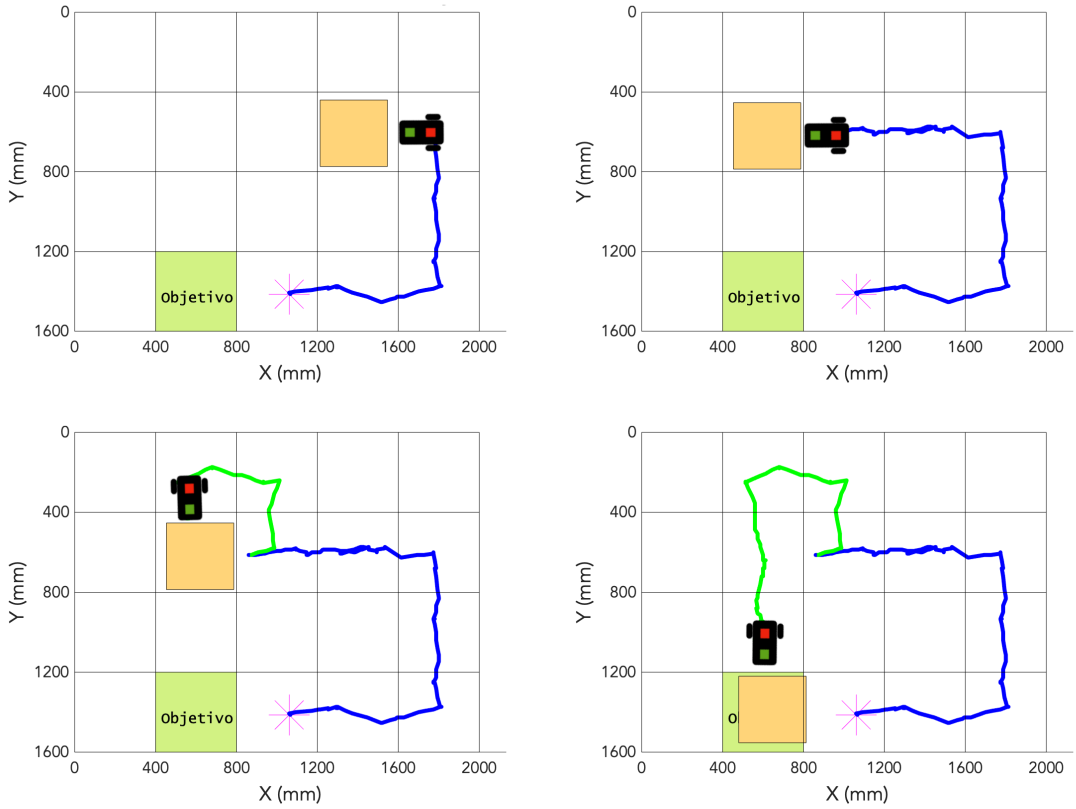
Figura 9.5: Imagen capturada por la webcam en el entorno simulado

El primer caso que se presenta es el ordenamiento de una caja en un entorno libre de obstáculos. En esta misión, el robot debe empujar la caja y llevarla a la posición indicada en verde en la Fig. 9.5. Como puede verse en la Fig. 9.6a, la secuencia que realiza el robot para posicionar la caja se puede dividir en cuatro etapas. Primero, el robot se acerca hasta la caja desde la posición *(1,4)* y la empuja hasta llevarla a la posición *(1,1)*. Luego, rodea la caja posicionándose en la celda *(0,1)* y finalmente empuja la caja hasta llevarla al objetivo. La especificación de esta misión en FLTL se presentó en la Sección 4.1.5, y para este caso en particular se escribe como:

$$\varphi = \Box\Diamond goBox.3.1 \tag{9.2}$$

En [39] se muestra un video de esta misión. En este video se evidencia lo mencionado en la Sección 8.3 referido a la secuencia que realiza el robot para empujar la caja a la celda siguiente. Siguiendo con el algoritmo descripto, el proceso de empujar la caja a la celda contigua comienza con el robot dirigiéndose a la celda *(i,j)* que ocupa la caja. Acto seguido, el módulo de ordenamiento de cajas bloquea la ocurrencia de un evento

at.i.j y el robot recibe como referencia la posición de la caja hasta que logra ubicarla en el centro de la celda destino. Una vez que el robot ubicó la caja en la posición deseada, el módulo híbrido de ordenamiento de cajas «libera» la ocurrencia de un evento *at.i.j* y el robot retoma la instrucción *go.i.j*, inicialmente indicada por el ENACTOR. Esta secuencia se traduce en un comportamiento en el que el robot avanza en dirección a la caja y, cuando la deja en la celda contigua, este vuelve al centro de la celda *(i,j)* que inicialmente ocupaba la caja y luego continúa con el plan. La implementación de un algoritmo que permita continuar en línea recta el movimiento del robot empujando la caja, sin que este deba regresar a la celda *(i,j)* inicial, se deja para un estudio posterior.



(a) Secuencia de la trayectoria del robot

Figura 9.6: Misión de ordenamiento de una caja en un entorno libre de obstáculos

Ordenamiento de una Caja en un Entorno con Obstáculos

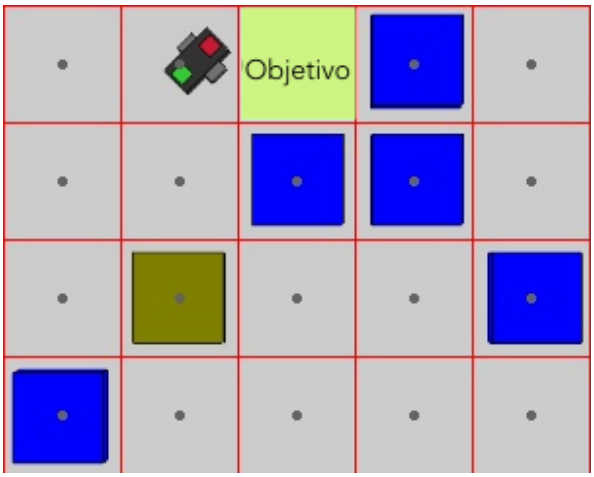


Figura 9.7: Imagen capturada por la webcam en el entorno simulado.

Cuando se agregan obstáculos al problema anterior, se pueden conseguir escenarios más complejos e interesantes. En la Fig. 9.7 se muestra el caso para el entorno simulado en el que la caja debe ser llevada a la posición *(0,2)*, ubicación que se encuentra rodeada por obstáculos. Para ello, el robot inicialmente debe identificar cuál caja puede moverse y cuáles no. En la Fig. 9.8 se tiene la secuencia que realizó el robot para llevar la caja a la celda de destino. Se puede apreciar que al principio el robot está «encerrado» por las cajas, y para resolver el problema primero debe empujar la caja marrón hacia la derecha para poder ubicarse en las filas inferiores y desde ahí poder empujarla hasta su destino. Cada vez que el robot se aproxima a la caja y la desplaza en línea recta se indica en la figura con una línea de distinto color para mayor claridad. Vale la pena destacar que el plan que resuelve esta misión fue sintetizado automáticamente a partir de un buen modelo del entorno (Sección 4.1.5), simplemente agregando la especificación

$$\varphi = \square\Diamond goBox.0.2 \tag{9.3}$$

En [40] se muestra un video del robot llevando a cabo el plan que resuelve esta misión en el entorno simulado.

9.2.3. Misiones Extra

Como instancia final, se diseñaron dos escenarios en los cuales el robot debe: 1) alcanzar una región, para lo cual debe esquivar obstáculos y mover una caja que se encuentra en medio del camino, y 2) ubicar una caja en un escenario sencillo pero que implica realizar múltiples trayectos dada la ubicación estratégica de los obstáculos.

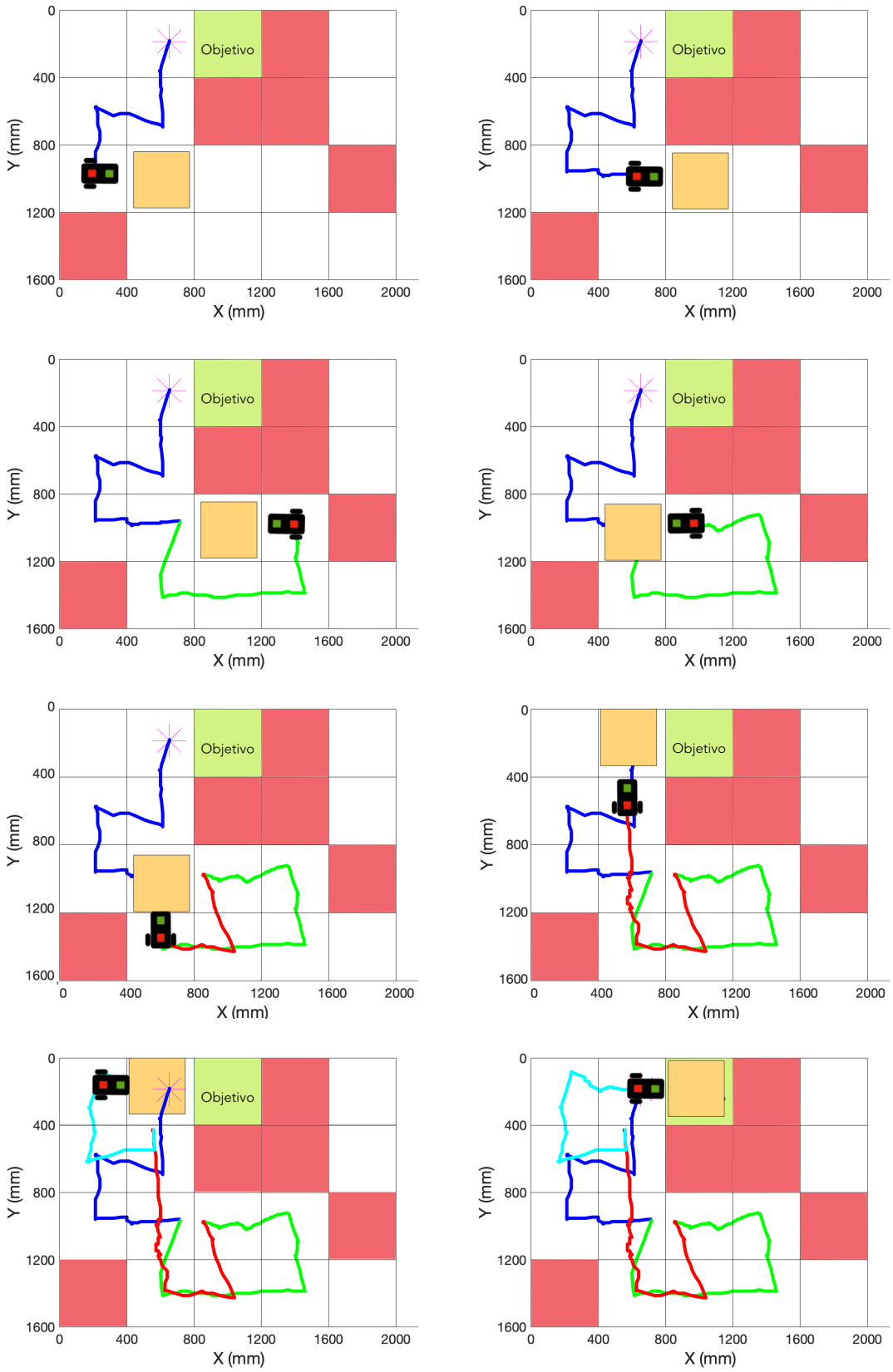


Figura 9.8: Secuencia de la trayectoria del robot

Misión Extra: 1

El escenario para la primera misión extra es el que se muestra en la Fig. 9.9. En esta misión el robot está «encerrado» y debe alcanzar la posición indicada en verde en la imagen. El robot planifica la misión teniendo en cuenta que una de las cajas que bloquea su camino puede moverse si la empuja. Sin embargo, al moverla, otra parte del camino que debe realizar el robot para llegar a su objetivo queda nuevamente bloqueado (cuando la caja ocupa la posición *(2,3)*). Finalmente, el robot empuja la caja ubicándola en la posición *(1,3)* y alcanza la celda objetivo *(2,4)* cumpliendo con su misión, como se muestra en la secuencia de la Fig. 9.10. El video de esta misión puede verse en [41].

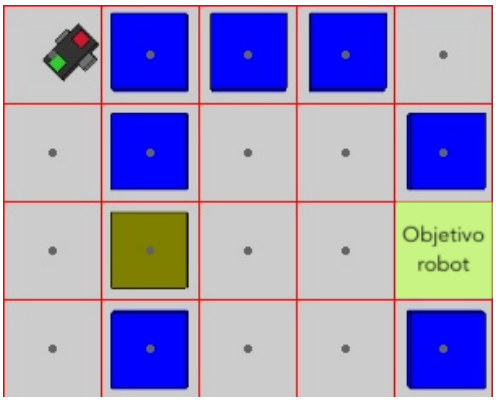
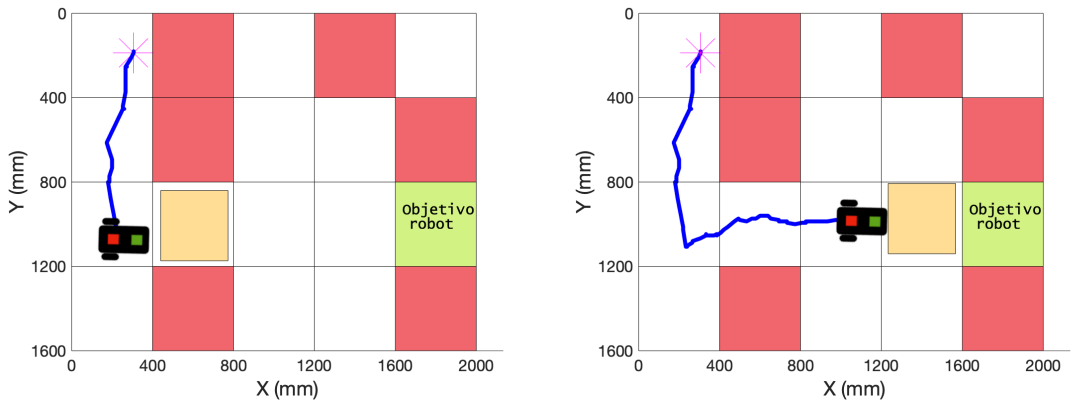


Figura 9.9: Imagen capturada por la webcam en el entorno simulado



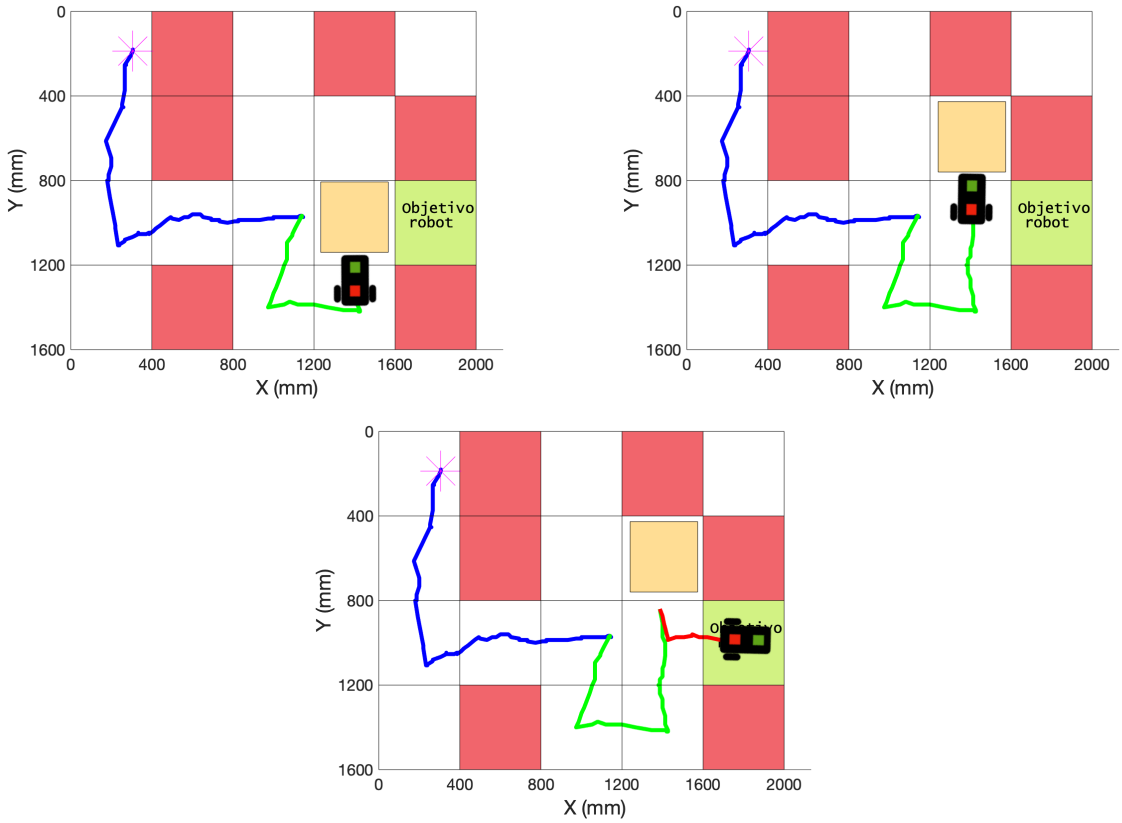


Figura 9.10: Secuencia de la trayectoria del robot

Misión Extra: 2

Esta segunda misión extra es una variante a la presentada entre las tareas de ordenamiento de una caja. Sin embargo, proponer distintas configuraciones para el robot y los obstáculos ayuda a validar el comportamiento del robot en diversos escenarios. Nuevamente en este caso el robot debe llevar la caja al casillero pintado de verde (ver Fig. 9.12) y el video de esta misión se encuentra subido en [42].

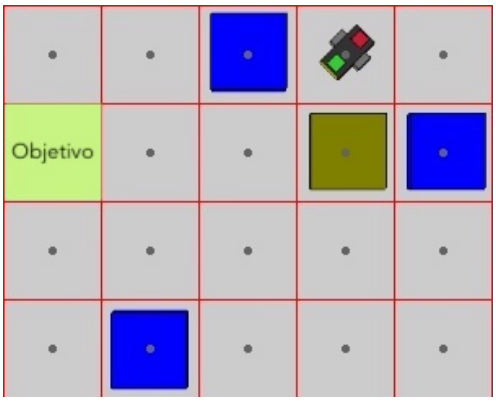


Figura 9.11: Imagen capturada por la webcam en el entorno simulado

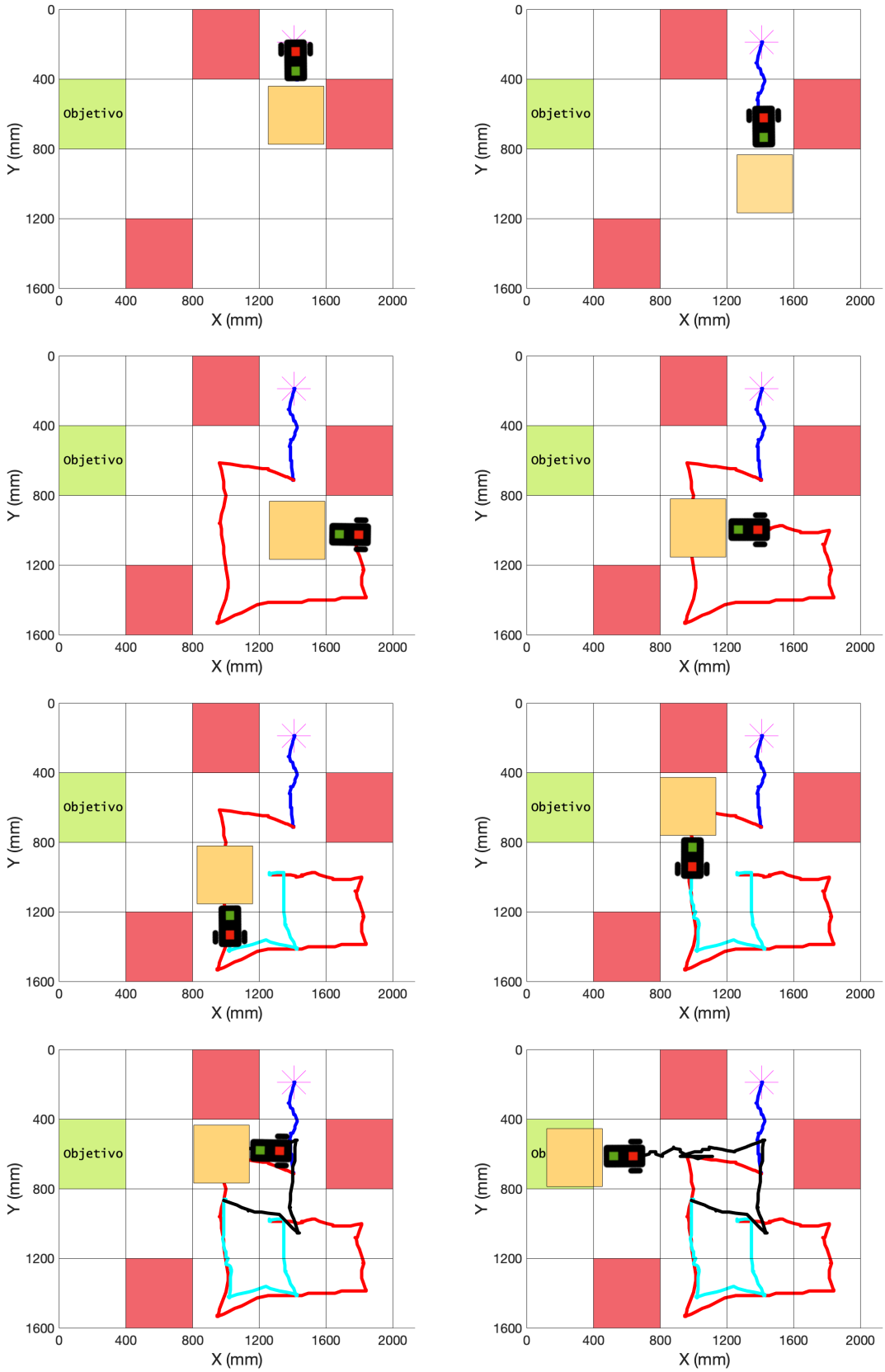
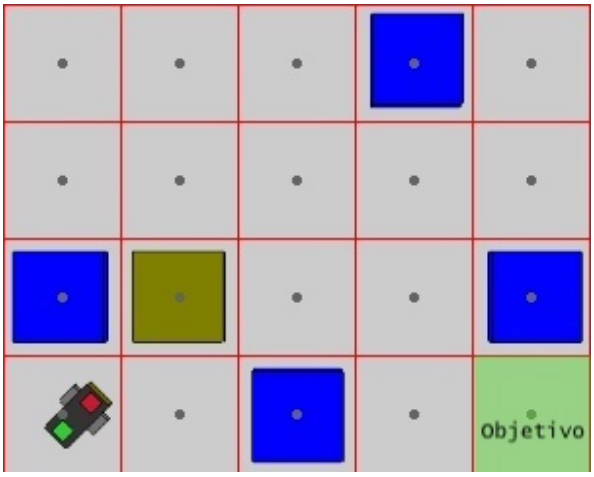


Figura 9.12: Secuencia de la trayectoria del robot

9.2.4. Escenario sin Solución

La herramienta de síntesis es utilizada para hallar controladores que satisfagan el modelo junto con las especificaciones que un usuario quiere que el sistema cumpla, pero también sirve para determinar la no existencia de dichos controladores. En el escenario de la Fig. 9.13a se muestra un caso no trivial para el que no es posible encontrar una estrategia que resuelva el problema. En esta misión el robot debe ser capaz de ubicar la caja en la celda inferior derecha, sin embargo esto no es realizable dada la ubicación de los obstáculos. La herramienta de síntesis procesa este escenario y despliega el error de la Fig. 9.13b indicando que no es posible hallar un controlador que satisfaga las especificaciones para este modelo.



(a) Imagen capturada por la webcam en el entorno simulado

```
Composition:
Controller = Environment || Never32 || Never03 || Never31 || Never21 || Never24 || Never20
State Space:
 3758 * 1 * 1 * 1 * 1 * 1 = 2 ** 12
Composing...
-- States: 3 Transitions: 4 Memory used: 20176K
Composed in 25ms
Solving the LTS control problem.
Environment states: 4
Game states: 4
There is no controller for model Controller for the given setting.
Analysis time: 32ms.
```

(b) Mensaje de error desplegado por la herramienta de síntesis

Figura 9.13: Escenario sin solución

Capítulo 10

Conclusiones y Trabajo Futuro

> *"Solo podemos ver poco del futuro, pero lo suficiente para darnos cuenta de que hay mucho que hacer".*
>
> — Alan Turing, criptógrafo británico considerado el «Padre de las Ciencias de la Computación»

De este trabajo de tesis se concluye que la metodología propuesta de diseño de un robot basado en controladores híbridos pudo ser usada y validada exitosamente mediante pruebas llevadas a cabo en el laboratorio con la plataforma robótica real y en un entorno simulado. El robot logró cumplir con las tareas consignadas en cada una de las pruebas, desde las misiones de patrullaje en un entorno libre de obstáculos, hasta escenarios más complejos en los que el robot debía empujar una caja hasta colocarla en una determinada posición esquivando obstáculos. Además, se verificó el potencial que tiene la herramienta de síntesis para planificar tanto misiones de navegación como la planificación de tareas, como por ejemplo encender una alarma o mover una caja a una posición determinada.

Para el desarrollo de este trabajo fue realmente útil adquirir conocimientos del lenguaje de programación Python v3.7, el cual permitió reducir los tiempos de implementación e integración de los módulos híbridos y de desarrollo de algoritmos de procesamiento de imágenes notablemente. Este tipo de lenguajes interpretados y de alto nivel aportan gran flexibilidad a la hora de desarrollar programas en los cuales no hay requerimientos muy exigentes en los tiempos de ejecución. Por otra parte, se estudió la sintaxis del software MTSA para la síntesis de los controladores de eventos discretos, y se vio que este lenguaje resultó muy conveniente para generar los planes de las misiones del robot a partir de un modelo y especificaciones basadas en la idea de «qué misión debe cumplir el sistema» y no en «cómo lo realizará».

La metodología de trabajo que se llevó a cabo en este proyecto representa un enfoque alternativo al método tradicional para abordar este tipo de proyectos. La idea de partir de las capas de mayor grado de abstracción permite ciertas garantías en el

proceso de diseño dado que el trabajo queda estructurado en función del objetivo que se quiere lograr. En las últimas décadas, se han producido grandes avances en la interacción combinada de *planning* y sistemas robóticos, por lo que, metodologías como esta, están comenzando a emerger como una estrategia razonable para ciertos escenarios que buscan garantías, sobre todo en entornos con posibilidades de fallas (ver [6]). Por este motivo, el resultado de este proyecto representa un aporte para las comunidades de robótica como un medio para sistematizar el proceso de diseño e implementación de algoritmos de *planning* en sistemas robóticos exclusivamente diseñados para cumplir el conjunto de misiones que se propone. Esto permite, entre otras cosas, adecuar el hardware y los controladores del robot a los requerimientos exigidos en las etapas de planificación y diseño de los módulos híbridos, evitando la necesidad de implementar controladores de movimiento complejos a plataformas preexistentes.

Se deja como propuesta de trabajo futuro el desarrollo de controladores de ángulo y posición que le brinden al robot un desplazamiento más «natural» en el entorno. Con un controlador de posición más sofisticado, se espera lograr reducir el tamaño de la celda de la discretización, lo que daría lugar a la posibilidad de realizar misiones más complejas. Por otro lado, resultaría interesante estudiar un escenario multirobot donde se planifiquen misiones en cooperación, ver [43], o de captura como en [44]. Agregar capacidades al robot permitiría expandir el rango de misiones que podrá realizar el robot. Por ejemplo, incorporar un brazo robótico en la plataforma robótica permitiría agarrar objetos o apretar botones para comandar algún sistema externo, como un tablero eléctrico. Otra alternativa que se propone es la incorporación de sistemas de detección de posición y ángulo basado en sensores de inercia, GPS o triangulación. Mediante este remplazo de sensores se lograría que el robot no dependa de una computadora personal y una cámara web lo que resultaría en una unidad más independiente y compacta. Finalmente, la incorporación de una cámara montada sobre la misma plataforma robótica podría utilizarse para percibir el entorno y para la localización del robot mediante técnicas más sofisticadas como SLAM o Lidar.

Bibliografía

[1] Škrjanc, I. Wheeled Mobile Robotics: From Fundamentals Towards Autonomous Systems. 2017. 1, 2

[2] Bin, X., Lei, G., Shimin, W., Yuan, S., Ying, Z. Dynamics modeling and system parameter identification experiment of a kind of two-wheeled robot. En: 2015 IEEE International Conf. on Information and Automation, págs. 404–408. 2015. 2

[3] Belta, C., Bicci, A., Egerstedt, M., Frazzoli, E., Klavins, E., Pappas, G. Symbolic control and planning of robotic motion [grand challenges of robotics], 03 2007. 2, 3, 4, 15

[4] Finucane, C., Gangyuan Jing, Kress-Gazit, H. Ltlmop: Experimenting with language, temporal logic and robot control. En: 2010 IEEE/RSJ International Conference on Intelligent Robots and Systems, págs. 1988–1993. 2010. 2, 4, 15, 19

[5] Baier, C., Katoen, J.-P. Principles of Model Checking, tomo 26202649. 2008. 2, 9, 10, 12

[6] Maniatopoulos, S., Schillinger, P., Pong, V., Conner, D. C., Kress-Gazit, H. Reactive high-level behavior synthesis for an atlas humanoid robot. En: 2016 IEEE International Conf. on Robotics and Automation (ICRA), págs. 4192–4199. 2016. 2, 15, 80

[7] Kress-Gazit, H., Fainekos, G. E., Pappas, G. J. Temporal-logic-based reactive mission and motion planning. *IEEE Trans. on Robotics*, págs. 1370–1381, 2009. 2

[8] Kress-Gazit, H., Fainekos, G., Pappas, G. Translating structured english to robot controllers. *Advanced Robotics*, **22**, 1343–1359, 10 2008. 2, 3, 4, 15, 16, 23

[9] Holzmann, G. The Spin Model Checker: Primer and Reference Manual. 2004. 2

[10] Cimatti, A., Clarke, E., Giunchiglia, E., Giunchiglia, F., Pistore, M., Roveri, M., *et al.* Nusmv 2: An opensource tool for symbolic model checking. *14th International Conference, CAV, Copenhagen, Denmark*, 01 2002.

[11] Fainekos, G. E., Kress-Gazit, H., Pappas, G. J. Temporal logic motion planning for mobile robots. En: Proceedings of the 2005 IEEE International Conference on Robotics and Automation, págs. 2020–2025. 2005. 2, 3, 4, 15, 16, 19

[12] Shah, D. Path planning for mobile robots using potential field method. 2018. 2

[13] Kingston, Z., Moll, M., Kavraki, L. E. Sampling-based methods for motion planning with constraints. *Annual Review of Control, Robotics, and Autonomous Systems*, **1** (1), 159–185, 2018. URL https://doi.org/10.1146/annurev-control-060117-105226. 3

[14] Wolff, E. M., Topcu, U., Murray, R. M. Automaton-guided controller synthesis for nonlinear systems with temporal logic. En: 2013 IEEE/RSJ International Conference on Intelligent Robots and Systems, págs. 4332–4339. 2013. 3

[15] Conner, D., Rizzi, A., Choset, H. Composition of local potential functions for global robot control and navigation. tomo 4, págs. 3546 – 3551 vol.3. 2003.

[16] D'Ippolito, N. Synthesis of event-based controllers: A software engineering challenge. En: 2012 34th International Conference on Software Engineering (ICSE), págs. 1547–1550. 2012. 3, 9, 11, 12, 19

[17] Jing, G., Tosun, T., Yim, M., Kress-Gazit, H. An end-to-end system for accomplishing tasks with modular robots: Perspectives for the ai community. págs. 4879–4883. 2017. 4, 15

[18] DeCastro, J. A., Raman, V., Kress-Gazit, H. Dynamics-driven adaptive abstraction for reactive high-level mission and motion planning. En: 2015 IEEE International Conference on Robotics and Automation (ICRA), págs. 369–376. 2015. 4

[19] Wing, J. A specifier's introduction to formal methods. *Computer*, **23**, 8, 10–22, 24, 10 1990. 9

[20] Keller, R. Formal verification of parallel programs. *Communications of the ACM*, **19**, 371–384, 07 1976. 9

[21] Giannakopoulou, D., Magee, J. Fluent model checking for event-based systems. ESEC/FSE-11, págs. 257–266. New York, NY, USA: ACM, 2003. 11

[22] D'Ippolito, N., Braberman, V., Piterman, N., Uchitel, S. Synthesis of live behaviour models for fallible domains. págs. 211–220. 2011. 11

[23] D'Ippolito, N., Fischbein, D., Chechik, M., Uchitel, S. Mtsa: The modal transition system analyser. págs. 475–476. 2008. 13, 24

[24] Piterman, N., Pnueli, A., Sa'ar, Y. Synthesis of reactive(1) designs. tomo 78, págs. 364–380. 2006. 13

[25] Maoz, S., Ringert, J. Gr(1) synthesis for ltl specification patterns. págs. 96–106. 2015. 15

[26] Wolff, E. M., Topcu, U., Murray, R. M. Efficient reactive controller synthesis for a fragment of linear temporal logic. En: 2013 IEEE International Conference on Robotics and Automation, págs. 5033–5040. 2013. 19, 24

[27] Nahabedian, L., Braberman, V., D'Ippolito, N., Honiden, S., Kramer, J., Tei, K., *et al.* Assured and correct dynamic update of controllers. En: 2016 IEEE/ACM 11th International Symposium on Software Engineering for Adaptive and Self-Managing Systems (SEAMS), págs. 96–107. 2016. 19

[28] Kim, J. M., Lim, K. I., Kim, J. H. Auto parking path planning system using modified reeds-shepp curve algorithm. En: 2014 11th International Conference on Ubiquitous Robots and Ambient Intelligence (URAI), págs. 311–315. 2014. 20, 32

[29] Wen, R., Tong, M. Mecanum wheels with astar algorithm and fuzzy pid algorithm based on genetic algorithm. En: 2017 International Conference on Robotics and Automation Sciences (ICRAS), págs. 114–118. 2017. 20

[30] Menghi, C., Tsigkanos, C., Pelliccione, P., Ghezzi, C., Berger, T. Specification patterns for robotic missions. *IEEE Trans. on Soft. Engineering*, págs. 1–1, 2019. 24, 25

[31] Masetti, A., Terissi, L. Prototype robot for computer vision and control systems applications. En: Concurso de Trabajos Estudiantiles - JAIIO 46, págs. 37–46. 2017. 32

[32] Franklin, G., Powell, J., Emami-Naeini, A. Feedback Control Of Dynamic Systems. 1994. 48

[33] Ogata, K. Modern control engineering, 01 2009. 48

[34] Rubio, F., Sánchez, M. Control adaptativo y robusto. Colección Ingeniería. Universidad de Sevilla, 1996. URL https://books.google.com.ar/books?id=54I4mCHvNz8C. 48

[35] University, C. M. The open source framework for 3d rendering and games. URL https://www.panda3d.org, , accedido 2020-03-06. 68

[36] LIENDRO, Tomás. Motion planning - patrulla simple. [YouTube video], May 30 2020. URL https://www.youtube.com/watch?v=V5GNkuVtI1U&list=PLHieTaAXTp06rWJa5CshBJS-w243lsJwK, accedido el Jun. 3, 2020. 69

[37] LIENDRO Tomás. Motion planning - evasión de obstáculos. [YouTube video], May 30 2020. URL https://www.youtube.com/watch?v=1P3v5ddpwuE&list=PLHieTaAXTp06rWJa5CshBJS-w243lsJwK&index=2, accedido el Jun. 3, 2020. 69

[38] LIENDRO, Tomás. Task & motion planning - patrullaje y activación de alarma. [YouTube video], May 30 2020. URL https://www.youtube.com/watch?v=jD2MiOhLWBQ&list=PLHieTaAXTp06rWJa5CshBJS-w243lsJwK&index=3, accedido el Jun. 3, 2020. 70

[39] LIENDRO, Tomás. Task & motion planning - ordenamiento de una caja 1. [YouTube video], May 30 2020. URL https://www.youtube.com/watch?v=dq7S33H0PwY&list=PLHieTaAXTp06rWJa5CshBJS-w243lsJwK&index=4, accedido el Jun. 3, 2020. 71

[40] LIENDRO, Tomás. Task & motion planning - ordenamiento de una caja 2. [YouTube video], May 30 2020. URL https://www.youtube.com/watch?v=xvO_h2oTYUY&list=PLHieTaAXTp06rWJa5CshBJS-w243lsJwK&index=5, accedido el Jun. 3, 2020. 73

[41] LIENDRO, Tomás. Task & motion planning - misión extra: 1. [YouTube video], May 30 2020. URL https://www.youtube.com/watch?v=76rfjYyVByo&list=PLHieTaAXTp06rWJa5CshBJS-w243lsJwK&index=6, accedido el Jun. 3, 2020. 75

[42] LIENDRO, Tomás. Task & motion planning - misión extra: 2. [YouTube video], May 30 2020. URL https://www.youtube.com/watch?v=lBJbQ-dieI0&list=PLHieTaAXTp06rWJa5CshBJS-w243lsJwK&index=7, accedido el Jun. 3, 2020. 76

[43] Jing, G., Tosun, T., Yim, M., Kress-Gazit, H. An end-to-end system for accomplishing tasks with modular robots. En: Proceedings of Robotics: Science and Systems. AnnArbor, Michigan, 2016. 80

[44] Alonso-Mora, J., DeCastro, J., Raman, V., Rus, D., Kress-Gazit, H. Reactive mission and motion planning with deadlock resolution avoiding dynamic obstacles. *Autonomous Robots*, 08 2017. 80

Publicaciones asociadas

1. Liendro, T., Zudaire S. Reporte técnico: Hybrid Control from Scratch: A Design Methodology for Assured Robotic Missions. En: https://arxiv.org/abs/2004.11258. 2020

Agradecimientos

Terminando esta etapa en mi vida no quisiera dejar de agradecer al Instituto Balseiro, a la Universidad Nacional de Cuyo y a la Comisión Nacional de Energía Atómica por darme la oportunidad de formarme en el Instituto tanto a nivel profesional como personal durante estos años de carrera. Hay muchas personas trabajando para que los estudiantes del Instituto podamos disfrutar del aprendizaje a lo largo de nuestros estudios y, entre tantas personas que participan allí, particularmente quisiera destacar a Graciela Bertolino por todo el apoyo durante mis estudios en el IB y a mi codirector, Sebastián, con quien trabajé principalmente durante el último año en este proyecto y quien tuvo la paciencia y dedicación para enseñarme y ayudarme con todos los problemas que fueron surgiendo en este trabajo, que ciertamente no fueron pocos, y con quien pude tener la experiencia de escribir un *paper*. También, quisiera agradecer a Marcelo Kuperman por aceptar ser director en este proyecto, a mis evaluadores, Sol Pedre y Andrés Etchepareborda, y al equipo del pañol del Laboratorio de Ingeniería.

El resultado de este trabajo fue el fruto de mucho esfuerzo y dedicación que no hubiera sido posible sin las personas que me acompañan desde el primer día que decidí comenzar una carrera orientada a la tecnología y la ciencia. En primer lugar, agradezco profundamente a mi papá, Cristóbal, mi mamá, Estela, y mis hermanas, Sofi y Vicky, por el apoyo incondicional en cada una de mis decisiones desde que decidí comenzar mis estudios en Mendoza en la UnCuyo y luego durante mi estadía en Bariloche. Somos una familia muy grande y podría llenar hojas y hojas agradeciendo a cada uno, sin embargo, quiero agradecer en particular a mis tíos Raúl, Nely, Ana y Roberto y a mis abuelas Erna y Florentina por todo el acompañamiento y apoyo a pesar de la distancia, y a mi primo Gastón por la impresión 3D de la carrocería del robot.

Quisiera dedicar un agradecimiento especial para mis amigos del IB Thomás, Seba, Tincho, Santi y Lucas; a Iván quién desde el primer año de la facultad en Mendoza estuvo dándome su apoyo; y a mis amigos Rodri, Fede y Lauti con quienes desde la secundaria seguimos compartiendo miles de historias. Finalmente, quisiera agradecer a Sol por toda la ayuda y apoyo, especialmente en este último periodo de mi carrera, preparando siempre las mejores comidas y dándome la mejor compañía posible.

¡Muchas gracias a todos los que estuvieron conmigo en este camino y que seguramente me van a seguir acompañando por muchos años más!